

Table of Contents

Overview	1
Getting Started	4
Features	8
What's New	12
Demo Projects	18
Component List	24
Hierarchy Chart	26
Compatibility	28
Editions	30
Requirements	32
Installation	33
Deployment	37
Licensing and Subscriptions	38
Getting Support	39
Frequently Asked Questions	40
Using ODAC	45
Connecting in Direct Mode	45
Updating Data with ODAC Dataset Components	47
Master/Detail Relationships	48
Automatic Key Field Value Generation	50
TOraLoader Component	51
TOraTransaction Component	52
TOraQueue, TOraQueueAdmin and TOraQueueTable Components	54
TOraChangeNotification Component	56
BLOB and CLOB Data Types	58
Unicode Character Data	61
Objects	63
XMLTYPE Data Type	65
VARRAY Data Type	68

DML Array	69
PL/SQL Tables	71
Writing Oracle External Procedures with ODAC	73
Transparent Application Failover Support	75
Working in an Unstable Network	77
Disconnected Mode	78
Data Type Mapping	79
Data Encryption	83
Increasing Performance	85
Using Connection Pooling	87
Macros	89
Using Several DAC Products in One IDE	90
DataSet Manager	91
DBMonitor	96
Migration Wizard	97
Writing GUI Applications with ODAC	98
Compatibility with Previous Versions	99
Oracle Package Wizard	100
Integration with Developer Tools	102
dbForge Studio for Oracle	102
OraDeveloper Tools	105
OraTools Add-In	110
64-bit Development with Embarcadero RAD Studio XE2	111
Database Specific Aspects of 64-bit Development	115
Reference	116
BDESession	118
Classes	119
...TBDESession Class	119
Members	119
Properties	122
CRAccess	126
Classes	127
...TCRCursor Class	127
Members	127
Types	128
...TBeforeFetchProc Procedure Reference	128
Enumerations	129
...TCRIsolationLevel Enumeration	129
...TCRTransactionAction Enumeration	129
CRBatchMove	130
Classes	131
...TCRBatchMove Class	131

Members.....	131
Properties.....	132
Methods.....	137
Events.....	137
Types	139
...TCRBatchMoveProgressEvent Procedure Reference.....	139
Enumerations.....	140
...TCRBatchMode Enumeration.....	140
...TCRFieldMappingMode Enumeration.....	140
CRDataTypeMap	141
Classes	142
...EDataMappingError Class.....	142
Members.....	142
...EDataTypeMappingError Class.....	142
Members.....	143
...EInvalidDBTypeMapping Class.....	143
Members.....	143
...EInvalidFieldTypeMapping Class.....	143
Members.....	144
...EUnsupportedDataTypeMapping Class.....	144
Members.....	144
...TMapRule Class.....	144
Members.....	144
Properties.....	145
CREncryption	148
Classes	149
...TCREncryptor Class.....	149
Members.....	149
Properties.....	149
Methods.....	151
Enumerations.....	153
...TCREncDataHeader Enumeration.....	153
...TCREncryptionAlgorithm Enumeration.....	153
...TCRHashAlgorithm Enumeration.....	154
...TCRInvalidHashAction Enumeration.....	154
DAAlerter	155
Classes	156
...TDAAlerter Class.....	156
Members.....	156
Properties.....	157
Methods.....	158
Events.....	159
Types	160
...TAlerterErrorEvent Procedure Reference.....	160
DADump	161
Classes	162
...TDADump Class.....	162
Members.....	162
Properties.....	163
Methods.....	165
Events.....	168
...TDADumpOptions Class.....	170
Members.....	170

Properties.....	170
Types	172
...TDABackupProgressEvent Procedure Reference.....	172
...TDARestoreProgressEvent Procedure Reference.....	172
DALoader	173
Classes	174
...TDAColumn Class.....	174
Members.....	174
Properties.....	174
...TDAColumns Class.....	175
Members.....	176
Properties.....	176
...TDALoader Class.....	176
Members.....	177
Properties.....	178
Methods.....	179
Events.....	181
Types	184
...TDAPutDataEvent Procedure Reference.....	184
...TGetColumnDataEvent Procedure Reference.....	184
...TLoaderProgressEvent Procedure Reference.....	185
DAScript	186
Classes	187
...TDAScript Class.....	187
Members.....	187
Properties.....	188
Methods.....	193
Events.....	196
...TDASTatement Class.....	197
Members.....	198
Properties.....	198
...TDASTatements Class.....	201
Members.....	202
Properties.....	202
Types	203
...TAfterStatementExecuteEvent Procedure Reference.....	203
...TBeforeStatementExecuteEvent Procedure Reference.....	203
...TOnErrorEvent Procedure Reference.....	203
Enumerations.....	205
...TErrorAction Enumeration.....	205
DASQLMonitor	206
Classes	207
...TCustomDASQLMonitor Class.....	207
Members.....	207
Properties.....	208
Events.....	209
...TDBMonitorOptions Class.....	210
Members.....	210
Properties.....	210
Types	212
...TDATraceFlags Set.....	212
...TMonitorOptions Set.....	212
...TOnSQLEvent Procedure Reference.....	212

Enumerations.....	213
...TDATraceFlag Enumeration.....	213
...TMonitorOption Enumeration.....	213
DBAccess	215
Classes	218
...EDAError Class.....	219
Members.....	219
Properties.....	219
...TCRDataSource Class.....	220
Members.....	220
...TCustomConnectDialog Class.....	220
Members.....	221
Properties.....	221
Methods.....	225
...TCustomDAConnection Class.....	226
Members.....	226
Properties.....	227
Methods.....	232
Events.....	242
...TCustomDADataset Class.....	243
Members.....	243
Properties.....	247
Methods.....	262
Events.....	277
...TCustomDASQL Class.....	280
Members.....	280
Properties.....	281
Methods.....	287
Events.....	291
...TCustomDAUpdateSQL Class.....	292
Members.....	292
Properties.....	293
Methods.....	297
...TDAConnectionOptions Class.....	299
Members.....	299
Properties.....	299
...TDADatasetOptions Class.....	301
Members.....	301
Properties.....	302
...TDAEncryptionOptions Class.....	309
Members.....	309
Properties.....	309
...TDAMapRule Class.....	310
Members.....	310
Properties.....	311
...TDAMapRules Class.....	314
Members.....	314
Methods.....	314
...TDAMetaData Class.....	321
Members.....	322
Properties.....	324
Methods.....	326
...TDAParam Class.....	328
Members.....	329

Properties.....	330
Methods.....	335
...TDAParams Class.....	337
Members.....	338
Properties.....	338
Methods.....	338
...TDATransaction Class.....	339
Members.....	340
Properties.....	340
Methods.....	341
Events.....	342
...TMacro Class.....	343
Members.....	344
Properties.....	344
...TMacros Class.....	346
Members.....	346
Properties.....	347
Methods.....	347
...TPoolingOptions Class.....	350
Members.....	350
Properties.....	350
Types.....	352
...TAfterExecuteEvent Procedure Reference.....	352
...TAfterFetchEvent Procedure Reference.....	352
...TBeforeFetchEvent Procedure Reference.....	353
...TConnectionLostEvent Procedure Reference.....	353
...TDAConnectionErrorEvent Procedure Reference.....	353
...TDATransactionErrorEvent Procedure Reference.....	354
...TRefreshOptions Set.....	354
...TUpdateExecuteEvent Procedure Reference.....	354
Enumerations.....	356
...TLabelSet Enumeration.....	356
...TLockMode Enumeration.....	356
...TRefreshOption Enumeration.....	357
...TRetryMode Enumeration.....	357
Variables.....	358
...BaseSQLOldBehavior Variable.....	358
...ChangeCursor Variable.....	358
...MacroChar Variable.....	359
...SQLGeneratorCompatibility Variable.....	359
Devart.Dac.DataAdapter	360
Classes.....	361
...DADataAdapter Class.....	361
Members.....	361
Properties.....	362
Methods.....	362
Devart.Odac.DataAdapter	364
Classes.....	365
...OraDataAdapter Class.....	365
Members.....	365
MemData	367
Classes.....	368
...TAttribute Class.....	368

Members.....	368
Properties.....	369
...TBlob Class.....	372
Members.....	373
Properties.....	373
Methods.....	375
...TCompressedBlob Class.....	379
Members.....	380
...TDBObject Class.....	381
Members.....	381
...TObjectType Class.....	381
Members.....	382
Properties.....	382
Methods.....	384
...TSharedObject Class.....	385
Members.....	386
Properties.....	386
Methods.....	386
Types	388
...TLocateExOptions Set.....	388
...TUpdateRecKinds Set.....	388
Enumerations.....	389
...TConnLostCause Enumeration.....	389
...TDANumericType Enumeration.....	390
...TLocateExOption Enumeration.....	390
...TSortType Enumeration.....	390
...TUpdateRecKind Enumeration.....	391
MemDS	392
Classes	393
...TMemDataSet Class.....	393
Members.....	393
Properties.....	394
Methods.....	398
Events.....	408
Variables	411
...DoNotRaiseExcetionOnUaFail Variable.....	411
...SendDataSetChangeEventAfterOpen Variable.....	411
OdacVcl	412
Classes	413
...TConnectDialog Class.....	413
Members.....	413
Properties.....	414
Ora	417
Classes	420
...TBFileField Class.....	421
Members.....	421
Properties.....	422
Methods.....	424
...TCursorField Class.....	424
Members.....	425
Properties.....	425
...TCustomOraQuery Class.....	426
Members.....	426

...ToraChangeNotification Class	432
Members.....	432
Properties.....	433
Methods.....	435
Events.....	435
...ToraDataSet Class	436
Members.....	436
Properties.....	442
Methods.....	456
...ToraDataSetField Class.....	467
Members.....	467
Properties.....	468
...ToraDataSetOptions Class.....	468
Members.....	468
Properties.....	470
...ToraDataSetOptionsDS Class.....	476
Members.....	477
Properties.....	478
...ToraDataSource Class	482
Members.....	482
...ToraEncryptor Class.....	482
Members.....	483
...ToraIntervalField Class.....	483
Members.....	483
Properties.....	484
...ToraMetaData Class.....	485
Members.....	486
...ToraNestedTable Class.....	487
Members.....	488
Properties.....	489
...ToraNumberField Class.....	491
Members.....	492
Properties.....	492
...ToraParam Class.....	492
Members.....	493
Properties.....	495
Methods.....	505
...ToraParams Class.....	507
Members.....	508
Properties.....	508
...ToraPoolingOptions Class.....	508
Members.....	509
Properties.....	509
...ToraQuery Class.....	510
Members.....	511
Properties.....	517
...ToraReferenceField Class.....	523
Members.....	524
Properties.....	524
...ToraSession Class	524
Members.....	525
Properties.....	527
Methods.....	539
Events.....	545
...ToraSessionOptions Class.....	547

Members.....	547
Properties.....	548
...TOraSQL Class.....	554
Members.....	554
Properties.....	556
Methods.....	560
...TOraStoredProc Class.....	563
Members.....	564
Properties.....	570
Methods.....	576
...TOraTimeStampField Class.....	582
Members.....	583
Properties.....	583
...TOraTrace Class.....	584
Members.....	584
Properties.....	585
Methods.....	587
...TOraUpdateSQL Class.....	591
Members.....	592
...TOraXMLField Class.....	593
Members.....	593
Properties.....	593
Types.....	595
...TConnectChangeEvent Procedure Reference.....	595
...TFailoverEvent Procedure Reference.....	595
...TOraChangeNotificationEvent Procedure Reference.....	596
...TPISqlTraceMode Set.....	596
...TSqlTraceMode Set.....	596
Enumerations.....	597
...TCheckMode Enumeration.....	597
...TFailoverState Enumeration.....	597
...TFailoverType Enumeration.....	598
...TOraIsolationLevel Enumeration.....	598
...TRefreshMode Enumeration.....	598
...TSequenceMode Enumeration.....	599
Routines.....	600
...AddWhere Function.....	600
...DefaultSession Function.....	600
...DeleteWhere Function.....	601
...GetOrderBy Function.....	601
...SessionByName Function.....	601
...SetFieldList Function.....	602
...SetGroupBy Function.....	602
...SetOrderBy Function.....	602
...SetSubscriptionPort Procedure.....	603
...SetTableList Function.....	603
...SetWhere Function.....	603
Variables.....	604
...DefSession Variable.....	604
...OraQueryCompatibilityMode Variable.....	604
...Sessions Variable.....	604
...UseDefSession Variable.....	605
Constants.....	606
...OdacVersion Constant.....	606

OraAlerter	607
Classes	608
...TOraAlerter Class	608
Members	608
Properties	609
Methods	612
Events	616
Types	618
...TOnEventEvent Procedure Reference	618
...TOnTimeOutEvent Procedure Reference	618
Enumerations	619
...TEventType Enumeration	619
OraAQ	620
Classes	622
...TDequeueOptions Class	622
Members	623
Properties	623
...TEnqueueOptions Class	626
Members	627
Properties	627
...TOraQueue Class	629
Members	630
Properties	630
Methods	633
Events	638
...TOraQueueAdmin Class	639
Members	639
Properties	641
Methods	644
...TOraQueueTable Class	657
Members	657
Properties	658
Methods	663
...TQueueAgent Class	668
Members	668
Properties	669
...TQueueAgents Class	670
Members	670
Properties	671
Methods	671
...TQueueMessage Class	672
Members	672
Properties	673
...TQueueMessageProperties Class	674
Members	675
Properties	675
Types	680
...TQueueMessageEvent Procedure Reference	680
Enumerations	681
...TDequeueMode Enumeration	681
...TQueueDeliveryMode Enumeration	681
...TQueueMessageGrouping Enumeration	682
...TQueueMessageState Enumeration	682
...TQueueNavigation Enumeration	682

...TQueueSequenceDeviation Enumeration.....	683
...TQueueSortList Enumeration.....	683
...TQueueType Enumeration.....	684
...TQueueVisibility Enumeration.....	684
OraCall	685
Routines	686
...DetectOCI Procedure.....	686
...FreeOCI Procedure.....	686
...GetSubscriptionPort Function.....	686
...InitOCI Procedure.....	687
...LoadedOCI Function.....	687
...LoadOCI Procedure.....	687
...OraError Procedure.....	687
Variables	688
...OCICallStyle Variable.....	688
...OCIDLL Variable.....	688
...OCIThreaded Variable.....	689
...OCIVersion Variable.....	689
...OCIVersionSt Variable.....	689
...OracleErrorMaxLength Variable.....	689
OraClasses	690
Classes	691
...TOraCursor Class.....	691
Members.....	691
Properties.....	692
Methods.....	693
...TOraFile Class.....	694
Members.....	695
Properties.....	696
Methods.....	698
...TOraInterval Class.....	702
Members.....	702
Properties.....	703
Methods.....	706
...TOraLob Class.....	708
Members.....	709
Properties.....	710
Methods.....	712
...TOraNumber Class.....	716
Members.....	717
Properties.....	717
Methods.....	720
...TOraTimeStamp Class.....	721
Members.....	721
Properties.....	722
Methods.....	726
Enumerations.....	730
...TConnectMode Enumeration.....	730
...TMessageType Enumeration.....	730
...TOptimizerMode Enumeration.....	731
Variables	732
...FloatPrecision Variable.....	732
...IntegerPrecision Variable.....	732
...LargeIntPrecision Variable.....	733

OraConnectionPool	734
Enumerations	735
...ToraPoolingType Enumeration	735
OraErrHand	736
Classes	737
...ToraErrorHandler Class	737
Members	737
Properties	738
Methods	739
Events	740
Types	741
...TOnOraErrorEvent Procedure Reference	741
OraError	742
Classes	743
...EOraError Class	743
Members	743
Properties	743
OraLoader	745
Classes	746
...TDPColumn Class	746
Members	746
Properties	747
...ToraLoader Class	748
Members	749
Properties	750
Events	751
Types	754
...TDPErrEvent Procedure Reference	754
...TDPGetColumnDataEvent Procedure Reference	754
...TDPPutDataEvent Procedure Reference	755
Enumerations	756
...TDPErrAction Enumeration	756
...TLoadMode Enumeration	756
OraObjects	757
Classes	758
...ToraArray Class	758
Members	758
Properties	761
Methods	766
...ToraNestTable Class	769
Members	769
Methods	771
...ToraObject Class	774
Members	775
Properties	776
Methods	782
...ToraRef Class	787
Members	788
Properties	790
Methods	792
...ToraType Class	795
Members	796

Properties.....	796
Methods.....	798
...TOraXML Class.....	799
Members.....	800
Properties.....	802
Methods.....	804
OraPackage	813
Classes	814
...TCustomOraPackage Class.....	814
Members.....	814
Properties.....	814
Methods.....	815
...TOraPackage Class.....	817
Members.....	818
Properties.....	818
OraProvider	820
Classes	821
...TOraProvider Class.....	821
Members.....	821
OraScript	822
Classes	823
...TOraScript Class.....	823
Members.....	823
Properties.....	824
...TOraStatement Class.....	827
Members.....	827
Properties.....	828
...TOraStatements Class.....	829
Members.....	829
Properties.....	829
OraSmart	831
Classes	832
...TCustomSmartQuery Class.....	832
Members.....	833
Properties.....	838
Methods.....	846
Events.....	852
...TOraTable Class.....	858
Members.....	859
Properties.....	865
Methods.....	872
...TSmartQuery Class.....	878
Members.....	879
...TSmartQueryOptions Class.....	885
Members.....	885
Enumerations.....	888
...TSmartState Enumeration.....	888
OraSQLMonitor	889
Classes	890
...TOraSQLMonitor Class.....	890
Members.....	890
OraTransaction	892

Classes	893
...TOraTransaction Class.....	893
Members.....	893
Properties.....	895
Methods.....	899
Events.....	903
Enumerations.....	905
...TGlobalCoordinator Enumeration.....	905
VirtualTable	906
Classes	907
...TVirtualTable Class.....	907
Members.....	907
Properties.....	909
Methods.....	910
Types	917
...TVirtualTableOptions Set.....	917
Enumerations.....	918
...TVirtualTableOption Enumeration.....	918

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, Mac OS X, Linux, and FreeBSD 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 <http://devart.com/odac/>

Table of Contents

1. [Advantages of ODAC Technology](#)
 - [Wide Coverage of Oracle Features](#)
 - [Native Connection Options](#)
 - [Oracle Advanced Features Support](#)
 - [Cross-Platform Solution for Delphi, C++Builder, and Lazarus](#)
 - [Optimized Code](#)
 - [Compatibility with Other Connectivity Methods](#)
 - [Development and Support](#)
2. [Key Features](#)
3. [How does ODAC work?](#)

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](#). Get a full list of supported Oracle features in the [Features](#) topic.

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, Mac OS X, Linux, and FreeBSD 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 Mac OS X.

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 the [Getting Support](#) topic. 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, VCL.NET, and CLX versions of library available
- Support of Windows, Mac OS X, Linux, and FreeBSD 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
- Integration with [OraDeveloper Tools](#) for performing advanced database development and administration tasks
- Easy migration from BDE with [Migration Wizard](#)
- Lets you [use Professional Edition of Delphi, C++Builder, and Kylix](#) 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 you can find in the [Features](#) topic.

How does ODAC work?

ODAC allows you to connect to Oracle in two ways: in Client mode, using Oracle Client software, or in Direct mode, over TCP/IP. The chosen connection mode is regulated by the

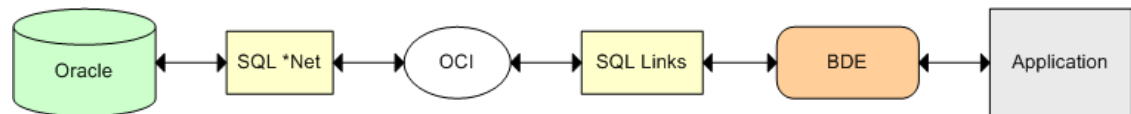
[ODAC Direct mode](#)

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 Direct mode, ODAC connects to Oracle directly without using Oracle client software. In this mode, ODAC requires only TCP/IP support, and lets you build really thin database application.

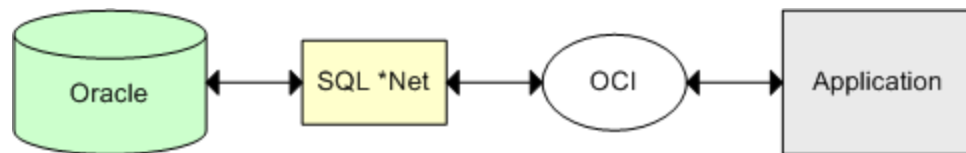
In comparison, the Borland Database Engine (BDE) uses several layers to access Oracle, and requires additional data access software to be installed on client machines.

The BDE data transfer protocol is shown below.



BDE Connection Protocol

Using ODAC in Client mode allows your application to connect to OCI directly.



ODAC Connection Flow Client Mode

Using ODAC in Direct mode provides the optimal transfer route.



ODAC Connection Flow Direct Mode

© 1997-2011 Devart. All Rights Reserved.

© 1997-2012 Devart. All Rights Reserved.

2 Getting Started

This page contains a quick introduction to setting up and using the Oracle Data Access Components library. It gives a walkthrough for each part of the ODAC usage process and points out the most relevant related topics in the documentation.

- [What is ODAC?](#)
- [How does ODAC work?](#)
- [Installing ODAC.](#)
- [Working with the ODAC demo projects.](#)
- [Compiling and deploying your ODAC project.](#)
- [Using the ODAC documentation.](#)
- [How to get help with ODAC.](#)

What is ODAC?

Oracle Data Access Components (ODAC) is a component library which provides direct connectivity to Oracle for Delphi, Delphi for .NET, C++Builder, and Kylix, and helps you to develop fast Oracle-based database applications with these environments.

Many ODAC classes are based on VCL, VCL for .NET, and CLX classes and interfaces. ODAC is a replacement for the [Borland Database Engine](#), it provides native database connectivity, and is specifically designed as an interface to the Oracle database.

The introduction to ODAC is provided in the [Overview](#) section.

The list of the ODAC features you may find useful is listed in the [Features](#) section.

The overview of the ODAC component classes is provided in the [Components List](#) section.

How does ODAC work?

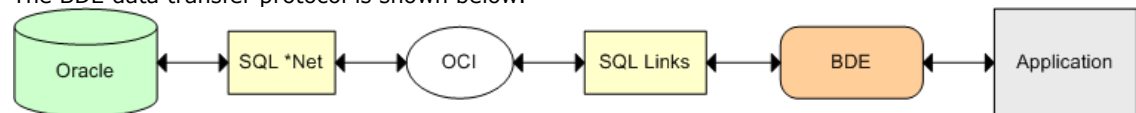
ODAC allows you to connect to Oracle in two ways: in Client mode, using Oracle Client software, or in Direct mode, over TCP/IP. The chosen connection mode is regulated by the [ODAC Direct mode](#).

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 Direct mode, ODAC connects to Oracle directly without using Oracle client software. In this mode, ODAC requires only TCP/IP support, and lets you build really thin database application.

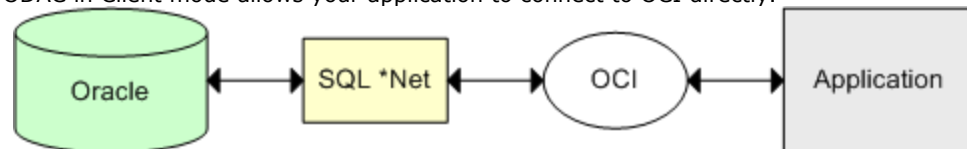
In comparison, the Borland Database Engine (BDE) uses several layers to access Oracle, and requires additional data access software to be installed on client machines.

The BDE data transfer protocol is shown below.



BDE Connection Protocol

Using ODAC in Client mode allows your application to connect to OCI directly.



ODAC Connection Flow Client Mode

Using ODAC in Direct mode provides the optimal transfer route.

**ODAC Connection Flow Direct Mode****Installing ODAC**

To install ODAC, complete the following steps.

1. Choose and download the version of the ODAC installation program that is compatible with your IDE. For instance, if you are installing ODAC 6.00, you should use the following files:

For BDS 2006 and Turbo - **odac600d10*.exe**

For Delphi 7 - **odac600d7*.exe**

For more information, visit the [ODAC download page](#).

2. Close all running IDEs.
3. Launch the ODAC installation program you've downloaded and follow the instructions.

By default, the ODAC installation program should install compiled ODAC libraries automatically on all IDEs except for Kylix. To view the installation instructions for Kylix click [here](#).

To check if ODAC has been installed properly, launch your IDE and make sure that Oracle Access page has been added to the Component palette and that Oracle menu was added to the Menu bar.

If you have bought ODAC Professional Edition with Source Code or ODAC Developer Edition with Source Code, you will be able to download both the compiled version of ODAC and the ODAC source code. The installation process for the compiled version is standard. The ODAC source code must be compiled and installed manually. For more details, consult the supplied *ReadmeSrc.txt* file.

To find out what gets installed with ODAC or to troubleshoot your ODAC installation, visit the [Installation](#) topic.

Working with the ODAC demo projects

The ODAC installation package includes a number of demo projects that demonstrate ODAC capabilities and use patterns. The ODAC demo projects are automatically installed in the ODAC installation folder. To get started working with ODAC quickly, launch and explore the introductory ODAC demo project, *OdacDemo*, from your IDE. This demo project is a collection of demos that show how ODAC can be used. The project creates a form which contains an explorer panel for browsing the included demos and a view panel for launching and viewing the selected demo.

OdacDemo Walkthrough

1. Launch your IDE.
2. From the menu bar choose File | Open Project
3. Find the ODAC directory and open the *OdacDemo* project. This project should be located in the Demos\OdacDemo folder.
For example, if you are using Borland Developer Studio 2006, the demo project may be found at \Program Files\Devart\ODAC for Delphi 2006\Demos\Win32\OdacDemo\OdacDemo.bdsproj
4. Select Run | Run or press F9 to compile and launch the demo project. *OdacDemo* should start, and a full-screen ODAC Demo window with a toolbar, an explorer panel, and a view panel will open. The explorer panel will contain a list of all the demo sub-projects included in *OdacDemo*, and the view panel will contain an overview of each included demo.
At this point, you will be able to browse through the available demos, read their descriptions, view their source code, and see the functionality provided by each demo for interacting with Oracle. However, you will not be able to actually retrieve data from Oracle or execute commands until you connect to the database.
5. Choose your method of connecting to the Oracle server. To connect in Client mode, leave the Direct checkbox in the toolbar unselected. To connect to Oracle in Direct mode over TCP/IP, select the Direct checkbox.
6. Click on the "Connect" button in the *OdacDemo* toolbar. A Connect dialog box will open. Enter the connection parameters you use to connect to your Oracle server and click "Connect" in the dialog box.

When connecting in Client mode, enter your Oracle username, password, and the name of your server
(e.g., *user/pwd/ora*).

When connecting in Direct mode, enter your Oracle username, password, and the full [Host:Port:SID](#) network path to your server.

e.g., *user/pwd/db:1521:orc920*

Note: For this step to work properly when connecting in Client mode, you must have the

Oracle Client installed and properly configured *tnsnames.ora* file. To connect in Direct mode, you need to be using a version of ODAC with Direct connectivity support. For more information, see the [ODAC Feature Matrix](#).

Now you have a fully functional interface to your Oracle server. You will be able to go through the different demos, browse tables, create and drop objects, and execute PL/SQL commands.

Warning! All changes you make to the database you are connected to, including creating and dropping objects used by the demo, will be permanent. Make sure to specify a test database in the connection step.

7. To create any object that will be used by *OdacDemo*, click the "Create" button. If some of these objects already exist in the database you have connected to, the following error message will appear.

"An error has occurred: ORA00955: name is already being used by an existing object. ... Ignore this exception?"

This is a standard warning from the object execution script. Click "Yes to All" to ignore this message. *OdacDemo* will create the *OdacDemo* objects on the server you have connected to.

8. Choose a demo that demonstrates an aspect of working with Oracle that you are interested in, and play with the demo frame in the view window on the right. For example, to find out more about working with Oracle tables, select the Table demo from the "Working with Components" folder. A simple Oracle table browser will open in the view panel that will let you open a table in your database by specifying its name and clicking on the "Open" button.
9. To find out how the demo you've selected was implemented click on the "Demo source" button in the *OdacDemo* toolbar. The source code behind the demo project will appear in the view panel. Try to find the places where ODAC components are used to connect to the database.
10. Click on the "Form as text" button in the *OdacDemo* toolbar to view the code behind the interface to the demo. Try to find the places where ODAC components are created on the demo form.
11. Repeat these steps for other demos listed in the explorer window. Available demos are organized in three folders.

Working with components

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

General demos

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

Oracle-specific demos

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

12. When you have finished working with the project, click on the "Drop" button in the *OdacDemo* toolbar to remove all the schema objects added in Step 7.

Other ODAC demo projects

ODAC is accompanied by a number of other demo projects. The description of all ODAC demos is located in the [Demo Projects](#) topic.

Compiling and deploying your ODAC project

Compiling ODAC-based projects

By default, to compile a project that uses ODAC classes, your IDE compiler needs to have access to the ODAC dcu (obj) files. If you are compiling runtime packages, the compiler will also need to have access to the ODAC bpl files. **All appropriate settings for both these scenarios should take place automatically during the installation of ODAC.** If you are using one of the ODAC editions that come with source code - ODAC Professional Edition with Source Code or ODAC Developer Edition with Source Code, you should just need to modify your environment manually.

You can check if your environment is properly configured by trying to compile one of the ODAC demo projects. If you have no problem compiling and launching the ODAC demos, your environment has been configured properly.

For more information on which library files and environment changes needed for compiling ODAC-based projects, consult the [Installation](#) topic.

Deploying ODAC-based projects

To deploy an application that uses ODAC, you will need to make sure the target workstation has access

to the following files.

- The Oracle Client software, if connecting in Client mode.
- The ODAC bpl files, if compiling with runtime packages.
- The ODAC assembly files, if using VCL for .NET components.

If you are evaluating deploying projects with ODAC Trial Edition, you will also need to deploy some additional .bpl files with your application even if you are compiling without runtime packages. As another trial limitation for C++Builder, applications written with ODAC Trial Edition for C++Builder will only work if the C++Builder IDE is launched. More information about ODAC Trial Edition limitations is provided [here](#).

The list of the files which may need to be deployed with ODAC-based applications is included in the [Deployment](#) topic.

Using the ODAC documentation

The ODAC documentation describes how to install and configure ODAC, how to use ODAC Demo Projects, and how to use the ODAC libraries.

The ODAC documentation includes a detailed reference of all the ODAC components and classes. Many of the ODAC components and classes inherit or implement members from other VCL, VCL for .NET, CLX classes and interfaces. The product documentation also includes the summary of all members within each of these classes. To view a detailed description of a particular component, look it up in the [Components List](#) section. To find out more about the specific standard VCL/CLX class the ODAC component is inherited from, see the corresponding topic in your IDE documentation.

At installation time, the ODAC documentation is integrated into your IDE. It can be invoked from the Oracle menu added to the Menu Bar, or by pressing F1 in the object inspector or on a selected code segment.

How to get help with ODAC

There are a number of resources for finding help on using ODAC classes in your project.

-

If you have any questions about ODAC installation or licensing, consult [Licensing](#) and [FAQ](#) section.

-

You can get community assistance and ODAC technical support on the [ODAC Support Forum](#).

-

- To get help through the ODAC [Priority Support](#) program, send an email to the ODAC development team at odac@devart.com.
- If you have any questions about ordering ODAC or any other Devart product, contact sales@devart.com.

For more information, consult the [Getting Support](#) topic.

3 Features

Table of Contents

1. [General usability](#)
2. [Network and connectivity](#)
3. [Compatibility](#)
4. [Oracle technology support](#)
5. [Oracle data types](#)
6. [Performance](#)
7. [Local data storage operations](#)
8. [Data access and data management automation](#)
9. [Extended data access functionality](#)
10. [Data exchange](#)
11. [Script execution](#)
12. [SQL execution monitoring](#)
13. [Visual extensions](#)
14. [Design-time enhancements](#)
15. [Product clarity](#)
16. [Licensing and support](#)

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
- OS authentication
- DBA privileges to open a session with
- Proxy Authentication support
- Support for the change expired password
- Connection with using Service Name or SID in the Direct mode

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
- [Compatible with all IDE versions starting with Delphi 5, C++Builder 5, Free Pascal, and Kylix 2 \(except Delphi 8\)](#)
- Includes provider for UniDAC Standard Edition
- [Wide reporting component support](#), including support for InfoPower, ReportBuilder, FastReport
- Support of all standard and third-party visual data-aware controls
- Delphi 5, 6, 7, C++ Builder 5, 6, Borland Delphi Studio 2005, 2006, Code Gear RAD Studio 2007, 2009, Embarcadero RAD Studio 2010, XE, XE2 support
- Lazarus 0.9.30.4 and FPC 2.6.0 for Windows and Unix support
- 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 [TOraAlterter](#) 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

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](#)
- Nested table 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

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

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
- [OraDataAdapter](#) component for WinForms and ASP.NET applications
- 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

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
- Integration with [OraDeveloper Tools](#) for browsing database schemas, manipulating database objects and visual building of queries
- Automatic design-time component linking
- [OraTools Add-in](#) extends design-time capabilities and lets you build and test queries, design and debug PL/SQL blocks and explore database schemas
- Easy migration from BDE with [Migration Wizard](#)
- More convenient data source setup with the [TOraDataSource](#) component
- [Syntax highlighting](#) in design-time editors

Product clarity:

- Complete documentation sets
- Documentation in CHM, PDF and Microsoft Help formats
- [A large amount of helpful demo projects](#)

Licensing and support:

- Included annual [ODAC Subscription](#)

with
[Priority Support](#)

- Licensed royalty-free per developer, per team, or per site

©

1997-2011 Devart. All Rights Reserved.

©

1997-2012 Devart. All Rights Reserved.

4 What's New

05-Sep-12 New Features in ODAC 8.5:

- Rad Studio XE3 is supported
- Windows 8 is supported

24-Jul-12 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

23-Nov-11 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

15-Sep-11 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

28-Apr-11 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

26-Jan-11 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

13-Sep-10 New Features in Oracle Data Access Components 7.00:

- Embarcadero RAD Studio XE supported
- 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

10-Sep-09 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

02-Apr-09 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

23-Oct-08 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

23-May-08 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

14-Nov-07 New Features in Oracle Data Access Components 6.25:

- Oracle 11g supported

27-Sep-07 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

12-Jun-07 New Features in Oracle Data Access Components 6.10:

- C++Builder 2007 supported

16-May-07 New Features in Oracle Data Access Components 6.05:

- Added [Oracle Package Wizard](#) that simplifies working with PL/SQL Packages

22-Mar-07 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 [import](#) 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](#)

26-Jan-06 New Features in Oracle Data Access Components 5.70:

- Delphi 2006 supported

31-May-05 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)

24-Jan-05 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

30-Jun-04 New Features in Oracle Data Access Components 5.10:

- Local sort ability using IndexFieldNames property added
- [OraDataAdapter](#) component for Delphi 8 added

09-Apr-04 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

08-Oct-03 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

05-Jun-03 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

30-Jan-03 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

30-Sep-02 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.

30-Aug-02 New Features in Oracle Data Access Components 3.90:

- Kylix 3 support

09-Aug-02 New Features in Oracle Data Access Components 3.85:

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

18-Jul-02 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

21-Mar-02 New Features in Oracle Data Access Components 3.60:

- supports C++Builder 6

14-Dec-01 New Features in Oracle Data Access Components 3.50:

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

12-Oct-01 New Features in Oracle Data Access Components 3.30:

- supports Oracle 9i

- Net edition for Kylix

08-Aug-01 New Features in Oracle Data Access Components 3.20:

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

20-Feb-01 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

11-Jul-00 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

30-Mar-00 New Features in Oracle Data Access Components 2.20:

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

10-Jan-00 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

14-Oct-99 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

26-May-99 New Features in Oracle Data Access Components 1.85:

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

12-Mar-99 New Features in Oracle Data Access Components 1.75:

- TOraTable component
- TStoredProc component

01-Mar-99 New Features in Oracle Data Access Components 1.70:

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

05-Feb-98 New Features in Oracle Data Access Components 1.50:

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

5 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, Borland Developer Studio, and Kylix. 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 with .NET support, the structure will be as following.

```
Demos
|
|--dotNet
|   |--OdacDemo [ .NET version of the main ODAC demo project ]
|   |--Miscellaneous
|       |-- [Some other .NET demo projects]
|
|--Win32
    |--OdacDemo [Win32 version of 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 Win32 demo projects on design technologies]
```

In Delphi 5, 6, 7, C++Builder 5, 6, Kylix, , and FreePascal .NET is not supported, and the root directories is omitted. For these IDEs you will see the following structure.

```
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.
BDESession	Demonstrates how to integrate ODAC with an existing BDE connection using the TBDESession ODAC component.
ChangeNotification	Demonstrates how to subscribe, receive, and reflect DML or DDL changes on objects associated with queries. Note: Requires CHANGE NOTIFICATION privilege. This functionality 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
dotNet/		folder appears only for IDEs with support for .NET














Miscellaneous	AspNet	OraDataAdapter to create a simple ASP .NET application. This demo shows how to create an ASP. NET application that lets you connect to a database and execute queries. Application displays query results in DataGrid and sends user changes back to the database.
	WinForms	Shows how to use ODAC to create WinForms application. This demo project creates simple WinForms application and fills data grid from OraDataAdapter data source.
OdacDemo	OdacDemo	<i>.NET version of the main ODAC demo project - see above</i>
Win32/		<i>folder appears only for IDEs with support for .NET. For all other IDEs contents appear in root</i>
	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.
ThirdParty		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.
	IntraWeb	
	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.

	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.
Miscellaneous	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




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 Oracle Access 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	Provides dataset functionality for data that has no real database connection. This component is placed on the Data Access page of the Component palette, not on the Oracle Access page.
	OraDataAdapter	.NET component for transferring data between TDataSets and System.Data.DataSets.

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.

	TOracleLoader	Allows you to quickly load data into Oracle databases.
	TOracleTransaction	Provides discrete transaction control over sessions. Can be used to manipulate both simple and distributed transactions.
	TOracleQueue	Lets you monitor the message queue. Provides an interface for enqueueing and dequeueing messages.
	TOracleQueueTable	Component for managing queue tables.
	TOracleQueueAdmin	Component for managing queues.
	TOracleChangeNotification	Allows you to use Oracle Database Change Notifications.
	TOracleTrace	Allows you to start and stop the SQL trace for a specified session. This component provides access to the DBMS_TRACE package.
	TOracleErrorHandler	Translates error messages.
	TOracleProvider	Loads data to and from a dataset.
	TOracleBDESession	Integrates ODAC components into BDE-based applications.
	TOracleMetaData	Retrieves metadata on specified SQL object.
	TOracleBatchMove	Transfers data between all types of TDataSet descendants. This component is placed on the Data Access page of the Component palette, not on the Oracle Access page.

7 Hierarchy Chart

Many ODAC classes are inherited from standard VCL/CLX 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
|   |   |   |   |   |-TBDESession
|   |   |-TDataSet
|   |   |   |-TMemDataSet
|   |   |   |   |-TCustomDADataSet
|   |   |   |   |   |-TOraDataSet
|   |   |   |   |   |   |-TCustomOraQuery
|   |   |   |   |   |   |   |-TCustomSmartQuery
|   |   |   |   |   |   |   |   |-TOraTable
|   |   |   |   |   |   |   |   |   |-TSmartQuery
|   |   |   |   |   |   |   |   |   |   |-TOraQuery
|   |   |   |   |   |   |   |   |   |   |   |-TOraStoredProc
|   |   |   |   |   |   |   |   |   |   |   |   |-TOraNestedTable
|   |   |   |   |   |   |   |   |   |   |   |   |   |-TDAMetaData
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |-TOraMetaData
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |-TVirtualTable
|   |   |-TDataSource
|   |   |   |-TCRDataSource
|   |   |   |   |-TOraDataSource
|   |-DADDataAdapter
|   |   |-OraDataAdapter
|   |-TCRBatchMove
|   |-TCustomConnectDialog
|   |   |-TConnectDialog
|   |-TCustomDASQL
|   |   |-TOraSQL
|   |-TCustomDASQLMonitor
|   |   |-TOraSQLMonitor
|   |-TDALoader
|   |   |-TOraLoader
|   |-TDAScript
|   |   |-TOraScript
|   |-TDAAlerter
|   |   |-TOraAlerter
|   |-TCREncryptor
|   |   |-TOraEncryptor
|   |-TOraChangeNotification
|   |-TOraErrorHandler
|   |-TOraPackage
|   |-TOraQueue
|   |-TOraQueueAdmin
|   |-TOraQueueTable
|   |-TOraTrace
|   |-TOraTransaction
|-TSharedObject
|   |-TBlob
|   |   |-TCompressedBlob
|   |   |   |-TOraLob
|   |   |   |   |-TOraFile
|   |-TOraObject
|   |   |-TOraArray
|   |   |   |-TOraNestTable
|   |   |   |   |-TOraRef

```

- | | [ToraXML](#)
- | [ToraCursor](#)
- | [ToraInterval](#)
- | [ToraNumber](#)
- | [ToraTimeStamp](#)

8 Compatibility

Oracle Compatibility

ODAC supports Oracle servers 11g, 10g, 9i, 8i, 8.0, and 7.3, including Oracle Express Edition 11g and 10g.

ODAC supports both x86 and x64 versions of the following Oracle Clients: 11g, 10g, 9i, 8i, 8.0, and 7.3. 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 XE3
 - Embarcadero Delphi XE3 for Win32
 - Embarcadero Delphi XE3 for Win64
 - Embarcadero Delphi XE3 for OSX32
 - Embarcadero C++Builder XE3 for Win32
 - Embarcadero C++Builder XE3 for OSX32
- Embarcadero RAD Studio XE2 (Requires [Update 4 Hotfix 1](#))
 - Embarcadero Delphi XE2 for Win32
 - Embarcadero Delphi XE2 for Win64
 - Embarcadero Delphi XE2 for OSX32
 - Embarcadero C++Builder XE2 for Win32
 - Embarcadero C++Builder XE2 for OSX32
- 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 for Win32
 - CodeGear Delphi 2007 for .NET
 - CodeGear C++Builder 2007
- Borland Developer Studio 2006
 - Borland Delphi 2006 for Win32
 - Borland Delphi 2006 for .NET
 - Borland C++Builder 2006
- Turbo Delphi Professional
- Turbo Delphi for .NET Professional
- Turbo C++ Professional
- Borland Delphi 2005
- Borland Delphi 7
- Borland Delphi 6 (Requires [Update Pack 2](#) – Delphi 6 Build 6.240)
- Borland Delphi 5
- Borland C++Builder 6 (Requires [Update Pack 4](#) – C++Builder 6 Build 10.166)
- Borland C++Builder 5
- [Lazarus](#) 0.9.30.4 and [Free Pascal](#) 2.6.0 for Windows, Linux, Mac OS X, FreeBSD for 32-bit and 64-bit platforms

Only Architect, Enterprise, and Professional IDE editions are supported. For Delphi XE/XE2/XE3, C++Builder XE/XE2/XE3 ODAC additionally supports Starter Edition.

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

Direct mode is available only for Delphi and C++Builder IDEs.

Supported Target Platforms

- Windows, 32-bit and 64-bit
- Mac OS X
- Linux, 32-bit and 64-bit (only in Lazarus and Free Pascal)
- FreeBSD (only in Lazarus and Free Pascal)

Note that support for 64-bit Windows was introduced in Delphi XE2, and is not available in C++Builder and older versions of Delphi. Support for Mac OS X was introduced in Delphi XE2 and C++Builder XE2, and is not available in older versions of Delphi and C++Builder.

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.

OraDeveloper Tools Compatibility

The current version of ODAC is compatible with OraDeveloper Tools 2.xx and higher versions for RAD Studio 2007 - XE2

dbForge Studio For Oracle

The current version of ODAC is compatible with dbForge Studio For Oracle 3.00 and higher versions for CodeGear RadStudio 2007 and higher

OraTools Add-in

The current version of ODAC is compatible with OraTools Add-in 2.65 and higher versions for all versions of Delphi

9 Editions

Oracle Data Access Components comes in four basic edition levels: ODAC Standard Edition, ODAC Professional Edition, ODAC Developer Edition, and ODAC Trial Edition.

ODAC Standard Edition includes the ODAC basic connectivity components and the ODAC Migration Wizard. ODAC Standard Edition is a cost-effective solution for database application developers who need general connectivity functionality for Oracle. ODAC Professional Edition shows off the full power of ODAC, enhancing ODAC Standard Edition with support for Oracle-specific functionality, access to Direct mode for connecting to the Oracle server directly over TCP/IP, and some advanced connection management features.

ODAC Developer Edition is a bundle package of ODAC Professional Edition with OraDeveloper Tools, an advanced add-in for Oracle database development and administration. ODAC Developer Edition is the best choice for overall.

ODAC Trial Edition is the evaluation version of ODAC. It includes all the functionality of ODAC Professional Edition with a trial limitation of 60 days. Kylix, C++Builder, and supported .NET IDEs have additional trial limitations.

You can get Source Access to the Client mode implementation of all the component classes in ODAC by purchasing the special ODAC Professional Edition with Source Code or ODAC Developer Edition with Source Code**.

For more information about getting the ODAC edition you want, visit the How to Order section.

ODAC Edition Matrix

Feature	Developer**	Professional**	Standard	Trial*
---------	-------------	----------------	----------	--------

Base Components

[TOraSession](#)

[TOraQuery](#)

[TSmartQuery](#)

[TOraSQL](#)

[TOraTable](#)

[TOraStoredProc](#)

[TOraUpdateSQL](#)

[TOraNestedTable](#)

[TOraDataSource](#)

[TOraScript](#)

[TOraSQLMonitor](#)

[TConnectDialog](#)

[TVirtualTable](#)

TCRDBGrid

[OraDataAdapter](#)

+

+

+

+

Additional Components[TOraEncryptor](#)[TOraPackage](#)[TOraAlerter](#)[TOraLoader](#)[TOraTransaction](#)[TOraQueue](#)[TOraQueueTable](#)[TOraQueueAdmin](#)[TOraChangeNotification](#)[TOraTrace](#)[TOraErrorHandler](#)[TOraProvider](#)[TBDESession](#)[TOraMetaData](#)[TCRBatchMove](#)**Direct connectivity
(connection without
Oracle client)****Design-time features,
including component
editors and property
editors****[DataSet Manager***](#)****[Migration Wizard***](#)****Smart Refresh in
TSmartQuery component****OraDeveloper Tools********Trial limitations*****FreePascal*******

+	+	-	+
+	+	+	+
+	+	-	+
+	+	+	+
+	+	-	+
+	-	-	+
-	-	-	+
+	+	-	+

* Trial Edition is a fully working version of ODAC Professional Edition for a trial period of 60 days on most supported IDEs. After the trial period expires you must either register or uninstall ODAC. ODAC Trial Edition for Kylix has an additional nag screen trial limitation. ODAC Trial Edition requires the launched IDE on the target workstation when testing .NET applications and applications written on C++Builder. For more information about trial limitations see the Ordering topic.

** Developer and Professional editions with source code for all Component classes are available. Distributed component source code does not include Direct mode implementation. Migration Wizard, DataSet Manager, and code for other products, including OraDeveloper Tools and OraTools, are not distributed.

*** Not available for C++Builder, Delphi 8, FreePascal or Kylix.

**** OraDeveloper Tools is not compatible with all IDEs. A full list of supported environments can be found in Using OraDeveloper Tools

***** Available only in editions with source code and trial edition.

10 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 Seven, 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.

11 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.txt* file.

Table of Contents

1. [Before installing ODAC](#)
2. [Installed packages](#)
 - [Delphi/C++Builder Win32 project packages](#)
 - [Delphi for .NET project packages](#)
 - [Additional packages for using ODAC managers and wizards](#)
 - [Additional .NET packages for using ODAC managers and wizards](#)
3. [Environment Changes](#)
 - [Delphi](#)
 - [Delphi for .NET](#)
 - [C++Builder](#)
 - [Lazarus](#)
4. [Installation of Additional Components and Add-ins](#)
 - [TOraProvider](#)
 - [OraDeveloper Tools](#)
 - [OraTools](#)
 - [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

The ODAC package libraries are divided into Win32 project files and .NET project files.

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

Delphi/C++Builder Win32 project packages

<i>Name</i>	<i>Description</i>	<i>Location</i>
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

** Not included in Borland Delphi 5 and C++Builder 5. In these IDEs this functionality is distributed among the other packages.*

Delphi for .NET project packages

<i>Name</i>	<i>Description</i>	<i>Location</i>
Devart.Dac.dll	DAC run-time package	Global Assembly Cache
Devart.Dac.Design.dll	DAC design-time package	%ODAC%\Bin
Devart.Dac.AdoNet.dll	Data provider core package	Delphi\Bin
Devart.Odac.dll	ODAC Delphi for .NET run-time package	Global Assembly Cache
Devart.Odac.Design.dll	ODAC design-time package	%ODAC%\Bin
Devart.Vcl.dll	TCRDBGrid component	Global Assembly Cache
Devart.Odac.AdoNet.dll	Data provider for Oracle package	Global Assembly Cache

Additional packages for using ODAC managers and wizards

<i>Name</i>	<i>Description</i>	<i>Location</i>
datasetmanagerXX.bpl	DataSet Manager package	Delphi\Bin
oramigwizdXX.dll	ODAC BDE Migration wizard	%ODAC%\Bin

Additional .NET packages for using ODAC managers and wizards

<i>Name</i>	<i>Description</i>	<i>Location</i>
Devart.Dac.DsManager.dll	DataSet Manager Assembly	Global Assembly Cache
Devart.Odac.MigWizard.dll*	ODAC BDE Migration wizard Assembly	Global Assembly Cache

** Included in Borland Delphi 8 only*

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 | Environment options | Library.

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

Delphi for .NET

- Devart.Dac and Devart.Odac should be included in the Namespace prefixes.
- %ODAC%\Lib should be included in the Library Path accessible from Tools | Options | Library - NET.
- %ODAC%\Bin should be included in the Library Path accessible from Tools | Options | Library - NET.
- %ODAC%\Bin should be included in the Component | Installed .NET components | Assembly Search Path.

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

C++Builder

C++Builder 5, 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\dclodac.lpk file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilt 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.

OraDeveloper Tools

OraDeveloper Tools is a powerful database development and administration tool for Oracle. OraDeveloper tools is available as an add-in for Borland Delphi 2005, 2006, and 2007 or as a standalone application. For more information, visit the [OraDeveloper Tools page online](#).

OraTools

OraTools is a separate add-in. It is installed once and can be used by both Delphi and C++Builder. Support for using OraTools in Kylix is not available. For more information, visit the [OraTools page online](#).

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.

©

1997-2012 Devart. All Rights Reserved.

12 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 Win32 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 Win32 applications with ODAC Trial Edition, you will need to deploy the following BPL files and their dependencies (required IDE BPL files) with your application, even if it is built without run-time packages:

dacXX.bpl	always
dacXX.bpl	always
odacXX.bpl	always

Deploying Win32 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 Win32 application:

dacXX.bpl	always
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

Deploying .NET applications

By default you should deploy the following assemblies with your ODAC .NET application:

dacXX.bpl	always
Devart.Dac.dll	always
Devart.Odac.dll	always
Devart.Dac. AdoNet.dll	If your application uses OraDataAdapter component
Devart.Odac. AdoNet.dll	If your application uses OraDataAdapter component

If you remove the names of these assemblies from the References list of your project, these files will not be required on the target computer.

13 Licensing and Subscriptions

Oracle Data Access Components are licensed, not sold. Please read the end-user license agreement (EULA) carefully before using the product. You can find the EULA in the *License.rtf* file in the ODAC installation folder.

Licensing

There are three types of full licenses for ODAC: Single Licenses, Team Licenses, and Site Licenses.

Single Licenses must be purchased for each developer working on a project that uses ODAC.

Purchasing a **Team License** automatically gives four developers a Single License.

Purchasing a **Site License** automatically gives all developers in a company a Single License.

For evaluation purposes only, you may also use ODAC Trial Edition under a temporary **Evaluation License**, which allows you to test ODAC Trial Edition for a period of 60 days, after which you must either remove all files associated with ODAC or purchase a full license.

Licenses can be purchased for the following editions of ODAC: ODAC Standard Edition, ODAC Professional Edition, and ODAC Professional Edition with Source Code, ODAC Developer Edition, and ODAC Developer Edition with Source Code. An edition comparison chart can be found [here](#).

To purchase a license for ODAC, please visit www.devart.com/odac/ordering.html.

If you have any questions regarding licensing, please contact sales@devart.com.

Editions

Full licenses can be purchased for the following editions of ODAC: ODAC Standard Edition, ODAC Professional Edition, and ODAC Professional Edition with Source Code, ODAC Developer Edition, and ODAC Developer Edition with Source Code.

Users can evaluate ODAC with ODAC Trial Edition under Evaluation License.

A comparison chart can be found [here](#).

Subscriptions

The ODAC Subscription program is an annual maintenance and support service for ODAC users.

Users with a valid ODAC Subscription get the following benefits:

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

If you have any questions regarding licensing or subscriptions not covered with Help, please contact sales@devart.com.

Trial Limitations

ODAC Evaluation License lets you try ODAC Trial Edition for a period of 60 days.

There are no functionality limitations in ODAC Trial Edition during the trial period for most supported IDEs, except the following:

- .NET applications and applications written in C++Builder require the corresponding IDE to be launched on the client workstation if they use ODAC Trial Edition
- If you are deploying a project built with ODAC Trial Edition, you will need to include the ODAC library files in your application deployment package. For more information, consult the [Deployment](#) topic.

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

- The version of Delphi, C++Builder or Kylix you are using.
- Your ODAC Registration number.
- Full ODAC edition name and version number. You can find both of these in the About sheet of TOraConnection Editor or from the Oracle | About menu.
- 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.

15 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:

- o Get a "Setup has detected already installed DAC packages which are incompatible with current version" message during ODAC installation.
- o Get a "Procedure entry point ... not found in ..." message when starting IDE.
- o 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 Oracle menu of your IDE.

4. How can I compile an ODAC with Sources Edition so that ODAC packages do not require BDE packages?

Find and comment the following line in the Odac.inc file:

```
{ $DEFINE BDESESSION
```

In this case ODAC will be compiled without the TBDESession component.

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:

- o Access to new versions of ODAC when they are released
- o Access to all ODAC updates and bug fixes
- o Product support through the ODAC Priority Support program
- o 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@crlab.com.

You will be able to renew your ODAC Subscription by email or on the ODAC website. For more information, please contact sales@crlab.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@crlab.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 enable syntax highlighting in ODAC component editors at design time?

Download and install [OraDeveloper Tools](#). In addition to syntax highlighting, OraDeveloper Tools provides a lot of [additional features](#).

Alternatively, you can download and install the freeware [SynEdit component set](#).

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

You can determine your ODAC version number in several ways:

- o During installation of ODAC, consult the ODAC Installer screen.

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

3. 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.

4. 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.

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

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

If *TOrQuery* 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.

6. 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 *TOrQuery.SQLDelete*:

```
begin
  Null;
end;
```

7. 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	TOracleNumberField
In other cases	TFloatField

The default values of these constants are:

IntegerPrecision = 9, *LargeIntPrecision* = 0, *FloatPrecision* = 15

8. 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.

- o Incomparably higher speed of data access, fetching, and processing
- o Possibility to individually adjust and optimize execution parameters for every query
- o Complete support of PL/SQL
- o Return of cursor from a stored procedure or anonymous PL/SQL block (by parameter)
- o Return of PL/SQL result sets from a stored procedure or anonymous PL/SQL block
- o Support for asynchronous execution of queries
- o Possibility to break the execution of long-duration queries
- o Built-in system for debugging in run-time
- o Convenient and simple-to-use standardized error processing mechanism
- o 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 deprecated 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.

- o 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.
- o 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 (not only those requested in SELECT clause, and smart refresh (in Professional and Developer editions only).
- o 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.

9.**I get an access violation when using the TBDESession component.**

This problem may occur if the BDE driver for ORACLE is not configured properly. Run BDE Administrator, go to the Configuration tab, select Configuration\Drivers\Native\ORACLE in the tree, make sure that the DLL32 driver property is set to SQLORA32.DLL.

16 Using ODAC

16.1 Connecting in Direct Mode

ODAC Developer Edition and ODAC Professional Edition allow you to connect to Oracle in two ways: in Client mode, using Oracle Client software, or in Direct mode, over TCP/IP. The chosen connection mode is regulated by 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 *Client mode*, and it 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 target workstations. It is inconvenient and causes additional installation and administration expenses. Furthermore, there are some situations in which installation of Oracle client is not advisable or may be even impossible; for example, when deploying an application to remote machines which do not have qualified administration. To overcome these problems, ODAC Developer Edition and ODAC Professional Edition include an option to connect to Oracle directly over the network using the TCP/IP protocol. This is referred to as connecting in *Direct mode*. Connecting in Direct mode does not require Oracle client software to be installed on target machines. The only requirement for running an ODAC-based application that uses the Direct mode is that the operating system must support the TCP/IP protocol.

Setting up Direct mode connections

To connect to Oracle server using Direct mode, set up your `ToraSession` object as follows.

- Set your `ToraSession` object's [Options.Direct](#) property to True
- Set your `ToraSession` object's [Server](#) to a string that contains the host address of the database server, port number, and the Oracle System Identifier (SID) or Oracle Service Name in the following format:

If Oracle servers has SID is equal to Service Name then string is the following (the format):

Host:Port:SID

If Oracle Server has SID differ from Service Name then is the following (the format):

Host:Port:sid=SID - for connection with using SID

Host:Port:sn=ServiceName - for connection with using Service Name.

Where

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 an Oracle database instance name.

ServiceName is a system alias to an Oracle database instance (or many instances).

Note that the syntax used to set up the `Server` property is different in Direct mode and in Client mode.

In Client mode, the `Server` property must be set to the TNS name of the Oracle server.

Note that if `sid=` prefix is unavailable or `sn=` isn't defined, then by default connection will be established using SID.

Here is an example that illustrates connecting to Oracle in Direct mode. The Oracle server's IP address is 205.227.44.44, its port number is 1521 (this is the most commonly used port for Oracle), and 'orcl' is the SID (this is the 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;
```

For connection using SID:

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

For connection using Service Name:

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

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

To return to working through OCI, just set `TOrasession.Options.Direct` to `False` and update the `Session` to the TNS name of your server.

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

Note: Direct mode is available in ODAC Developer Edition and ODAC Professional Edition, and can be evaluated with Oracle Trial Edition. Attempting to set the `TOrasession.Options.Direct` property to `True` in ODAC Standard Edition will generate a *"Feature is not supported"* error.

Comparison of Client mode vs. Direct mode

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

Advantages of using Direct mode

- Installation and administration of Oracle client software are not required.
- System requirements are reduced.

Direct mode limitations

- Connect using TCP/IP protocol only
- Some types are not available, like `OBJECT`, `ARRAY`, `REF`, `XML`, `BINARY DOUBLE`, `BINARY FLOAT`.
- The `RowsAffected` property is not returned correctly for queries with `RETURNING` clause
- [TOraloader](#) direct loading is not supported.
- Certain problems may occur when using firewalls.
- Direct mode does not support NLS conversion on the client side.
- [Transparent Application Failover](#) is not supported.
- [statement caching](#) is not available.
- OS authentication feature are not available.
- Change notification feature ([TOrachangefeature](#)) cannot be used.
- We do not guarantee stability of multithreaded applications. It is highly recommended to use separate `TOrasession` component for each thread when using ODAC from different threads.

Please note that we do not guarantee ODAC Direct mode compatibility with all Oracle servers and in every network. We have tested Direct mode for all versions of Oracle servers for Windows and Linux since 7.3 on a local network. Other platforms may cause some incompatibility issues.

Connecting in Direct mode is managed transparently by the `TOrasession` object, and you can easily return to connecting through OCI in Client mode at any time if restrictions above become critical for you.

See Also

- [TOrasession.Server](#)
- [TOrasession.Options](#)
- `M:Devart.Odac.TOradataSet.BreakExec()`
- [TOrasql.BreakExec](#)

16.2 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](#)

16.3 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 foregin 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 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" foregin 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](#)

16.4 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](#)

16.5 TOraLoader Component

There are cases 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](#)

16.6 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.

```
// Application 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;
```

```
// Application 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;
```

16.7 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;
    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 queue name
        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 enqueueing and dequeuing must be started on the queue. Use [StartQueue](#) method of TOraQueueAdmin to start enqueueing and dequeuing on the queue.

16.8 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](#)

16.9 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 execute 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
BEGIN
    UPDATE ClobTable
    SET
        Name = p_Name,
        Desc = p_Desc
    WHERE
        Id = p_Id;
```


END;

the following code can be used:

```
OraStoredProc.Options.TemporaryLobUpdate := True;
OraStoredProc.StoredProcName := 'ClobTableUpdate';
OraStoredProc.Prepare;
OraStoredProc.ParamByName('p_Id').AsInteger := Id;
OraStoredProc.ParamByName('p_Name').AsString := Name;
OraStoredProc.ParamByName('Desc').ParamType := ptInput;
OraStoredProc.ParamByName('Desc').AsOraClob.LoadFromFile(FileName);
OraStoredProc.Execute;
```

Note that LOB parameter can have ptInputOutput type only when [TemporaryLobUpdate](#) option is set to True. 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 `P:Devart.Dac.TDADatasetOptions.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, Delphi 8 (for .NET) and Delphi 7. To use BLOB compression under Delphi 6, Delphi 5 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; sourceLen: longint): longint;
```

var

```
CompressProc: TCompressProc;
UncompressProc: TUncompressProc;
```

You can compress and decompress a single LOB. To do it set the `P:Devart.Dac.TCompressedBlob.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

- Oracle8 Application Developer's Guide - Large Objects (LOBs)
- DBMS LOB package
- [TOraLob](#)
- [TDAParam.ParamType](#)
- [TCustomDADataset.Options](#)
- [TOraDataSet.Options](#)
- BlobPic demo

- Clob demo
-

© 1997-2012 Devart. All Rights Reserved.

16.10 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 VARCHAR 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].AsString := 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](#)
-

16.11 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 `TOrasObject`. Use `TOrasDataSet.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 ODAAC 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 `TOrasQuery` or `TOrasSQL` component use `TOrasParam.AsObject` property to set attributes' value of `PERSON` parameter. But before you should assign `dtObject` to `TOrasParam.DataType` property and allocate object handle by `TOrasParam.AllocObject` method.

```
var
  OraSQL: TOrasSQL;
. . .
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
-

16.12 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 data stored 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> </root>';
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> </root>';
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_lob>'));
    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. Then 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. [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](#)

16.13 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 performance 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

16.14 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 [OraTools](#) for this purpose) and see four new records appended.

```
OraSQL.Execute(4);
```


16.15 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;
```

© 1997-2012 Devart. All Rights Reserved.

16.16 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), Length(e.Message));
    on e: Exception do
        OCIExtProcRaiseExcpWithMsg(Context, 20000, PChar(e.Message), Length(e.Message));
    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, file_text OCI

```

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

16.17 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](#)

16.18 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](#)

16.19 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 [Disconnect](#) 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 ImLockImmediate, 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.Disconnect](#)
 - [Connected](#)
 - [Working in unstable network](#)
-

16.20 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      := pgNumeric;
  MinPrecision := 0;
  MaxPrecision := 4;
  MinScale    := 0;
  MaxScale    := 0;
  FieldType   := ftSmallint;
  OraSession.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecision, MinScale, MaxScale,
  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, ftSmallint);
// rule for numeric(10,0)
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 5, 10, 0, 0, ftInteger);
// rule for numeric(15,0)
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 11, rlAny, 0, 0, ftLargeint);
// rule for numeric(5,2)
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, 9, 1, rlAny, ftFloat);
// 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, ftFMTBcd);
```

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, ftFloat);
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny, 0, 4, ftBCD);
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny, 0, rlAny, ftFMTBCD);
```

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, ftFMTBCD);
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny, 0, 4, ftBCD);
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, 9, rlAny, rlAny, ftFloat);
```

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 specified 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 dtm 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.

16.21 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 VARCHAR(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 VARBINARY 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.AddFieldNamedRule ('ENAME', ftString);
OraQuery.DataTypeMap.AddFieldNamedRule ('HIREDATE', ftDateTime);
OraQuery.DataTypeMap.AddFieldNamedRule ('SAL', ftFloat);
OraQuery.Open;
```

16.22 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 `TDAUpdateSQL`. 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 `P:Devart.Dac.TDADatasetOptions.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 [TDevart.Odac.TOraLoader](#) component instead of Insert/Post methods, or execution of the INSERT commands multiple times in a cycle. Sometimes usage of [TDevart.Odac.TOraLoader](#) improves performance several times.

16.23 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, ODA 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](#)
-

16.24 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](#)

16.25 Using Several DAC Products in One IDE

UniDAC, ODAC, SDAC, MyDAC, IBDAC, PgDAC, and LiteDAC components use common base packages (for Win32) and assemblies (for .NET) listed below:

Packages:

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

Assemblies:

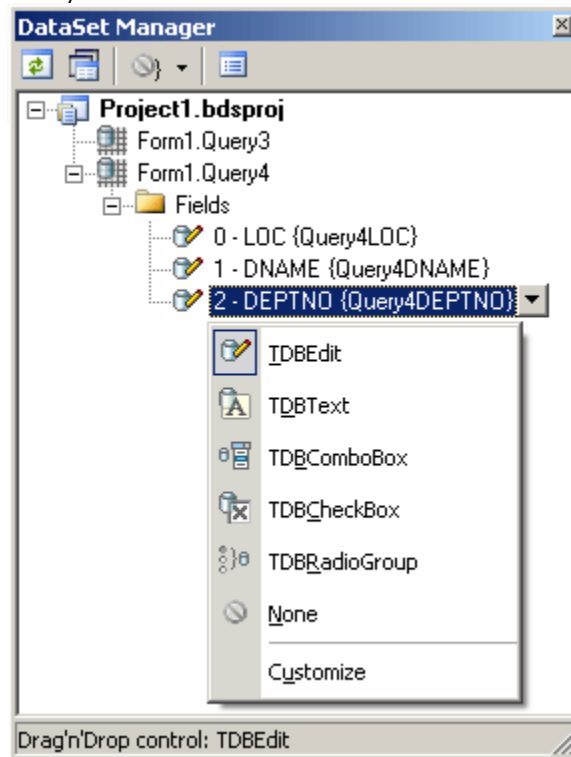
- Devart.Dac.dll
- Devart.Vcl.dll
- Devart.Dac.Design.dll
- Devart.Dac.AdoNet.dll

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.

16.26 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

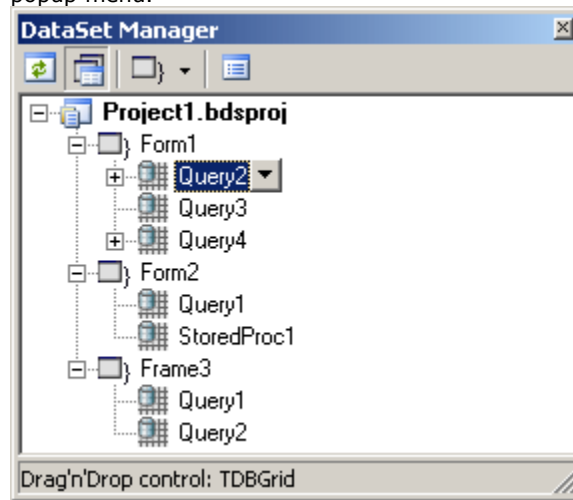
You can display the DataSet Manager window by clicking DataSet Manager on the Tools menu. You can also use IDE desktop saving/loading to save DataSet Manager window position and restore it during the next IDE loads.

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 Borland Delphi 2005 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

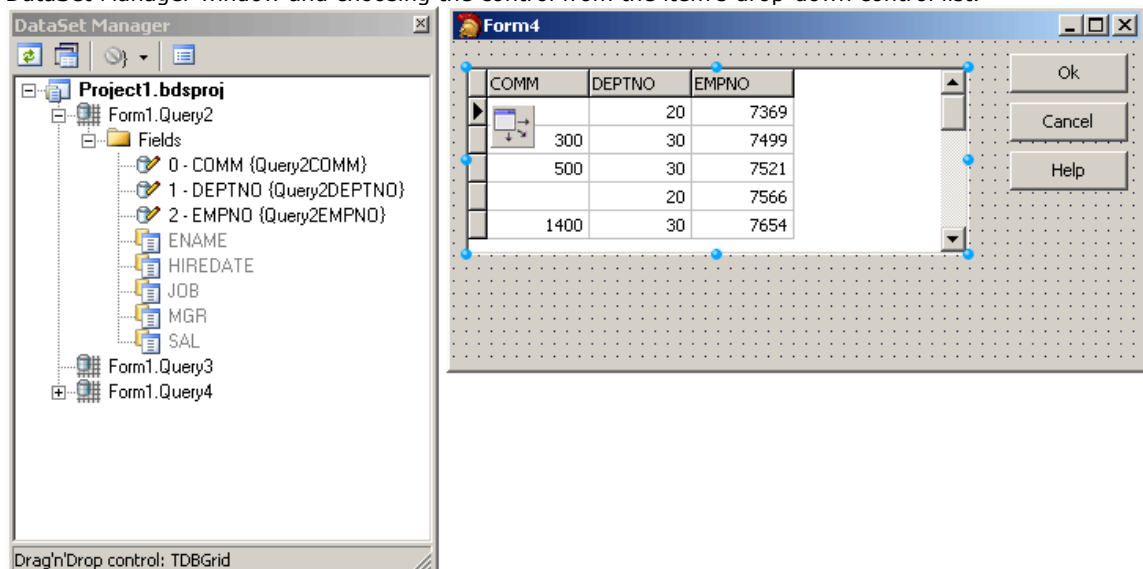
DataSet Manager window popup menu.



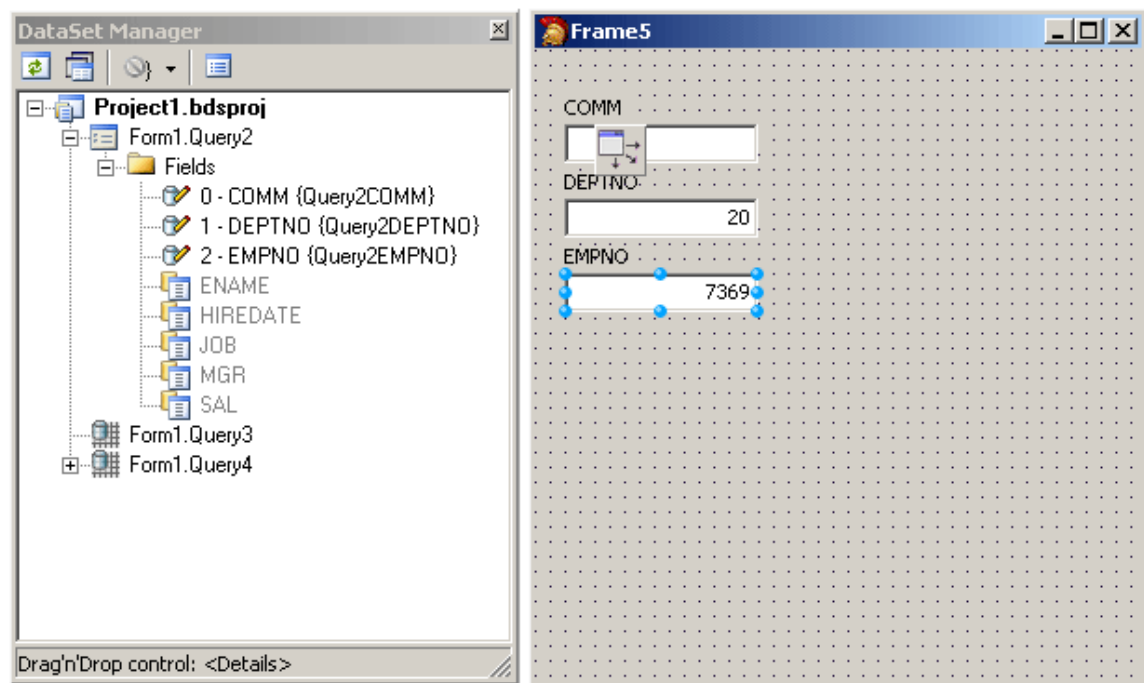
Creating Data-bound Controls

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.

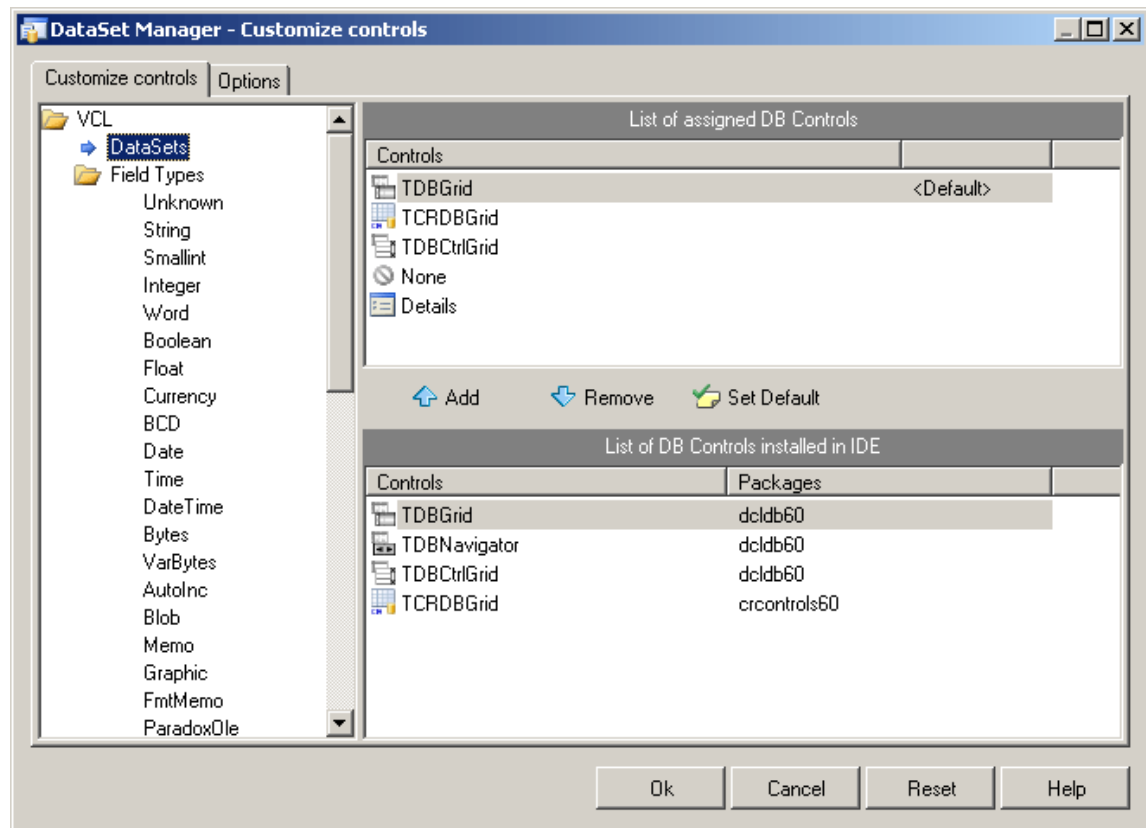


To use Details layout choose Details from the DataSet node drop-down control list in the DataSet Manager window. Then select required controls in the drop-down control list for each DataSet field. DataSet fields must be created. After setting required options you can drag the DataSet to the form from 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 - Customi e 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

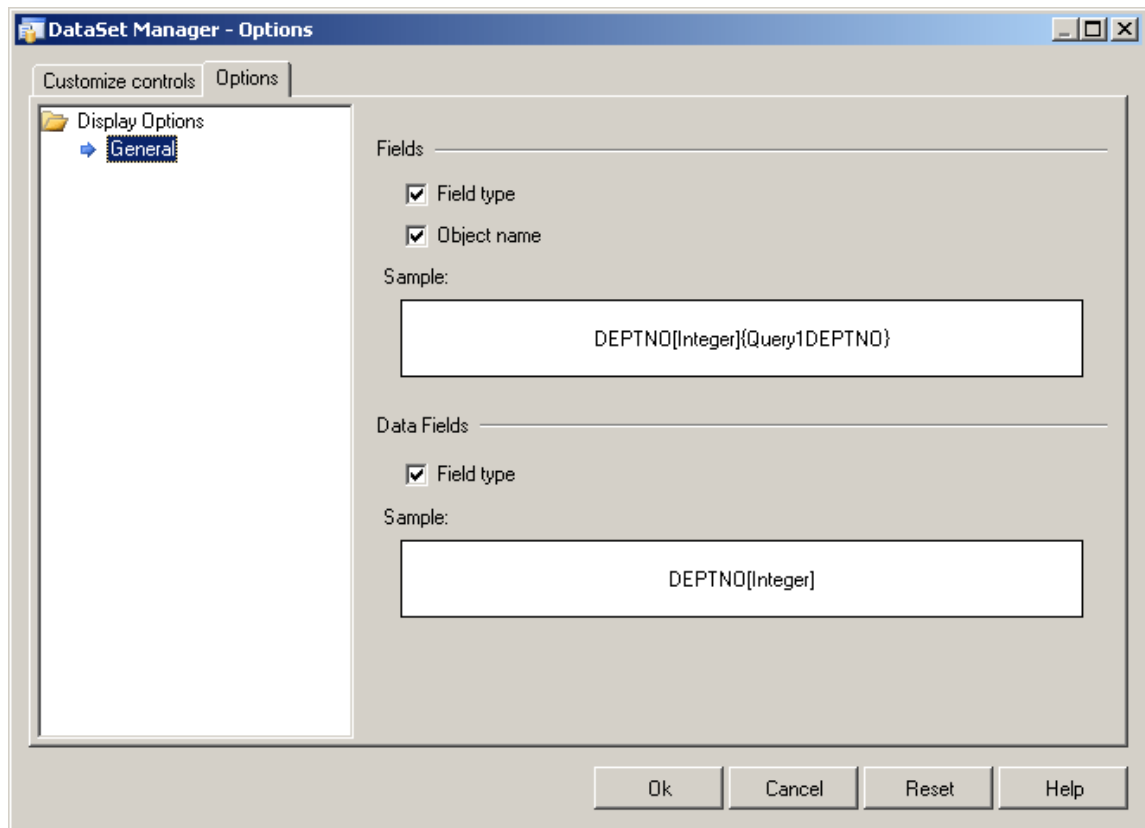
DataSet Manager allows you to create and remove TField objects. DataSet must be active to work with its fields in the DataSet Manager. You can add fields, based on the database table columns, create new fields, remove fields, use drag-n-drop to change fields order.

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 choose 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.

16.27 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.
-

16.28 Migration Wizard

NOTE:

Migration Wizard is available only for Delphi IDE and is not available for C++Builder. BDE Migration Wizard allows you to convert your BDE projects to ODAC. This wizard replaces BDE components at the specified project (dfm-and pas-files) to ODAC.

To convert a project, perform the following steps.

- Select **BDE Migration Wizard** from **Oracle** menu
- Select **Replace BDE components** to replace corresponding components with ODAC and press the Next button.
- 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 **BDE Migration Wizard** from **Oracle** 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 **es** in the dialog that appeared.

Your project will be restored to its previous state.

16.29 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 (OdacClx under Linux) unit in your application. This feature is needed for writing console applications.

D I h and C++B Id

By default ODAC does not require Forms, Controls and other GUI related units. Only TConnectDialog, ToraErrorHandler and ToraAlerter components require the Forms unit.

Kyl x

By default ODAC does not require QT library. Only ToraErrorHandler and ToraAlerter components include QT-dependent code. In addition, SmartRefresh feature is disabled under Kylix. But having professional version this feature can be enabled by defining SMART_REFRESH macro for Linux in Odac.inc file. By default SMART_REFRESH macro is defined for Windows only. But be aware that SmartRefresh function requires QT.

16.30 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 DataSetChangeEvent 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.

16.31 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 Oracle 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 Oracle 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 5 - 7, 2005, 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.

16.32 Integration with Developer Tools

16.32.1 dbForge Studio for Oracle

Starting with version 8.2.7, ODAC supports working with dbForge Studio for Oracle. [dbForge Studio for Oracle](#) (formerly known as [OraDeveloper Tools](#)) is a powerful Oracle database development environment.

This tutorial explains the integration between ODAC and dbForge Studio for Oracle.

Compatibility

ODAC supports dbForge Studio for Oracle in almost all Delphi and C++Builder versions except the following: Delphi 2006, Delphi 2006 .NET, C++Builder 2006, Delphi 2005, Delphi 5, and C++Builder 5.

Choosing integration

It is possible to choose what tool will be used by ODAC. Moreover, it is possible to disable integration with any tool whatsoever. It can be done in the Delphi main menu->ODAC->Database Tools menu. There you can choose one of the three options (two options for IDE versions earlier than RAD Studio 2007):

Option	Meaning
dbForge Studio for Oracle Integration	All the functionality described below uses dbForge Studio for Oracle
OraDeveloper Tools Integration	All functionality described below uses OraDeveloper Tools. OraDeveloper Tools does not support IDE versions earlier than RAD Studio 2007, therefore there is no such option in these IDEs.
No Integration	All functionality described below is disabled and cannot be used

Using dbForge Studio for Oracle in the components

When dbForge Studio for Oracle integration is enabled, several ODAC components obtain additional functionality.

Note: Using any submenus of the "dbForge Studio for Oracle" popup menu described below opens dbForge Studio for Oracle (if it was not opened before).

TOraSession

The TOraSession component obtains an additional popup menu (available by right-click on the component) named "dbForge Studio for Oracle". This menu contains the "Find in Database Explorer" submenu. Clicking this submenu will cause the following steps to be performed by ODAC and dbForge Studio for Oracle:

- If TOraSession.ConnectionString is blank, the "Connection is not defined" error is generated.
- If TOraSession.ConnectionString is not blank, ODAC sends the message to dbForge Studio for Oracle to find in its Database Explorer the connection with the same connection parameters as TOraSession has. Note that if dbForge Studio for Oracle was not running until this moment, it starts running.
- If dbForge Studio for Oracle found the appropriate connection in its Database Explorer, it will make this connection the current connection and open it. Otherwise, dbForge Studio for Oracle will show the window with the following content:

There is no suitable connection found in Database Explorer. Do you want to create one?

It is up to you to decide whether you want or not to create a new connection in Database Explorer of dbForge Studio for Oracle.

Also, the TOraSession editor form obtains a dropdown list that is filled with connections from Database Explorer of dbForge Studio for Oracle. When the value from this list is chosen, TOraSession's options are filled with the options obtained from dbForge Studio for Oracle.

Note: The dropdown list on the TOraSession editor form is filled only if dbForge Studio for Oracle was running before the editor was opened.

TOraQuery and TSmartQuery

The TOraQuery and TSmartQuery components obtain an additional popup menu (available by right-click on the component) named "dbForge Studio for Oracle". This menu contains the following submenus:

- "Edit SQL" - clicking this submenu opens the new SQL editor in dbForge Studio for Oracle. Saving changes made in this editor saves them to the linked TOraQuery or TSmartQuery component.
- "Query Builder" - clicking this submenu opens the new Query Builder editor in dbForge Studio for Oracle.
- "Step Into" - clicking this submenu starts debugging of a query in dbForge Studio for Oracle.
- "Retrieve data" - clicking this submenu opens a query in dbForge Studio for Oracle. If the query contains the SELECT statement, the data is retrieved. Otherwise, the query is executed.

Also, opening the TOraQuery and TSmartQuery editor form opens dbForge Studio for Oracle (if it was not opened before). This allows using powerful SQL editor of dbForge Studio for Oracle directly in the IDE.

TOraTable

The TOraTable component obtains an additional popup menu (available by right-click on the component) named "dbForge Studio for Oracle". This menu contains the following submenus:

- "Find in Database Explorer" - clicking this submenu looks for the table specified in TOraTable. TableName in Database Explorer of dbForge Studio for Oracle. If the table is found, it is made current.
- "Edit Object" - clicking this submenu opens the table editor in dbForge Studio for Oracle. In the editor, it is possible to modify the structure of the table, its indexes, etc.

TOraStoredProc

The TOraStoredProc component obtains an additional popup menu (available by right-click on the component) named "dbForge Studio for Oracle". This menu contains the following submenus:

- "Find in Database Explorer" - clicking this submenu looks for the stored procedure specified in TOraStoredProc.StoredProcName in Database Explorer of dbForge Studio for Oracle. If the stored procedure is found, it is made the current.
- "Step Into" - clicking this submenu starts debugging of a stored procedure in dbForge Studio for Oracle.
- "Recompile" - clicking this submenu recompiles a stored procedure in dbForge Studio for Oracle.
- "Recompile with debug info" - clicking this submenu recompiles a stored procedure with debug info in dbForge Studio for Oracle.

Also, opening the TOraStoredProc editor form opens dbForge Studio for Oracle (if it was not opened before). This allows using powerful SQL editor of dbForge Studio for Oracle directly in the IDE.

TOraSQL

The TOraSQL component obtains an additional popup menu (available by right-click on the component) named "dbForge Studio for Oracle". This menu contains the following submenus:

- "Edit SQL" - clicking this submenu opens the new SQL editor in dbForge Studio for Oracle. Saving changes made in this editor saves them to the linked TOraSQL component.
- "Step Into" - clicking this submenu starts debugging of a query in dbForge Studio for Oracle.

Also, opening the TOraSQL editor form opens dbForge Studio for Oracle (if it was not opened before). This allows using powerful SQL editor of dbForge Studio for Oracle directly in the IDE.

TOraScript

The TOraScript component obtains an additional popup menu (available by right-click on the component) named "dbForge Studio for Oracle". This menu contains the following submenus:

- "Edit SQL" - clicking this submenu opens the new SQL editor in dbForge Studio for Oracle. Saving changes made in this editor saves them to the linked TOraScript component.
- "Step Into" - clicking this submenu starts debugging of a script in dbForge Studio for Oracle.

Also, opening the TOraScript editor form opens dbForge Studio for Oracle (if it was not opened before). This allows using powerful SQL editor of dbForge Studio for Oracle directly in the IDE.

TOraPackage

The TOraPackage component obtains an additional popup menu (available by right-click on the component) named "dbForge Studio for Oracle". This menu contains the following submenus:

- "Find in Database Explorer" - clicking this submenu looks for the package specified in TOraPackage. PackageName in Database Explorer of dbForge Studio for Oracle. If the package is found, it is made the current.
- "Edit Object" - clicking this submenu opens the package editor in dbForge Studio for Oracle. In the editor, it is possible to modify the content of the package.

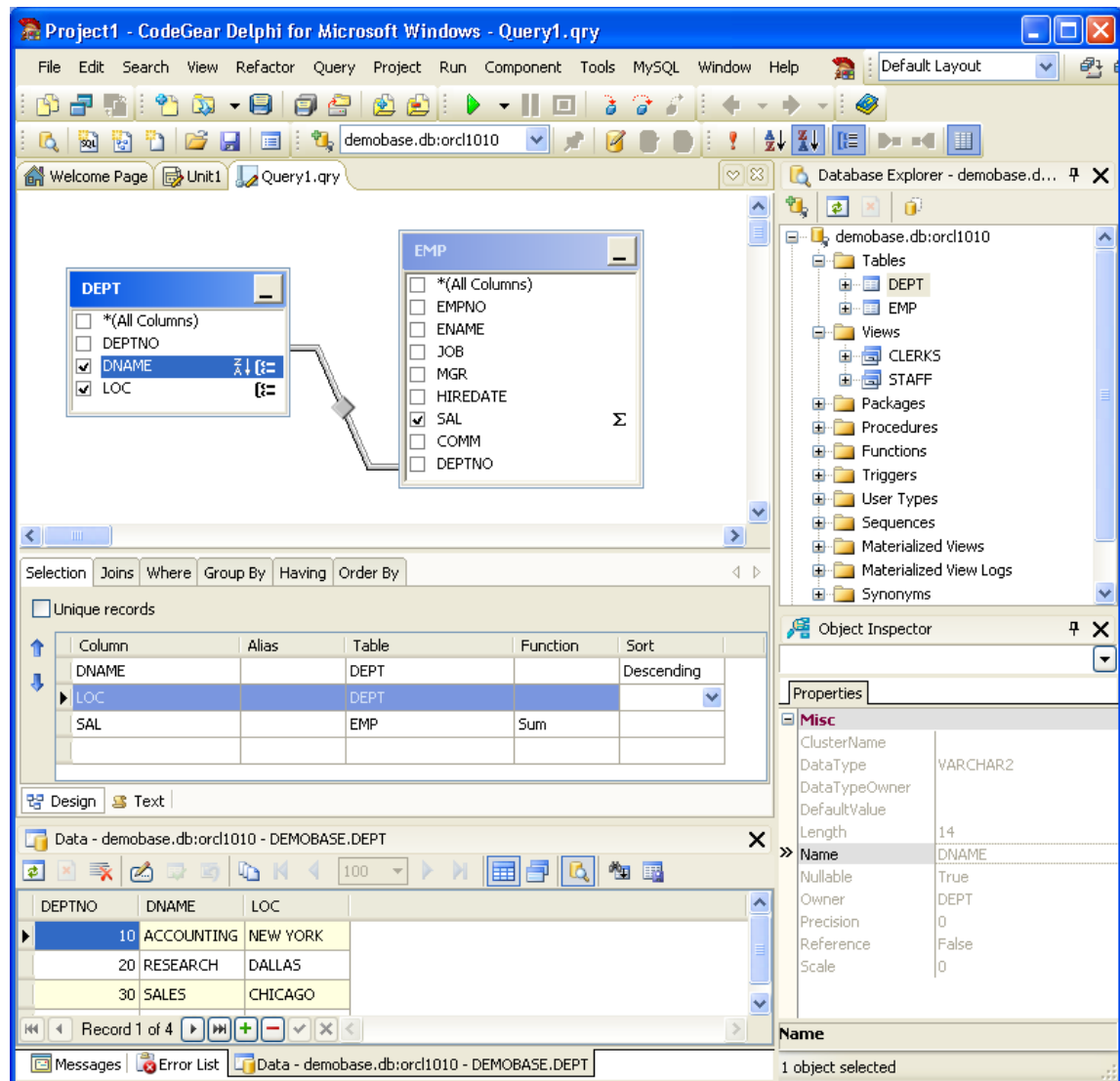
16.32.2 OraDeveloper Tools

This article provides basic information about OraDeveloper Tools. The article explains what OraDeveloper Tools is, where to download it, how to install it, and how to start using it. For more in-depth information about OraDeveloper Tools please refer to the product documentation.

Introduction

OraDeveloper Tools is a powerful IDE add-in designed to automate and simplify the Oracle database development process. It integrates into Visual Studio and Delphi, making all database development and administration tasks available from your favorite IDE. Using OraDeveloper Tools, you can:

- Create, modify and delete database connections and easily navigate server-specific database schema information in tree view
- Create, modify, and drop various database objects
- View and edit table data with an intelligent grid-based editor
- Edit SQL and PL/SQL code in a comfortable scripting environment with context-sensitive code completion, syntax highlighting, outlining, code navigation and code templates
- Debug SQL scripts and PL/SQL programs
- Open and save SQL documents
- Create and execute SQL statements
- Examine the SQL query execution plan
- Visually design queries using Query Builder
- Create and deploy Oracle database projects
- Administrate Oracle security, sessions, and events
- Easily export data, database objects, schemas and databases
- Create new components by dragging items from Database Explorer
- Take advantage of the extended integration functionality in ODAC component designers



Versions and Compatibility

OraDeveloper Tools is available in two editions.

- [OraDeveloper Tools for Visual Studio](#), which includes support for Visual Studio .NET 2003 and Visual Studio 2005
- [OraDeveloper Tools for Delphi](#), which includes support for Delphi 2005, BDS 2006, and Delphi 2007.

ODAC 6.00 and higher is compatible with OraDeveloper Tools for Delphi 2.00 and higher.

Related Products

Devart also offers a number of other database products, including OraDeveloper Studio, the standalone version of this Oracle development tool, and dbForge Fusion and dbForge Studio for MySQL, a parallel product line for MySQL.

You can find a full description of all the Devart database tools on the [Devart web site](#).

Downloading and Installing

OraDeveloper Tools comes in separate installation packages for each supported IDE. If you have purchased ODAC Developer Edition, you are entitled to receive one free license for the full version of OraDeveloper Tools. Please consult your order confirmation email for instructions on how to download the installation package for the IDE you are using. Otherwise, you can purchase OraDeveloper Tools on the [Devart website](#) or download a free trial copy of the version you need from the [OraDeveloper Tools download page](#).

Before installing OraDeveloper Tools, make sure that an older version of the software is not installed for the target IDE. Close all IDE instances, launch the downloaded installer, and follow the instructions of the wizard to install the product. Now upon launching the IDE, the OraDeveloper Tools logo should appear on the splash screen and a new OraDeveloper Tools toolbar should be added to the IDE interface.

Basic Usage Instructions

Working with database connections

To start using OraDeveloper Tools, you will need to establish a connection to the database you want to work with first. After a connection is established, you can open it to retrieve and manipulate the data provided.

In OraDeveloper Tools database connections are managed in a separate Database Explorer window. The Database Explorer window displays all available database connections at the top level of its tree hierarchy.

To add a database connection in the Database Explorer, complete the following steps.

1. On the Database Explorer window toolbar, press the New Connection button or select the appropriate item from the popup menu.
2. On the "Data Source" tab of the Database Connection Properties dialog box, choose a database server from the list.
3. On the "Connection" tab of the Database Connection Properties dialog box, provide the main logon information required to connect to the server.
4. On the "Parameter" tab of the Database Connection Properties dialog box, provide all the specific connection properties you need.
5. Test the connection you have created by clicking the "Test Connection" button.
6. Click OK to establish the database connection.

The Database Connection Properties dialog box will close, and a newly created database connection will appear at the top level of the tree, allowing you to access your Oracle database.

You can modify an existing database connection by right-clicking on its node in Database Explorer, and choosing "Modify Connection" from the node popup menu. In the Database Connection Properties dialog box that will appear, make any necessary changes to the connection properties. After you apply these changes by pressing OK, the database connection will close and reopen with the new parameters.

You can rename database connection using the in-place item editor of the Database Explorer tree view.

You can drop a database connection by choosing "Delete" from its node popup menu.

Displaying server-specific database schema information in tree view

After a database connection is created and opened, you can explore its database schema by navigating its hierarchy tree. Database Explorer allows you to view, edit, create and drop database objects for all connections. To modify or add an item to the database schema, right click on its node to display a popup menu with the available actions for this node.

If other users are modifying this database simultaneously, you can update the list of database objects displayed in the Database Explorer and their properties to reflect the latest changes by pressing the "Refresh" button.

Working with database objects

You can create database objects by using the Database Explorer popup menu or by pressing the "Create New Database Object" button on the OraDeveloper Tools toolbar.

To modify an object displayed in the Database Explorer tree, double click on its node to invoke its object editor. In OraDeveloper Tools, objects are represented as tabbed documents that appear in the main IDE editor space. Object editor documents have several interrelated views, and let you manually apply or cancel the changes you make.

An object's properties can also be viewed quickly in a separate Properties window by navigating to that object in the Database Explorer.

To drop a database object, select it and choose "Delete" from popup menu.

Working with database projects

You can use database projects to manage SQL scripts, query files, and database objects easily.

Database projects let you to organize related scripts and queries and provide fast access to selected database objects. They can be created, compiled, and deployed. Some of the advanced benefits of using database projects include the possibility to compile a collection of source objects, create a whole database from several scripts, and specify project deployment order automatically. Projects are an added feature of OraDeveloper Tools, and project folder and file structure, connection and database object links, deployment order are stored locally in a file with .oradev extension.

To create a new project select Tools | Devart Developer Tools | Oracle | New Blank Project.

To open an existing project, select Tools | Devart Developer Tools | Open Project ...

Each project can be associated with one connection. Project deployment and compilation are performed

through this connection. To associate a connection with the project, right click on the connection in Database Explorer and select "Assign to project" from the popup menu.

To deploy a project perform the following steps.

1. Select Tools | Devart Developer Tools | Project | Deployment Order.
2. Specify the files which are to be executed by setting the proper check box. Use the "Select All" and "Deselect All" buttons, if necessary.
3. Define the order of the files in the list using the "Move Up" and "Move Down" buttons or dragging the required files. The scripts will be executed in this order.
4. Press "Okay" to apply changes and exit or "Cancel" to exit without applying the changes you have made.
5. Select Tools | Devart Developer Tools | Project | Deploy.

To compile all the database source objects in a project, select Tools | Devart Developer Tools | Project | Compile or Tools | Devart Developer Tools | Project | Compile With Debug Info.

Creating and executing SQL statements and scripts with SQL editor

To execute an SQL statement or script, first open a new SQL document by clicking on the "Create New SQL Editor" button on the toolbar. Type your query or script in it, and click the "Execute SQL" button. Query results and any error messages will be redirected to the common Output window. You can view the datasets returned from SELECT queries in the Data tab (select Data from the View menu if this tab is not yet visible). Note that OraDeveloper Tools allows obtaining multiple result sets from SQL scripts.

Visually designing queries with the Query Builder

In an SQL document, you can switch to Design view to construct a query using Query Builder. In this mode you can create SELECT statements visually without using SQL. The Query Builder view is synchronized with the text view, and if you had a correct SELECT statement in the SQL editor, it is automatically inserted into the Query Builder. In the Query Builder you can drag and drop tables from the Database Explorer, use a special tabbed editor to setup JOIN statements easily, as well as WHERE, GROUP BY, HAVING and ORDER BY clauses.

Opening and saving SQL documents

You can save your SQL document at any time for future use. SQL editor documents are saved with extension ".sql". Query Builder documents have extension ".qry". When opened, Query Builder table controls restore their original position on the data diagram.

Examining the SQL query execution plan

One of the most important factors to worry about when developing SQL queries is query performance. With OraDeveloper Tools you can evaluate and optimize the performance of a critical query easily by inspecting it visually in Plan view. Just paste your query into an SQL document and switch to Plan view to see even the most complicated statements parsed by Oracle and presented in a tree with explanations of what every step in the plan does.

Viewing and editing data using grid based editor

The data of table and view objects can be edited in a grid-based data editor. This data editor is accessible from object's popup menu or from the Data view. When you open the editor, it is automatically filled with the data contained in the object. Here you can edit data directly in a grid format.

To insert a new row, press the Ins key. To delete a row, select it and press Del. Changes are stored until you commit them; to apply changes made, press Enter, and to cancel all pending changes press Escape. To refresh data from table choose "Refresh" from the popup menu.

Creating new components by dragging items from Database Explorer

You automatically create new components that reference existing resources by selecting a connection, table, view, stored procedure or package object in the Database Explorer and dragging it onto a form designer. Then the IDE will create a new component that references the selected resource automatically.

Note Drag-n-drop support is not available for Delphi 2005.

Extended Integration Features with ODAC Component Editors

OraDeveloper Tools integrates with ODAC to give you a number of extended design-time benefits.

- Drag-n-drop support for creating new components from some database objects
- Easy selection of existing connections in TOraSession component
- Added "Find", "Debug", "Edit SQL", "Query Builder", and "Retrieve Data" verbs to component popup menus
- Standard SQL editor in all ODAC components with SQL field properties is replaced with the full-featured OraDeveloper Tools SQL editor, completed with code completion, syntax highlighting, outlining, and other functionality.

Complete Documentation

OraDeveloper Tools comes with comprehensive documentation that describes all aspects of the software usage and contains a number of walkthroughs and reference topics.

There are several ways to open this documentation:

- Use the appropriate shortcut in Start menu, for instance, Start | Programs | Devart OraDeveloper Tools for Delphi 2006 | Documentation.
- Use command from the IDE menu: Tools | Devart Database Developer Tools | Oracle | Help.
- Focus on any OraDeveloper Tools window (for example, on the Database Explorer), and press F1.

16.32.3 OraTools Add-In

To extend ODAC design-time capabilities there is an OraTools Add-in provided. It is an easy-to-use and versatile ODAC design-time extension to manipulate data and database objects of Oracle RDBMS. With OraTools Add-in you can build, execute, verify and optimize your SQL and PL/SQL statements, navigate and view properties of schema objects and debug stored procedures.

OraTools Add-in consists of four parts:

- OraDesigner
- OraExplorer
- OraBuilder
- OraDebugger

They are embedded in IDE and can be called from its main menu, component editors and component popup menus.

Sometimes when you install or upgrade OraTools Add-in or upgrade ODAC there is an error message during ODAC design-time packages initialization. It says: 'Current version of OraTools Add-in is incompatible with ODAC X.XX'. To solve this problem go to the OraTools Add-in directory and view Requirements section in ReadMe.txt. There you will find the lowest ODAC version compatible with current add-in version. Now if your current ODAC version number is lower than required by add-in you should upgrade ODAC and if current version is higher then upgrade OraTools Add-in. In rarer cases you may need to upgrade both products.

16.33 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.

For more information about RAD Studio XE2, please refer to [World Tour](#).

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:

T PE	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.

T PE	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:

T PE	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:

T PE	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:

T PE	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/GetWindowLong 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 installation 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. Your 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/en/Installing_and_Running_the_Platform_Assistant_on_the_Target_Platform

16.34 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.

17 Reference

This page shortly describes units that exist in ODAC.

Units

Unit Name	Description
BDESession	This unit contains implementation of the TBDESession component.
CRAccess	This unit contains base classes for accessing databases.
CRBatchMove	This unit contains implementation of the TCRBatchMove component.
CRDataTypeMap	This unit contains base classes for Data Type Mapping
CREncryption	This unit contains base classes for data encryption.
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.
Devart.Dac.DataAdapter	This unit contains implementation of the DADDataAdapter class.
Devart.Odac.DataAdapter	This unit contains implementation of the OraDataAdapter class.
MemData	This unit contains classes for storing data in memory.
MemDS	This unit contains implementation of the TMemDataSet class.
MemUtils	This unit contains auxiliary procedures and functions used in the DAC code.
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.

[OraErrHand](#)

This unit contains the TOraErrorHandler component.

[OraError](#)

Description is not available at the moment.

[OraLoader](#)

This unit contains implementation of the TOraLoader component.

[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.

[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.

[VirtualTable](#)

This unit contains implementation of the TVirtualTable component.

17.1 BDESession

This unit contains implementation of the TBDESession component.

Classes

Name	Description
<u>TBDESession</u>	Allows integrating ODAC components into existing BDE-based applications without making separate connections.

© 1997-2012 Devart. All Rights Reserved.

17.1.1 Classes

Classes in the **BDESession** unit.

Classes

Name	Description
TBDESession	Allows integrating ODAC components into existing BDE-based applications without making separate connections.

© 1997-2012 Devart. All Rights Reserved.

17.1.1.1 BDESession.TBDESession Class

Allows integrating ODAC components into existing BDE-based applications without making separate connections.

For a list of all members of this type, see [TBDESession](#) members.

Unit

[BDESession](#)

Syntax

```
TBDESession = class(TOraSession);
```

Remarks

TBDESession allows you to integrate ODAC components into existing BDE-based applications without making separate connections. TBDESession derives its functionality from its ancestor TOraSession. Assign your TDatabase component name to the DatabaseName property, then call the Connect method to establish a TBDESession connection.

Note: TBDESession does not work with BDE version 5.11 or higher in the OCI8 call style. You may still use it with the OCI7 call style. To use it in such way set the BDE driver DLL32 option to 'SQLORA32.DLL'.

Inheritance Hierarchy

```
TObject
  TCustomDAConnection
    TOraSession
      TBDESession
```

See Also

- [TOraSession](#)

© 1997-2012 Devart. All Rights Reserved.

[TBDESession](#) class overview.

Properties

Name	Description
AutoCommit (inherited from TOraSession)	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 (inherited from TOraSession)	Used to indicate if the database connection is active.
ConnectMode (inherited from TOraSession)	Used to specify the system privileges to use when a user connects to the server.

ConnectPrompt (inherited from TOraSession)	Used to supply a prompt for a name and password.
ConnectionString (inherited from TOraSession)	Used to assign the TOraSession.Username , TOraSession.Password and TOraSession.Server properties at once.
ConvertEOL (inherited from TCustomDAConnection)	Allows customizing line breaks in string fields and parameters.
Database	Used to access properties, events, and methods of the database component associated with this TBDESession.
DatabaseName	Used to specify the name of the database to associate with this TBDESession component.
Debug (inherited from TOraSession)	Used to display SQL statements being executed with their parameter values and data types.
Home (inherited from TOraSession)	Not supported.
HomeName (inherited from TOraSession)	Used to select the Oracle client to use with the application.
InternalName (inherited from TOraSession)	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 (inherited from TOraSession)	Used to get an error code which resulted from previous call to the OCI interface function.
LDA (inherited from TOraSession)	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 (inherited from TOraSession)	Indicates the set of OCI routines used.
OCISvcCtx (inherited from TOraSession)	Used to return Oracle 8 service context handle of the current connection.
Options (inherited from TOraSession)	Used to specify the behaviour of a TOraSession object.
OracleVersion (inherited from TOraSession)	Used to get Oracle server version number as string.
Password (inherited from TOraSession)	Used to specify a password for a connection.
Pooling (inherited from TCustomDAConnection)	Enables or disables using connection pool.
PoolingOptions (inherited from TOraSession)	Used to specify the behaviour of connection pool.
ProxySession (inherited from TOraSession)	Used to enable multiple user sessions within a single database session.
Schema (inherited from TOraSession)	Used to change the current schema of the session to the specified schema.
Server (inherited from TOraSession)	Contains the server name.

[SessionName](#)

Use the SessionName property to specify the session with which TBDESession component is associated.

[SQL](#) (inherited from [TOraSession](#))

Uses embedded TOraSQL object to execute any SQL statement.

[ThreadSafety](#) (inherited from [TOraSession](#))

Used to allow the usage of the OCI in multi-threaded environment.

[Username](#) (inherited from [TOraSession](#))

Contains username.

Methods

Name	Description
ApplyUpdates (inherited from TCustomDAConnection)	Overloaded. Applies changes in datasets.
AssignConnect (inherited from TOraSession)	Shares database connection between the TOraSession components.
ChangePassword (inherited from TOraSession)	Changes the current user password for the session by a new one.
Commit (inherited from TCustomDAConnection)	Commits current transaction.
Connect (inherited from TCustomDAConnection)	Establishes a connection to the server.
CreateDataSet (inherited from TCustomDAConnection)	Creates a dataset component.
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.
GetSequenceNames (inherited from TOraSession)	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 (inherited from TOraSession)	Provides access to OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc.
Ping (inherited from TOraSession)	Checks whether the connection to the server can be established.
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](#) (inherited from [TOraSession](#))

[Savepoint](#) (inherited from [TOraSession](#))

[StartTransaction](#) (inherited from [TOraSession](#))

Cancels all updates for the current transaction.

Defines a point in the transaction to which you can roll back later.

Overloaded. Begins a new user transaction against the database server.

Events

Name	Description
OnConnectChange (inherited from TOraSession)	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 (inherited from TOraSession)	Occurs when Transparent Application Failover (TAF) seamlessly attempts to failover to another Oracle instance.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TBDESession** class.

For a complete list of the **TBDESession** class members, see the [TBDESession Members](#) topic.

Public

Name	Description
ApplyUpdates (inherited from TCustomDAConnection)	Overloaded. Applies changes in datasets.
AssignConnect (inherited from TOraSession)	Shares database connection between the TOraSession components.
ChangePassword (inherited from TOraSession)	Changes the current user password for the session by a new one.
Commit (inherited from TCustomDAConnection)	Commits current transaction.
Connect (inherited from TCustomDAConnection)	Establishes a connection to the server.
ConnectDialog (inherited from TCustomDAConnection)	Allows to link a TCustomConnectDialog component.
ConvertEOL (inherited from TCustomDAConnection)	Allows customizing line breaks in string fields and parameters.
CreateDataSet (inherited from TCustomDAConnection)	Creates a dataset component.
CreateSQL (inherited from TCustomDAConnection)	Creates a component for queries execution.
Database	Used to access properties, events, and methods of the database component associated with this TBDESession.
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.

<u>ExecSQLEx</u> (inherited from <u>TCustomDAConnection</u>)	Executes any SQL statement outside the TQuery or TSQL components.
<u>GetDatabaseNames</u> (inherited from <u>TCustomDAConnection</u>)	Returns a database list from the server.
<u>GetSequenceNames</u> (inherited from <u>TOraSession</u>)	Provides the names of available sequences.
<u>GetStoredProcNames</u> (inherited from <u>TCustomDAConnection</u>)	Returns a list of stored procedures from the server.
<u>GetTableNames</u> (inherited from <u>TCustomDAConnection</u>)	Provides a list of available tables names.
<u>InternalName</u> (inherited from <u>TOraSession</u>)	Used to get or set the client database name that will be recorded when performing global transactions.
<u>InTransaction</u> (inherited from <u>TCustomDAConnection</u>)	Indicates whether the transaction is active.
<u>LastError</u> (inherited from <u>TOraSession</u>)	Used to get an error code which resulted from previous call to the OCI interface function.
<u>LDA</u> (inherited from <u>TOraSession</u>)	Provides a pointer to Oracle 7 login data area of the current connection.
<u>LoginPrompt</u> (inherited from <u>TCustomDAConnection</u>)	Specifies whether a login dialog appears immediately before opening a new connection.
<u>MonitorMessage</u> (inherited from <u>TCustomDAConnection</u>)	Sends a specified message through the <u>TCustomDASQLMonitor</u> component.
<u>OCICallStyle</u> (inherited from <u>TOraSession</u>)	Indicates the set of OCI routines used.
<u>OCISvcCtx</u> (inherited from <u>TOraSession</u>)	Used to return Oracle 8 service context handle of the current connection.
<u>OnConnectionLost</u> (inherited from <u>TCustomDAConnection</u>)	This event occurs when connection was lost.
<u>OnError</u> (inherited from <u>TCustomDAConnection</u>)	This event occurs when an error has arisen in the connection.
<u>OracleVersion</u> (inherited from <u>TOraSession</u>)	Used to get Oracle server version number as string.
<u>ParamByName</u> (inherited from <u>TOraSession</u>)	Provides access to OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc.
<u>Ping</u> (inherited from <u>TOraSession</u>)	Checks whether the connection to the server can be established.
<u>Pooling</u> (inherited from <u>TCustomDAConnection</u>)	Enables or disables using connection pool.
<u>ProxySession</u> (inherited from <u>TOraSession</u>)	Used to enable multiple user sessions within a single database session.
<u>RemoveFromPool</u> (inherited from <u>TCustomDAConnection</u>)	Marks the connection that should not be returned to the pool after disconnect.
<u>Rollback</u> (inherited from <u>TCustomDAConnection</u>)	Discards all current data changes and ends transaction.
<u>RollbackToSavepoint</u> (inherited from <u>TOraSession</u>)	Cancels all updates for the current transaction.
<u>Savepoint</u> (inherited from <u>TOraSession</u>)	Defines a point in the transaction to which you can roll back later.

[SQL](#) (inherited from [TOraSession](#))

[StartTransaction](#) (inherited from [TOraSession](#))

Uses embedded ToraSQL object to execute any SQL statement.

Overloaded. Begins a new user transaction against the database server.

Published

Name	Description
AutoCommit (inherited from TOraSession)	Used to permit or prevent permanent updates, insertions, and deletions of data associated with the current transaction against the database server.
Connected (inherited from TOraSession)	Used to indicate if the database connection is active.
ConnectMode (inherited from TOraSession)	Used to specify the system privileges to use when a user connects to the server.
ConnectPrompt (inherited from TOraSession)	Used to supply a prompt for a name and password.
ConnectionString (inherited from TOraSession)	Used to assign the TOraSession.Username , TOraSession.Password and TOraSession.Server properties at once.
DatabaseName	Used to specify the name of the database to associate with this TBDESession component.
Debug (inherited from TOraSession)	Used to display SQL statements being executed with their parameter values and data types.
Home (inherited from TOraSession)	Not supported.
HomeName (inherited from TOraSession)	Used to select the Oracle client to use with the application.
OnConnectChange (inherited from TOraSession)	Occurs after Connected property was changed.
OnFailover (inherited from TOraSession)	Occurs when Transparent Application Failover (TAF) seamlessly attempts to failover to another Oracle instance.
Options (inherited from TOraSession)	Used to specify the behaviour of a TOraSession object.
Password (inherited from TOraSession)	Used to specify a password for a connection.
PoolingOptions (inherited from TOraSession)	Used to specify the behaviour of connection pool.
Schema (inherited from TOraSession)	Used to change the current schema of the session to the specified schema.
Server (inherited from TOraSession)	Contains the server name.
SessionName	Use the SessionName property to specify the session with which TBDESession component is associated.
ThreadSafety (inherited from TOraSession)	Used to allow the usage of the OCI in multi-threaded environment.
Username (inherited from TOraSession)	Contains username.

See Also

- [TBDESession Class](#)
- [TBDESession Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to access properties, events, and methods of the database component associated with this TBDESession.

Class

[TBDESession](#)

Syntax

```
property Database: TDatabase;
```

Remarks

Use the Database property to access properties, events, and methods of the database component associated with this TBDESession. Database is a read-only property, that is set automatically when the database specified by the DatabaseName property is opened.

© 1997-2012 Devart. All Rights Reserved.

Used to specify the name of the database to associate with this TBDESession component.

Class

[TBDESession](#)

Syntax

```
property DatabaseName: string;
```

Remarks

Use the DatabaseName property to specify the name of the database to associate with this TBDESession component. DatabaseName should match the name of a database component used in the application.

© 1997-2012 Devart. All Rights Reserved.

Use the SessionName property to specify the session with which TBDESession component is associated.

Class

[TBDESession](#)

Syntax

```
property SessionName: string;
```

Remarks

Use the SessionName property to specify the session with which TBDESession component is associated. SessionName is set automatically to the name of the SessionName property of the database component with which a dataset component is associated. If SessionName is blank, a dataset component is automatically associated with the default session, Session.

© 1997-2012 Devart. All Rights Reserved.

17.2 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 TCustomDADDataSet.BeforeFetch event.

Enumerations

Name	Description
TCRIsoIationLevel	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.

17.2.1 Classes

Classes in the **CRAccess** unit.

Classes

Name	Description
TCRCursor	A base class for classes that work with database cursors.

© 1997-2012 Devart. All Rights Reserved.

17.2.1.1 CRAccess.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

TObject

[TSharedObject](#)

TCRCursor

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

17.2.2 Types

Types in the **CRAccess** unit.

Types

Name	Description
TBeforeFetchProc	This type is used for the TCustomDADataset.BeforeFetch event.

© 1997-2012 Devart. All Rights Reserved.

17.2.2.1 CRAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.2.3 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.

© 1997-2012 Devart. All Rights Reserved.

17.2.3.1 CRAccess.TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

Unit

[CRAccess](#)

Syntax

```
TCRIsolationLevel = (ilReadCommitted);
```

Values

Value	Meaning
ilReadCommitted	The default transaction behavior. If the transaction contains DML that requires row locks held by another transaction, then the DML statement waits until the row locks are released.

© 1997-2012 Devart. All Rights Reserved.

17.2.3.2 CRAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.3 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.

17.3.1 Classes

Classes in the **CRBatchMove** unit.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

© 1997-2012 Devart. All Rights Reserved.

17.3.1.1 CRBatchMove.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 Oracle Access page.

Inheritance Hierarchy

TObject

TCRBatchMove

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to get the number of records changed in the destination dataset.

Class

[TCRBatchMove](#)

Syntax

```
property ChangedCount: Longint;
```

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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: Longint;
```

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](#)

© 1997-2012 Devart. All Rights Reserved.

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
```

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to get the number of records that were read from the source dataset during the batch operation.

Class

[TCRBatchMove](#)

Syntax

```
property MovedCount: Longint;
```

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.

© 1997-2012 Devart. All Rights Reserved.

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: Longint;
```

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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: Longint 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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Performs the batch operation.

Class

[TCRBatchMove](#)

Syntax

procedure Execute;

Remarks

Call the Execute method to perform the batch operation.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

17.3.2 Types

Types in the **CRBatchMove** unit.

Types

Name	Description
TCRBatchMoveProgressEvent	This type is used for the TCRBatchMove.OnBatchMoveProgress event.

© 1997-2012 Devart. All Rights Reserved.

17.3.2.1 CRBatchMove.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.

© 1997-2012 Devart. All Rights Reserved.

17.3.3 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.

© 1997-2012 Devart. All Rights Reserved.

17.3.3.1 CRBatchMove.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.

© 1997-2012 Devart. All Rights Reserved.

17.3.3.2 CRBatchMove.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.

© 1997-2012 Devart. All Rights Reserved.

17.4 CRDataTypeMap

This unit contains base classes for Data Type Mapping

Classes

Name	Description
EDataMappingError	Occurs when unable to map data to a specified type.
EDataTypeMappingError	Base class for errors occurring at data mapping
EInvalidDBTypeMapping	Occurs when DB field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties.
EInvalidFieldTypeMapping	Occurs when Delphi field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties.
EUnsupportedDataTypeMapping	Occurs when attempting to register or perform unsupported data type mapping.
TMapRule	Setting rule for data type mapping

17.4.1 Classes

Classes in the **CRDataTypeMap** unit.

Classes

Name	Description
EDataMappingError	Occurs when unable to map data to a specified type.
EDataTypeMappingError	Base class for errors occurring at data mapping
EInvalidDBTypeMapping	Occurs when DB field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties.
EInvalidFieldTypeMapping	Occurs when Delphi field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties.
EUnsupportedDataTypeMapping	Occurs when attempting to register or perform unsupported data type mapping.
TMapRule	Setting rule for data type mapping

© 1997-2012 Devart. All Rights Reserved.

17.4.1.1 CRDataTypeMap.EDataMappingError Class

Occurs when unable to map data to a specified type.

For a list of all members of this type, see [EDataMappingError](#) members.

Unit

[CRDataTypeMap](#)

Syntax

```
EDataMappingError = class (EDataTypeMappingError) ;
```

Remarks

EDataMappingError occurs when unable to map data to a specified type. Use EDataMappingError in an exception handling block.

Inheritance Hierarchy

TObject

[EDataTypeMappingError](#)

EDataMappingError

© 1997-2012 Devart. All Rights Reserved.

[EDataMappingError](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.4.1.2 CRDataTypeMap.EDataTypeMappingError Class

Base class for errors occurring at data mapping

For a list of all members of this type, see [EDataTypeMappingError](#) members.

Unit

[CRDataTypeMap](#)

Syntax

```
EDataTypeMappingError = class (Exception) ;
```

Remarks

Base class for errors occurring at data mapping

Inheritance Hierarchy

TObject

EDataTypeMappingError

© 1997-2012 Devart. All Rights Reserved.

[EDataTypeMappingError](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.4.1.3 CRDataTypeMap.EInvalidDBTypeMapping Class

Occurs when DB field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties.

For a list of all members of this type, see [EInvalidDBTypeMapping](#) members.

Unit

[CRDataTypeMap](#)

Syntax

```
EInvalidDBTypeMapping = class (EDataTypeMappingError) ;
```

Remarks

EInvalidDBTypeMapping occurs when DB field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties. Use EInvalidDBTypeMapping in an exception handling block.

Inheritance Hierarchy

TObject

[EDataTypeMappingError](#)

EInvalidDBTypeMapping

© 1997-2012 Devart. All Rights Reserved.

[EInvalidDBTypeMapping](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.4.1.4 CRDataTypeMap.EInvalidFieldTypeMapping Class

Occurs when Delphi field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties.

For a list of all members of this type, see [EInvalidFieldTypeMapping](#) members.

Unit

[CRDataTypeMap](#)

Syntax

```
EInvalidFieldTypeMapping = class (EDataTypeMappingError) ;
```

Remarks

EInvalidFieldTypeMapping occurs when Delphi field type is set incorrectly or when attempting to set Length or Scale for a type that doesn't have such properties. Use EInvalidFieldTypeMapping in an exception handling block.

Inheritance Hierarchy

TObject

[EDataTypeMappingError](#)

EInvalidFieldTypeMapping

© 1997-2012 Devart. All Rights Reserved.

[EInvalidFieldTypeMapping](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.4.1.5 CRDataTypeMap.EUnsupportedDataTypeMapping Class

Occurs when attempting to register or perform unsupported data type mapping.

For a list of all members of this type, see [EUnsupportedDataTypeMapping](#) members.

Unit

[CRDataTypeMap](#)

Syntax

```
EUnsupportedDataTypeMapping = class(EDataTypeMappingError);
```

Remarks

EUnsupportedDataTypeMapping occurs when attempting to register or perform unsupported data type mapping. Use EUnsupportedDataTypeMapping in an exception handling block.

Inheritance Hierarchy

TObject

[EDataTypeMappingError](#)

EUnsupportedDataTypeMapping

© 1997-2012 Devart. All Rights Reserved.

[EUnsupportedDataTypeMapping](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.4.1.6 CRDataTypeMap.TMapRule Class

Setting rule for data type mapping

For a list of all members of this type, see [TMapRule](#) members.

Unit

[CRDataTypeMap](#)

Syntax

```
TMapRule = class(TCollectionItem);
```

Inheritance Hierarchy

TObject

TMapRule

© 1997-2012 Devart. All Rights Reserved.

[TMapRule](#) class overview.

Properties

Name	Description
DBLengthMax	Maximum DB field size
DBLengthMin	Minimum DB field size
DBScaleMax	Maximum DB field scale
DBScaleMin	Minimal DB field scale
DBType	DB type
FieldLength	Delphi field length
FieldName	field name in DataSet

[FieldScale](#)
[IgnoreErrors](#)

Delphi field scale
Ignore data conversion errors.
Default value is False.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TMapRule** class.

For a complete list of the **TMapRule** class members, see the [TMapRule Members](#) topic.

Public

Name	Description
DBLengthMax	Maximum DB field size
DBLengthMin	Minimum DB field size
DBScaleMax	Maximum DB field scale
DBScaleMin	Minimal DB field scale
DBType	DB type
FieldLength	Delphi field length
FieldName	field name in DataSet
FieldScale	Delphi field scale
IgnoreErrors	Ignore data conversion errors. Default value is False.

See Also

- [TMapRule Class](#)
- [TMapRule Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Maximum DB field size

Class

[TMapRule](#)

Syntax

```
property DBLengthMax: Integer;
```

© 1997-2012 Devart. All Rights Reserved.

Minimum DB field size

Class

[TMapRule](#)

Syntax

```
property DBLengthMin: Integer;
```

© 1997-2012 Devart. All Rights Reserved.

Maximum DB field scale

Class

[TMapRule](#)

Syntax

```
property DBScaleMax: Integer;
```

© 1997-2012 Devart. All Rights Reserved.

Minimal DB field scale

Class

[TMapRule](#)

Syntax

```
property DBScaleMin: Integer;
```

© 1997-2012 Devart. All Rights Reserved.

DB type

Class

[TMapRule](#)

Syntax

```
property DBType: Word;
```

© 1997-2012 Devart. All Rights Reserved.

Delphi field length

Class

[TMapRule](#)

Syntax

```
property FieldLength: Integer;
```

© 1997-2012 Devart. All Rights Reserved.

field name in DataSet

Class

[TMapRule](#)

Syntax

```
property FieldName: string;
```

© 1997-2012 Devart. All Rights Reserved.

Delphi field scale

Class

[TMapRule](#)

Syntax

```
property FieldScale: Integer;
```

© 1997-2012 Devart. All Rights Reserved.

Ignore data conversion errors. Default value is False.

Class

[TMapRule](#)

Syntax

```
property IgnoreErrors: Boolean;
```

17.5 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.

17.5.1 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 .

© 1997-2012 Devart. All Rights Reserved.

17.5.1.1 CREncryption.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) ;
```

Inheritance Hierarchy

TObject
TCREncryptor

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#).

© 1997-2012 Devart. All Rights Reserved.

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](#).

© 1997-2012 Devart. All Rights Reserved.

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](#).

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to set a password that is used to generate a key for encryption.

Class

[TCREncryptor](#)

Syntax

```
property Password: string;
```

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

17.5.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.5.2.1 CREncryption.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.

© 1997-2012 Devart. All Rights Reserved.

17.5.2.2 CREncryption.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.

© 1997-2012 Devart. All Rights Reserved.

17.5.2.3 CREncryption.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.

© 1997-2012 Devart. All Rights Reserved.

17.5.2.4 CREncryption.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.

© 1997-2012 Devart. All Rights Reserved.

17.6 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.

17.6.1 Classes

Classes in the **DAAlerter** unit.

Classes

Name	Description
TDAAlerter	A base class that defines functionality for database event notification.

© 1997-2012 Devart. All Rights Reserved.

17.6.1.1 DAAlerter.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.

Inheritance Hierarchy

TObject
TDAAlerter

© 1997-2012 Devart. All Rights Reserved.

[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

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to determine if TDAAlerter waits for messages.

Class

[TDAAlerter](#)

Syntax

```
property Active: boolean default False;
```

Remarks

Check the Active property to know whether TDAAlerter waits for messages or not. Set it to True to register events.

See Also

- [Start](#)
- [Stop](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to specify the connection for TDAAlerter.

Class

[TDAAlerter](#)

Syntax

property Connection: [TCustomDAConnection](#);

Remarks

Use the Connection property to specify the connection for TDAAlerter.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Stops waiting process.

Class

[TDAAlerter](#)

Syntax

```
procedure Stop;
```

Remarks

Call Stop method to end waiting process.

See Also

- [Start](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.6.2 Types

Types in the **DAAlerter** unit.

Types

Name	Description
TAlerterErrorEvent	This type is used for the TDAAlerter.OnError event.

© 1997-2012 Devart. All Rights Reserved.

17.6.2.1 DAAlerter.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.

© 1997-2012 Devart. All Rights Reserved.

17.7 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.

17.7.1 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.

© 1997-2012 Devart. All Rights Reserved.

17.7.1.1 DADump.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 descendants 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.

Inheritance Hierarchy

TObject
TDADump

© 1997-2012 Devart. All Rights Reserved.

[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 executing statement, all its parameters' values, and the type of 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.

© 1997-2012 Devart. All Rights Reserved.

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 executing statement, all its parameters' values, and the type of 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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to display executing statement, all its parameters' values, and the type of parameters.

Class

[TDADump](#)

Syntax

property Debug: boolean **default** False;

Remarks

Used to display executing statement, all its parameters' values, and the type of parameters.

See Also

- [TCustomDADataset.Debug](#)
- [TCustomDASQL.Debug](#)

© 1997-2012 Devart. All Rights Reserved.

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.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

©

1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 commas. If it is empty, the [Backup](#) method will dump all available tables.

See Also

- [Backup](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Executes a script from a file.

Class

[TDADump](#)

Syntax

```
procedure RestoreFromFile(const FileName: string);
```

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.7.1.2 DADump.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);
```

Inheritance Hierarchy

```
TObject
  TDADumpOptions
```

© 1997-2012 Devart. All Rights Reserved.

[TDADumpOptions](#) class overview.

Properties

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

© 1997-2012 Devart. All Rights Reserved.

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.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.7.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.7.2.1 DADump.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.

© 1997-2012 Devart. All Rights Reserved.

17.7.2.2 DADump.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.

© 1997-2012 Devart. All Rights Reserved.

17.8 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.

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.

17.8.1 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.

© 1997-2012 Devart. All Rights Reserved.

17.8.1.1 DALoader.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.

Inheritance Hierarchy

```

Object
  TDAColumn
  
```

See Also

- [TDALoader](#)
- [TDAColumns](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.8.1.2 DALoader.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.

Inheritance Hierarchy

TObject
TDAColumns

See Also

- [TDALoader](#)
- [TDAColumn](#)

© 1997-2012 Devart. All Rights Reserved.

[TDAColumns](#) class overview.

Properties

Name	Description
Items	Used to access individual columns.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.8.1.3 DALoader.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.

Inheritance Hierarchy

TObject
 TDALoader

See Also

- [TOraLoader Component](#)
- [TOraLoader](#)

© 1997-2012 Devart. All Rights Reserved.

[TDALoader](#) class overview.

Properties

Name	Description
Columns	Used to add a TDAColumn object for each field that will be loaded.
Connection	Used to specify TCustomDACConnection in which TDALoader will be executed.
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.

© 1997-2012 Devart. All Rights Reserved.

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	Used to specify TCustomDAConnection in which TDALoader will be executed.
TableName	Used to specify the name of the table to which data will be loaded.

See Also

- [TDALoader Class](#)
- [TDALoader Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to specify TCustomDAConnection in which TDALoader will be executed.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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*

Holds the name of a loading column.

Row

Holds the number of loading row. Row starts from 1.

Value

Holds the column value.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.8.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.8.2.1 DALoader.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.

© 1997-2012 Devart. All Rights Reserved.

17.8.2.2 DALoader.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.

© 1997-2012 Devart. All Rights Reserved.

17.8.2.3 DALoader.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.

17.9 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.
TDASStatement	This class has attributes and methods for controlling single SQL statement of a script.
TDASStatements	Holds a collection of TDASStatement 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.

17.9.1 Classes

Classes in the **DAScript** unit.

Classes

Name	Description
TDAScript	Makes it possible to execute several SQL statements one by one.
TDASStatement	This class has attributes and methods for controlling single SQL statement of a script.
TDASStatements	Holds a collection of TDASStatement objects.

© 1997-2012 Devart. All Rights Reserved.

17.9.1.1 DAScript.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 descendant 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.

Inheritance Hierarchy

TObject
 TDAScript

See Also

- [TCustomDASQL](#)

© 1997-2012 Devart. All Rights Reserved.

[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	Indicates whether a specified macro exists in a dataset.
MacroByName	Finds a Macro with the name passed in 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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 script execution and all its parameter values. Also displays the type of parameters.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to get the end position of the current statement.

Class

[TDA Script](#)

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.

© 1997-2012 Devart. All Rights Reserved.

Used to change SQL script text in design- or run-time easily.

Class

[TDA Script](#)

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to get or set script text.

Class

[TDA Script](#)

Syntax

```
property SQL: _TStrings;
```

Remarks

Use the SQL property to get or set script text.

© 1997-2012 Devart. All Rights Reserved.

Used to get the current statement start line number in a script.

Class

[TDA Script](#)

Syntax

property StartLine: Int64;

Remarks

Use the StartLine property to get the current statement start line number in a script.

© 1997-2012 Devart. All Rights Reserved.

Used to get the offset in the first line of the current statement.

Class

[TDA Script](#)

Syntax

property StartOffset: Int64;

Remarks

Use the StartOffset property to get the offset in the first line of the current statement.

© 1997-2012 Devart. All Rights Reserved.

Used to get the start position of the current statement in a script.

Class

[TDA Script](#)

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.

© 1997-2012 Devart. All Rights Reserved.

Contains a list of statements obtained from the SQL property.

Class

[TDA Script](#)

Syntax

property Statements: [TDASentences](#);

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;
        end;
```

See Also

- [TDASentences](#)

© 1997-2012 Devart. All Rights Reserved.

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	Indicates whether a specified macro exists in a dataset.
MacroByName	Finds a Macro with the name passed in Name.

See Also

- [TDAScript Class](#)
- [TDAScript Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Stops script execution.

Class

[TDAScript](#)

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Executes the next statement in the script and then stops.

Class

[TDA Script](#)

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](#)

© 1997-2012 Devart. All Rights Reserved.

Executes SQL statements contained in a stream object.

Class

[TDA Script](#)

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.

© 1997-2012 Devart. All Rights Reserved.

Indicates whether a specified macro exists in a dataset.

Class

[TDA Script](#)

Syntax

```
function FindMacro(Name: string): TMacro;  
Parameters
```

Name

Holds the name of the macro to search for.

Return Value

a TMacro object, if a macro with matching name was found, otherwise returns nil.

Remarks

Call the FindMacro method to determine if a specified macro exists. If FindMacro finds a macro with a

matching name, it returns a TMacro object for the specified Name. Otherwise it returns nil.

See Also

- [TMacro](#)
 - [Macros](#)
 - [MacroByName](#)
-

© 1997-2012 Devart. All Rights Reserved.

Finds a Macro with the name passed in Name.

Class

[TDAScript](#)

Syntax

```
function MacroByName (Name: string): TMacro;
```

Parameters

Name

Holds the name of the Macro to search for.

Return Value

the Macro, if a match was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Name. If a match was found, MacroByName returns the Macro. 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 FindMacro method.

To assign the value of macro use the [TMacro.Value](#) property.

See Also

- [TMacro](#)
 - [Macros](#)
 - [FindMacro](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Occurs after a SQL script execution.

Class

[TDAScript](#)

Syntax

property AfterExecute: [TAfterStatementExecuteEvent](#);

Remarks

Occurs after a SQL script has been executed.

See Also

- [Execute](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.9.1.2 DAScript.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

```
TDASstatement = class (TCollectionItem);
```

Remarks

TDAScript contains SQL statements, represented as TDASstatement objects. The TDASstatement class has attributes and methods for controlling single SQL statement of a script.

Inheritance Hierarchy

TObject
 TDASstatement

See Also

- [TDAScript](#)
- [TDASstatements](#)

© 1997-2012 Devart. All Rights Reserved.

[TDASstatement](#) 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 parameters 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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TDASstatement** class.

For a complete list of the **TDASstatement** class members, see the [TDASstatement 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 parameters 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

- [TDASStatement Class](#)
- [TDASStatement Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to determine the number of the last statement line in a script.

Class

[TDASStatement](#)

Syntax

```
property EndLine: integer;
```

Remarks

Use the EndLine property to determine the number of the last statement line in a script.

© 1997-2012 Devart. All Rights Reserved.

Used to get the offset in the last line of the statement.

Class

[TDASStatement](#)

Syntax

```
property EndOffset: integer;
```

Remarks

Use the EndOffset property to get the offset in the last line of the statement.

© 1997-2012 Devart. All Rights Reserved.

Used to get the end position of the statement in a script.

Class

[TDASStatement](#)

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.

© 1997-2012 Devart. All Rights Reserved.

Used to avoid execution of a statement.

Class

[TDASentence](#)

Syntax

property Omit: boolean;

Remarks

Set the Omit property to True to avoid execution of a statement.

© 1997-2012 Devart. All Rights Reserved.

Contains parameters for an SQL statement.

Class

[TDASentence](#)

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to determine the TDAScript object the SQL Statement belongs to.

Class

[TDASentence](#)

Syntax

property Script: [TDAScript](#);

Remarks

Use the Script property to determine the TDAScript object the SQL Statement belongs to.

© 1997-2012 Devart. All Rights Reserved.

Used to get or set the text of an SQL statement.

Class

[TDASentence](#)

Syntax

property SQL: string;

Remarks

Use the SQL property to get or set the text of an SQL statement.

© 1997-2012 Devart. All Rights Reserved.

Used to determine the number of the first statement line in a script.

Class

[TDASentence](#)

Syntax

```
property StartLine: integer;
```

Remarks

Use the StartLine property to determine the number of the first statement line in a script.

© 1997-2012 Devart. All Rights Reserved.

Used to get the offset in the first line of a statement.

Class

[TDASentence](#)

Syntax

```
property StartOffset: integer;
```

Remarks

Use the StartOffset property to get the offset in the first line of a statement.

© 1997-2012 Devart. All Rights Reserved.

Used to get the start position of the statement in a script.

Class

[TDASentence](#)

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.

© 1997-2012 Devart. All Rights Reserved.

17.9.1.3 DAScript.TDASentences Class

Holds a collection of [TDASentence](#) objects.

For a list of all members of this type, see [TDASentences](#) members.

Unit

[DAScript](#)

Syntax

```
TDASentences = class (TCollection);
```

Remarks

Each TDASentences holds a collection of [TDASentence](#) objects. TDASentences maintains an index of the statements in its Items array. The Count property contains the number of statements in the collection. Use TDASentences class to manipulate script SQL statements.

Inheritance Hierarchy

TObject

TDASentences

See Also

- [TDAScript](#)
 - [TDASStatement](#)
-

© 1997-2012 Devart. All Rights Reserved.

[TDASStatements](#) class overview.

Properties

Name	Description
Items	Used to access separate script statements.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TDASStatements** class.

For a complete list of the **TDASStatements** class members, see the [TDASStatements Members](#) topic.

Public

Name	Description
Items	Used to access separate script statements.

See Also

- [TDASStatements Class](#)
 - [TDASStatements Class Members](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to access separate script statements.

Class

[TDASStatements](#)

Syntax

```
property Items[Index: Integer]: TDASStatement; 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 [TDASStatement](#).

See Also

- [TDASStatement](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.9.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.9.2.1 DAScript.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.

© 1997-2012 Devart. All Rights Reserved.

17.9.2.2 DAScript.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.

© 1997-2012 Devart. All Rights Reserved.

17.9.2.3 DAScript.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.

17.9.3 Enumerations

Enumerations in the **DAScript** unit.

Enumerations

Name	Description
TErrorAction	Indicates the action to take when the OnError handler exits.

© 1997-2012 Devart. All Rights Reserved.

17.9.3.1 DAScript.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.

© 1997-2012 Devart. All Rights Reserved.

17.10 DASQLMonitor

This unit contains the base class for the TOraSQLMonitor component.

Classes

Name	Description
<u>TCustomDASQLMonitor</u>	A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.
<u>TDBMonitorOptions</u>	This class holds options for dbMonitor.

Types

Name	Description
<u>TDATraceFlags</u>	Represents the set of <u>TDATraceFlag</u> .
<u>TMonitorOptions</u>	Represents the set of <u>TMonitorOption</u> .
<u>TOnSQLEvent</u>	This type is used for the <u>TCustomDASQLMonitor.OnSQL</u> event.

Enumerations

Name	Description
<u>TDATraceFlag</u>	Use TraceFlags to specify which database operations the monitor should track in an application at runtime.
<u>TMonitorOption</u>	Used to define where information from SQLMonitor will be displayed.

17.10.1 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.

© 1997-2012 Devart. All Rights Reserved.

17.10.1.1 DASQLMonitor.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.

Inheritance Hierarchy

TObject
 TCustomDASQLMonitor

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to set options for dbMonitor.

Class

[TCustomDASQLMonitor](#)

Syntax

```
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.10.1.2 DASQLMonitor.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);
```

Inheritance Hierarchy

TObject

TDBMonitorOptions

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.10.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.10.2.1 DASQLMonitor.TDATraceFlags Set

Represents the set of [TDATraceFlag](#).

Unit

[DASQLMonitor](#)

Syntax

```
TDATraceFlags = set of TDATraceFlag;
```

© 1997-2012 Devart. All Rights Reserved.

17.10.2.2 DASQLMonitor.TMonitorOptions Set

Represents the set of [TMonitorOption](#).

Unit

[DASQLMonitor](#)

Syntax

```
TMonitorOptions = set of TMonitorOption;
```

© 1997-2012 Devart. All Rights Reserved.

17.10.2.3 DASQLMonitor.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.

© 1997-2012 Devart. All Rights Reserved.

17.10.3 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.

© 1997-2012 Devart. All Rights Reserved.

17.10.3.1 DASQLMonitor.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.

© 1997-2012 Devart. All Rights Reserved.

17.10.3.2 DASQLMonitor.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.

17.11 DBAccess

This unit contains base classes for most of the components.

Classes

Name	Description
EDAEError	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.
TDACConnectionOptions	This class allows setting up the behaviour of the TDACConnection class.
TDADatasetOptions	This class allows setting up the behaviour of the TDADataset class.
TDAEncryptionOptions	Used to specify the options of the data encryption in a dataset.
TDAMapRule	Class that forms 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.

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
TLabelSet	Sets the language of labels in the connect dialog.
TLockMode	Specifies when the locking of an editing record should be performed.
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.
MacroChar	Determinates what character is used for macros.

[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.

17.11.1 Classes

Classes in the **DBAccess** unit.

Classes

Name	Description
EDAEError	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.
TDACConnectionOptions	This class allows setting up the behaviour of the TDACConnection class.
TDADatasetOptions	This class allows setting up the behaviour of the TDADataset class.
TDAEncryptionOptions	Used to specify the options of the data encryption in a dataset.
TDAMapRule	Class that forms 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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.1 DBAccess.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.

Inheritance Hierarchy

TObject
EDAError

© 1997-2012 Devart. All Rights Reserved.

[EDAError](#) class overview.

Properties

Name	Description
Component	Contains the component that caused the error.
ErrorCode	Determines the error code returned by the server.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Contains the component that caused the error.

Class

[EDAError](#)

Syntax

```
property Component: TObject;
```

Remarks

The Component property contains the component that caused the error.

© 1997-2012 Devart. All Rights Reserved.

Determines the error code returned by the server.

Class

[EDAEError](#)

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.11.1.2 DBAccess.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.

Inheritance Hierarchy

TObject

TCRDataSource

© 1997-2012 Devart. All Rights Reserved.

[TCRDataSource](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.3 DBAccess.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.

Inheritance Hierarchy

TObject
TCustomConnectDialog

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 lsEnglish.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.11.1.4 DBAccess.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.

Inheritance Hierarchy

TObject

TCustomDAConnection

© 1997-2012 Devart. All Rights Reserved.

[TCustomDAConnection](#) class overview.

Properties

Name	Description
ConnectDialog	Allows to link a TCustomConnectDialog component.
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.
CreateDataSet	Creates a dataset component.
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.
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.
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.

© 1997-2012 Devart. All Rights Reserved.

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.
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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Specifies whether a login dialog appears immediately before opening a new connection.

Class

[TCustomDAConnection](#)

Syntax

```
property LoginPrompt default True;
```

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.

© 1997-2012 Devart. All Rights Reserved.

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
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](#)

©

1997-2012 Devart. All Rights Reserved.

Serves to supply a password for login.

Class

[TCustomDAConnection](#)

Syntax

```
property Password: string;
```

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](#)

© 1997-2012 Devart. All Rights Reserved.

Enables or disables using connection pool.

Class

[TCustomDAConnection](#)

Syntax

```
property Pooling: boolean default False;
```

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 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 the connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

- [Pooling](#)

©

1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.
CreateDataSet	Creates a dataset component.
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.

[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.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

Applies changes in datasets.

Class

[TCustomDAConnection](#)

Overload List

Name	Description
ApplyUpdates	Applies changes from all active datasets.
ApplyUpdates(DataSets: array of TCustomDADataSet)	Applies changes from the specified datasets.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Applies changes from the specified datasets.

Class

[TCustomDAConnection](#)

Syntax

```
procedure ApplyUpdates(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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Establishes a connection to the server.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Connect;
```

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](#)

© 1997-2012 Devart. All Rights Reserved.

Creates a dataset component.

Class

[TCustomDAConnection](#)

Syntax

```
function CreateDataSet: TCustomDADataSet; virtual;  
Return Value
```

Returns a new instance of the class.

Remarks

Call the CreateDataSet method to return a new instance of the [TCustomDADataSet](#) class and associate it with this connection object. In the descendant classes this method should be overridden to create an appropriate descendant of the TCustomDADataSet component.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Allows to execute stored procedure or function providing its name and parameters.

Class

[TCustomDAConnection](#)

Syntax

```
function ExecProc(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)

See Also

-

[ExecSQL](#)

- [ExecSQLEx](#)
 - [ExecProc](#)
-

© 1997-2012 Devart. All Rights Reserved.

Executes a SQL statement with parameters.

Class

[TCustomDAConnection](#)

Syntax

```
function ExecSQL(Text: string): variant; overload;function ExecSQL  
(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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Executes any SQL statement outside the TQuery or TSQL components.

Class

[TCustomDAConnection](#)

Syntax

```
function ExecSQLEx(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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 procedures (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](#)
 - [GetTableNames](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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: you should explicitly add the [MemData](#) unit to the 'uses' list to use the OnConnectionLost event handler.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.5 DBAccess.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

TObject

[TMemDataSet](#)

TCustomDADataset

© 1997-2012 Devart. All Rights Reserved.

[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.
Connection	Used to specify a connection object to use to connect to a data store.
Debug	Used to display executing statement, all its parameters' values, and the type of 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.
Encryption	Used to specify the options of the data encryption in a dataset.
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.
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.

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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec	Breaks execution of a SQL statement on the server.
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.

Execute	Executes a SQL statement on the server.
Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to learn whether TCustomDADataset has already 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	Indicates whether a specified macro exists in a dataset.
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.
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 name passed in 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.
Resync	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
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.
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.

Public

Name	Description
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
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.
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.
Connection	Used to specify a connection object to use to connect to a data store.
Debug	Used to display executing statement, all its parameters' values, and the type of parameters.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
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.
Encryption	Used to specify the options of the data encryption in a dataset.
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.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
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.
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.

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.
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.
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.
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.
Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
ReadOnly	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions	Used to indicate when the editing record is refreshed.
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.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SaveToXML (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

[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.

[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.

[UnPrepare](#) (inherited from [TMemDataSet](#))

Frees the resources allocated for a previously prepared query on the server and client sides.

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

Used to indicate the update status for the current record when cached updates are enabled.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TCustomDADataset Class](#)
- [TCustomDADataset Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to display executing statement, all its parameters' values, and the type of parameters.

Class

[TCustomDADataset](#)

Syntax

property Debug: boolean **default** False;

Remarks

Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

You should add the OdacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: To enable debug window you should explicitly include the OdacVcl (OdacClx under Kylix) unit to your project.

See Also

- [TCustomDASQL.Debug](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to specify the options of the data encryption in a dataset.

Class

[TCustomDADataset](#)

Syntax

```
property Encryption: TDAEncryptionOptions;
```

Remarks

Set the properties of Encryption to specify the options of the data encryption in a dataset.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

Example

```
Query1.FilterSQL := 'Dept >= 20 and DName LIKE ''M%''';
```

See Also

-

[AddWhere](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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:= 'SELECT * FROM Dept ORDER BY &Order';  
OraQuery.MacroByName('Order').Value:= 'DeptNo';  
OraQuery.Open;
```

See Also

- [TMacro](#)
- [MacroByName](#)
- [Params](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
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.
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.
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](#)

©

1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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. The default value of `UniDirectional` is `False`, enabling forward and backward navigation.

See Also

- [TOraDataSet.FetchAll](#)

© 1997-2012 Devart. All Rights Reserved.

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.

ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates 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.
Execute	Executes a SQL statement on the server.
Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to learn whether TCustomDADataset has already 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	Indicates whether a specified macro exists in a dataset.
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.
GetOrderBy	Retrieves an ORDER BY clause from a SQL statement.

[GotoCurrent](#)

[IndexFieldNames](#) (inherited from [TMemDataSet](#))

[LocalConstraints](#) (inherited from [TMemDataSet](#))

[LocalUpdate](#) (inherited from [TMemDataSet](#))

[Locate](#) (inherited from [TMemDataSet](#))

[LocateEx](#) (inherited from [TMemDataSet](#))

[Lock](#)

[MacroByName](#)

[OnUpdateError](#) (inherited from [TMemDataSet](#))

[OnUpdateRecord](#) (inherited from [TMemDataSet](#))

[ParamByName](#)

[Prepare](#)

[Prepared](#) (inherited from [TMemDataSet](#))

[RefreshRecord](#)

[RestoreSQL](#)

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[Resync](#)

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveSQL](#)

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SetOrderBy](#)

[SQLSaved](#)

[UnLock](#)

Sets the current record in this dataset similar to the current record in another dataset.

Used to get or set the list of fields on which the recordset is sorted.

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Used to prevent implicit update of rows on database server.

Overloaded. Searches a dataset for a specific record and positions the cursor on it.

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Locks the current record.

Finds a Macro with the name passed in Name.

Occurs when an exception is generated while cached updates are applied to a database.

Occurs when a single update component can not handle the updates.

Sets or uses parameter information for a specific parameter based on its name.

Allocates, opens, and parses cursor for a query.

Determines whether a query is prepared for execution or not.

Actualizes field values for the current record.

Restores the SQL property modified by AddWhere and SetOrderBy.

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Cancels changes made to the current record when cached updates are enabled.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Builds an ORDER BY clause of a SELECT statement.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Releases a record lock.

[UnPrepare](#) (inherited from [TMemDataSet](#))

Frees the resources allocated for a previously prepared query on the server and client sides.

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

Used to indicate the update status for the current record when cached updates are enabled.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TCustomDADataset Class](#)
- [TCustomDADataset Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Adds condition to the WHERE clause of SELECT statement in the SQL property.

Class

[TCustomDADataset](#)

Syntax

```
procedure AddWhere (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.

See Also

- [DeleteWhere](#)

© 1997-2012 Devart. All Rights Reserved.

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

- [Execute](#)
- [TOraDataSet.NonBlocking](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Executes a SQL statement on the server.

Class

[TCustomDADataset](#)

Syntax


```
procedure Execute; virtual;
```

Remarks

Call the Execute method to execute a SQL statement on the server. If SQL statement is a query, Execute calls the Open method.

Execute implicitly prepares SQL statement by calling the [Prepare](#) method if the [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

- [AfterExecute](#)
- [Executing](#)
- [Prepare](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to learn whether TCustomDADataset has already fetched all rows.

Class

[TCustomDADataset](#)

Syntax

```
function Fetched: boolean; virtual;
```

Return Value

True, if all rows are fetched.

Remarks

Check Fetched to learn whether TCustomDADataset has already fetched all rows.

See Also

- [Fetching](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Indicates whether a specified macro exists in a dataset.

Class

[TCustomDADataset](#)

Syntax

```
function FindMacro(const Value: string): TMacro;
```

Parameters

Value

Holds the name of the macro to search for.

Return Value

a TMacro object, if a macro with matching name was found, otherwise returns nil.

Remarks

Call the FindMacro method to determine if a specified macro exists. If FindMacro finds a macro with a matching name, it returns a TMacro object for the specified Name. Otherwise it returns nil.

See Also

- [TMacro](#)
- [Macros](#)
- [MacroByName](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Returns a multireference shared object from field.

Class

[TCustomDADataset](#)

Syntax

```
function GetFieldObject(Field: TField): TSharedObject; overload;  
function GetFieldObject(FieldDesc: TFieldDesc): 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
- [TCustomOraQuery.SQLLock](#)
- [Unlock](#)

© 1997-2012 Devart. All Rights Reserved.

Finds a Macro with the name passed in Name.

Class

[TCustomDADataset](#)

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of the Macro to search for.

Return Value

the Macro, if a match was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Name. If a match was found, MacroByName returns the Macro. 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 FindMacro method.

To assign the value of 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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Class

[TCustomDADataset](#)

Syntax

```
procedure Resync (Mode: TResyncMode); override;
```

Parameters

Mode

Holds optional processing that Resync should handle.

Remarks

Resync is used to resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Builds an ORDER BY clause of a SELECT statement.

Class

[TCustomDADataset](#)

Syntax

```
procedure SetOrderBy(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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
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.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates 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.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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.

[Locate](#) (inherited from [TMemDataSet](#))

[LocateEx](#) (inherited from [TMemDataSet](#))

[OnUpdateError](#) (inherited from [TMemDataSet](#))

[OnUpdateRecord](#) (inherited from [TMemDataSet](#))

[Prepare](#) (inherited from [TMemDataSet](#))

[Prepared](#) (inherited from [TMemDataSet](#))

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Overloaded. Searches a dataset for a specific record and positions the cursor on it.

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Occurs when an exception is generated while cached updates are applied to a database.

Occurs when a single update component can not handle the updates.

Allocates resources and creates field components for a dataset.

Determines whether a query is prepared for execution or not.

Marks all records in the cache of updates as unapplied.

Cancels changes made to the current record when cached updates are enabled.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Frees the resources allocated for a previously prepared query on the server and client sides.

Used to indicate the update status for the current record when cached updates are enabled.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Used to check the status of the cached updates buffer.

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TCustomDADataset Class](#)
- [TCustomDADataset Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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

- [Execute](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.11.1.6 DBAccess.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.

Inheritance Hierarchy

TObject
TCustomDASQL

© 1997-2012 Devart. All Rights Reserved.

[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 executing statement, all its parameters' values, and the type of 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
Execute	Overloaded. Executes SQL commands.
Executing	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro	Searches for a macro with the specified name.
FindParam	Finds a parameter with the specified name.
MacroByName	Finds a Macro with the name passed in 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.

© 1997-2012 Devart. All Rights Reserved.

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 executing statement, all its parameters' values, and the type of 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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to display executing statement, all its parameters' values, and the type of parameters.

Class

[TCustomDASQL](#)

Syntax

property Debug: boolean **default** False;

Remarks

Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

You should add the OdacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: To enable debug window you should explicitly include the OdacVcl (OdacClx under Kylix) unit to your project.

See Also

- [TCustomDADataset.Debug](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to get or set the values of individual field parameters that are identified by name.

Class

[TCustomDASQL](#)

Syntax

```
property ParamValues[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TCustomDASQL** class.

For a complete list of the **TCustomDASQL** class members, see the [TCustomDASQL Members](#) topic.

Public

Name	Description
Execute	Overloaded. Executes SQL commands.
Executing	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro	Searches for a macro with the specified name.
FindParam	Finds a parameter with the specified name.
MacroByName	Finds a Macro with the name passed in 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](#)

© 1997-2012 Devart. All Rights Reserved.

Executes SQL commands.

Class

[TCustomDASQL](#)

Overload List

Name	Description
Execute	Executes SQL commands.
Execute(Iter: integer)	Executes a SQL statement specified number of times.

© 1997-2012 Devart. All Rights Reserved.

Executes SQL commands.

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.

© 1997-2012 Devart. All Rights Reserved.

Executes a SQL statement specified number of times.

Class

[TCustomDASQL](#)

Syntax

procedure Execute(Iter: integer); **overload; virtual**

Parameters

Iter

Specifies the number of times this statement is executed for the DML array operations.

© 1997-2012 Devart. All Rights Reserved.

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. Executing method is used for nonblocking execution.

© 1997-2012 Devart. All Rights Reserved.

Searches for 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

the TMacro object, if a macro with the specified name has been found. If it has not, returns nil.

Remarks

Call the FindMacro method to find a macro with the specified name in a dataset.

See Also

- [TMacro](#)
- [Macros](#)
- [MacroByName](#)

© 1997-2012 Devart. All Rights Reserved.

Finds a parameter with the specified name.

Class

[TCustomDASQL](#)

Syntax

```
function FindParam(const Value: string): TDAParm;
```

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](#)

© 1997-2012 Devart. All Rights Reserved.

Finds a Macro with the name passed in Name.

Class

[TCustomDASQL](#)

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of the Macro to search for.

Return Value

the Macro, if a match was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Name. If a match was found, MacroByName returns the Macro. 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 FindMacro method.

To assign the value of macro use the [TMacro.Value](#) property.

See Also

- [TMacro](#)
- [Macros](#)

- [FindMacro](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.11.1.7 DBAccess.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.

Inheritance Hierarchy

TObject
TCustomDAUpdateSQL

See Also

- [TOraDataSet.UpdateObject](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

[ModifyObject](#)

[ModifySQL](#)

[RefreshObject](#)

[RefreshSQL](#)

Used to lock the current record.

Provides ability to perform advanced adjustment of modify operations.

Used when updating a record.

Provides ability to perform advanced adjustment of refresh operations.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.11.1.8 DBAccess.TDASConnectionOptions Class

This class allows setting up the behaviour of the TDASConnection class.

For a list of all members of this type, see [TDASConnectionOptions](#) members.

Unit

[DBAccess](#)

Syntax

```
TDASConnectionOptions = class (TPersistent);
```

Inheritance Hierarchy

TObject

TDASConnectionOptions

© 1997-2012 Devart. All Rights Reserved.

[TDASConnectionOptions](#) class overview.

Properties

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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TDASConnectionOptions** class.

For a complete list of the **TDASConnectionOptions** class members, see the [TDASConnectionOptions 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.

See Also

- [TDACConnectionOptions Class](#)
- [TDACConnectionOptions Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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

[TDACConnectionOptions](#)

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.

© 1997-2012 Devart. All Rights Reserved.

Used to open a connection only when needed for performing a server call and closes after performing the operation.

Class

[TDACConnectionOptions](#)

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.

© 1997-2012 Devart. All Rights Reserved.

Used to prevent an application from establishing a connection at the time of startup.

Class

[TDACConnectionOptions](#)

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.9 DBAccess.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);
```

Inheritance Hierarchy

TObject

TDADatasetOptions

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
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.
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.

© 1997-2012 Devart. All Rights Reserved.

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.

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.
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.
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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 default False;
```

Remarks

If True, TCustomDADataset fills the Origin property of the TField objects by appropriate value when opening a dataset.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 [TOraDataSet.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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.10 DBAccess.TDAEncryptionOptions Class

Used to specify the options of the data encryption in a dataset.

For a list of all members of this type, see [TDAEncryptionOptions](#) members.

Unit

[DBAccess](#)

Syntax

```
TDAEncryptionOptions = class(TPersistent);
```

Remarks

Set the properties of Encryption to specify the options of the data encryption in a dataset.

Inheritance Hierarchy

TObject

TDAEncryptionOptions

© 1997-2012 Devart. All Rights Reserved.

[TDAEncryptionOptions](#) 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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TDAEncryptionOptions** class.

For a complete list of the **TDAEncryptionOptions** class members, see the [TDAEncryptionOptions 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

- [TDAEncryptionOptions Class](#)
- [TDAEncryptionOptions Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to specify the encryptor class that will perform the data encryption.

Class

[TDAEncryptionOptions](#)

Syntax

property Encryptor: [TCREncryptor](#);

Remarks

Use the Encryptor property to specify the encryptor class that will perform the data encryption.

© 1997-2012 Devart. All Rights Reserved.

Used to set field names for which encryption will be performed.

Class

[TDAEncryptionOptions](#)

Syntax

property Fields: string;

Remarks

Used to set field names for which encryption will be performed. Field names must be separated by semicolons.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.11 DBAccess.TDAMapRule Class

Class that forms 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

TObject

[TMapRule](#)

TDAMapRule

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

The resultant field length in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldLength: Integer default rlAny;
```

Remarks

Setting the Delphi field length after conversion.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

The resultant field Scale in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldScale: Integer default rlAny;
```

Remarks

Setting the Delphi field Scale after conversion.

© 1997-2012 Devart. All Rights Reserved.

Delphi field type, that the specified DB type or DataSet field will be mapped to.

Class

[TDAMapRule](#)

Syntax

```
property FieldType: TFieldType default ftUnknown;
```

Remarks

Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.12 DBAccess.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

```
TObject
  TMapRules
    TDAMapRules
```

© 1997-2012 Devart. All Rights Reserved.

[TDAMapRules](#) class overview.

Methods

Name	Description
AddDBTypeRule	Overloaded. Adding rules for mapping Database field types to Delphi field types.
AddFieldNameRule	Overloaded. Adding rules for mapping named fields to Delphi field types and setting resultant length and scale for Delphi fields
AddRule	A unified method of adding rules for mapping a DataSet named field or DB field type with the specified length and scale to a field type with the specified length and scale in Delphi.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TDAMapRules** class.

For a complete list of the **TDAMapRules** class members, see the [TDAMapRules Members](#) topic.

Public

Name	Description
AddDBTypeRule	Overloaded. Adding rules for mapping Database field types to Delphi field types.

[AddFieldNameRule](#)

Overloaded. Adding rules for mapping named fields to Delphi field types and setting resultant length and scale for Delphi fields

[AddRule](#)

A unified method of adding rules for mapping a DataSet named field or DB field type with the specified length and scale to a field type with the specified length and scale in Delphi.

See Also

- [TDAMapRules Class](#)
- [TDAMapRules Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types.

Class

[TDAMapRules](#)

Overload List

Name	Description
AddDBTypeRule(DBType: Word; FieldType: TFieldType; IgnoreErrors: boolean)	Adding rules for mapping Database field types to Delphi field types.
AddDBTypeRule(DBType: Word; FieldType: TFieldType; FieldLength: Integer; IgnoreErrors: boolean)	Adding rules for mapping Database field types to Delphi field types with the specified Delphi field length.
AddDBTypeRule(DBType: Word; FieldType: TFieldType; FieldLength: Integer; FieldScale: Integer; IgnoreErrors: boolean)	Adding rules for mapping Database field types to Delphi field types with the specified resultant length and scale of Delphi field.
AddDBTypeRule(DBType: Word; DBLengthMin: Integer; DBLengthMax: Integer; FieldType: TFieldType; IgnoreErrors: boolean)	Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length of DB fields, for which the specified conversion will be applied.
AddDBTypeRule(DBType: Word; DBLengthMin: Integer; DBLengthMax: Integer; FieldType: TFieldType; FieldLength: Integer; IgnoreErrors: boolean)	Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length of DB fields, for which the specified conversion will be applied.
AddDBTypeRule(DBType: Word; DBLengthMin: Integer; DBLengthMax: Integer; DBScaleMin: Integer; DBScaleMax: Integer; FieldType: TFieldType; IgnoreErrors: boolean)	Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length and scale of DB fields, for which the specified conversion will be applied, and with setting the resultant Delphi field length.
AddDBTypeRule(DBType: Word; DBLengthMin: Integer; DBLengthMax: Integer; DBScaleMin: Integer; DBScaleMax: Integer; FieldLength: Integer; FieldScale: Integer; IgnoreErrors: boolean)	Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length and scale of DB fields, for which the specified conversion will be applied, and with setting the resultant Delphi field length and scale.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types.

Class

[TDAMapRules](#)

Syntax

```
procedure AddDBTypeRule(DBType: Word; FieldType: TFieldType;
```

```
IgnoreErrors: boolean = False); overload
```

Parameters

DBType

DB type

FieldType

Delphi field type

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

This method can be applied to all DB fields and Delphi fields, that support conversion between each other.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types with the specified Delphi field length.

Class

[TDAMapRules](#)

Syntax

```
procedure AddDBTypeRule(DBType: Word; FieldType: TFieldType;  
FieldLength: Integer; IgnoreErrors: boolean = False); overload
```

Parameters

DBType

DB type

FieldType

Delphi field type

FieldLength

Delphi field length

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

This method can be used for retrieving Delphi fields ftString, ftWideString, ftBytes, ftVarBytes.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types with the specified resultant length and scale of Delphi field.

Class

[TDAMapRules](#)

Syntax

```
procedure AddDBTypeRule(DBType: Word; FieldType: TFieldType;  
FieldLength: Integer; FieldScale: Integer; IgnoreErrors: boolean  
= False); overload
```

Parameters

DBType

DB type

FieldType

Delphi field type

FieldLength

Delphi field length

FieldScale

Delphi field scale

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

This method can be used for retrieving Delphi fields ftBCD and ftFMTBCD.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length of DB fields, for which the specified conversion will be applied.

Class

[TDAMapRules](#)

Syntax

```
procedure AddDBTypeRule(DBType: Word; DBLengthMin: Integer;  
    DBLengthMax: Integer; FieldType: TFieldType; IgnoreErrors:  
    boolean = False); overload
```

Parameters

DBType

DB type

DBLengthMin

Minimum DB field length

DBLengthMax

Maximum DB field length

FieldType

Delphi field type

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

This method can be applied for all DB text fields.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length of DB fields, for which the specified conversion will be applied.

Class

[TDAMapRules](#)

Syntax

```
procedure AddDBTypeRule(DBType: Word; DBLengthMin: Integer;  
    DBLengthMax: Integer; FieldType: TFieldType; FieldLength:  
    Integer; IgnoreErrors: boolean = False); overload
```

Parameters

DBType

DB type

DBLengthMin

Minimum DB field length

DBLengthMax

Maximum DB field length

FieldType

Delphi field type

FieldLength
Delphi field length

IgnoreErrors
Ignore data conversion errors. Default value is False.

Remarks

This method can be applied to DB text fields for retrieving Delphi fields `ftString`, `ftWideString`, `ftBytes`, `ftVarBytes`.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length and scale of DB fields, for which the specified conversion will be applied, and with setting the resultant Delphi field length.

Class

[TDAMapRules](#)

Syntax

```
procedure AddDBTypeRule(DBType: Word; DBLengthMin: Integer;
  DBLengthMax: Integer; DBScaleMin: Integer; DBScaleMax: Integer;
  FieldType: TFieldType; IgnoreErrors: boolean = False); overload
```

Parameters

DBType
DB type

DBLengthMin
Minimum DB field length

DBLengthMax
Maximum DB field length

DBScaleMin
Minimum DB field scale

DBScaleMax
Maximum DB field scale

FieldType
Delphi field type

IgnoreErrors
Ignore data conversion errors. Default value is False.

Remarks

This method can be applied to those DB fields, for which it is possible to set Scale and Length.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping Database field types to Delphi field types with the specified minimum and maximum length and scale of DB fields, for which the specified conversion will be applied, and with setting the resultant Delphi field length and scale.

Class

[TDAMapRules](#)

Syntax

```
procedure AddDBTypeRule(DBType: Word; DBLengthMin: Integer;
  DBLengthMax: Integer; DBScaleMin: Integer; DBScaleMax: Integer;
  FieldType: TFieldType; FieldLength: Integer; FieldScale:
  Integer; IgnoreErrors: boolean = False); overload
```

Parameters

DBType
DB type

DBLengthMin
Minimum DB field length

DBLengthMax
Maximum DB field length

DBScaleMin
Minimum DB field scale

DBScaleMax
Maximum DB field scale

FieldType
Delphi field type

FieldLength
Delphi field length

FieldScale
Delphi field scale

IgnoreErrors
Ignore data conversion errors. Default value is False.

Remarks

This method can be applied to those DB fields, for which it is possible to set Scale and Length for retrieving Delphi fields ftBCD, ftFMTBCD.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping named fields to Delphi field types and setting resultant length and scale for Delphi fields

Class

[TDAMapRules](#)

Overload List

Name	Description
AddFieldNameRule(FieldName: string; FieldType: TFieldType; IgnoreErrors: Boolean)	Adding rules for mapping named fields to Delphi field types.
AddFieldNameRule(FieldName: string; FieldType: TFieldType; FieldLength: Integer; IgnoreErrors: Boolean)	Adding rules for mapping named fields to Delphi field types and setting the length for Delphi fields.
AddFieldNameRule(FieldName: string; FieldType: TFieldType; FieldLength: Integer; FieldScale: Integer; IgnoreErrors: Boolean)	Adding rules for mapping named fields to Delphi field types and setting the resultant length and scale for Delphi fields

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping named fields to Delphi field types.

Class

[TDAMapRules](#)

Syntax

```
procedure AddFieldNameRule(FieldName: string; FieldType:
TFieldType; IgnoreErrors: Boolean = False); overload
```

Parameters

FieldName
Field name in DataSet

FieldType

Delphi field type

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

This method can be applied to all DataSet field names and Delphi fields. If the DB field type, whose name is specified in the rule, doesn't support conversion to the specified Delphi type, the [Unsupported Data Type Mapping](#) error will occur when opening DataSet.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping named fields to Delphi field types and setting the length for Delphi fields.

Class

[TDAMapRules](#)

Syntax

```
procedure AddFieldNameRule(FieldName: string; FieldType:
TFieldType; FieldLength: Integer; IgnoreErrors: Boolean =
False); overload
```

Parameters

FieldName

Field name in DataSet

FieldType

Delphi field type

FieldLength

Delphi field length

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

This method can be used for retrieving Delphi fields ftString, ftWideString, ftBytes, ftVarBytes.

© 1997-2012 Devart. All Rights Reserved.

Adding rules for mapping named fields to Delphi field types and setting the resultant length and scale for Delphi fields

Class

[TDAMapRules](#)

Syntax

```
procedure AddFieldNameRule(FieldName: string; FieldType:
TFieldType; FieldLength: Integer; FieldScale: Integer;
IgnoreErrors: Boolean = False); overload
```

Parameters

FieldName

Field name in DataSet

FieldType

Delphi field type

FieldLength

Delphi field length

FieldScale

Delphi field scale

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

This method can be used for retrieving Delphi fields ftBCD and ftFMTBCD.

© 1997-2012 Devart. All Rights Reserved.

A unified method of adding rules for mapping a DataSet named field or DB field type with the specified length and scale to a field type with the specified length and scale in Delphi.

Class

[TDAMapRules](#)

Syntax

```
procedure AddRule(FieldName: string; DBType: Word; DBLengthMin: Integer; DBLengthMax: Integer; DBScaleMin: Integer; DBScaleMax: Integer; FieldType: TFieldType; FieldLength: Integer; FieldScale: Integer; IgnoreErrors: boolean = False); overload;  
procedure AddRule(Rule: string); overload;
```

Parameters

FieldName

Field name in DataSet

DBType

DB type

DBLengthMin

Minimum DB field length

DBLengthMax

Maximum DB field length

DBScaleMin

Minimum DB field scale

DBScaleMax

Maximum DB field scale

FieldType

Delphi field type

FieldLength

Delphi field length

FieldScale

Delphi field scale

IgnoreErrors

Ignore data conversion errors. Default value is False.

Remarks

One of two parameters requires to be specified: FieldName or DBType. Also, it is required to specify the FieldType parameter. The other parameters are not required, therefore it is allowed to set the rAny constant for them instead of a specific value. If the rAny constant is set, then the given rule will be applied for all fields independently on their length and scale.

For example, if it is necessary to set the field length in a database to 20 or more, then DBLengthMin should be set to 20, and DBLengthMax - to rAny.

If it is necessary to set scale to 5 or less, then DBScaleMin should be set to rAny, and DBScaleMax - to 5.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.13 DBAccess.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

```

TObject
  TMemDataSet
    TDAMetaData

```

See Also

- [TDAMetaData.MetaDataKind](#)
- [TDAMetaData.Restrictions](#)
- [TDAMetaData.GetMetaDataKinds](#)
- [TDAMetaData.GetRestrictions](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.

[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
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
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.
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.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TDAMetaData** class.

For a complete list of the **TDAMetaData** class members, see the [TDAMetaData Members](#) topic.

Public

Name	Description
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates 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.
Connection	Used to specify a connection object to use to connect to a data store.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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.
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.
MetaDataKind	Used to specify which kind of metainformation to show.
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.
Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.

[Restrictions](#)

Used to provide one or more conditions restricting the list of objects to be described.

[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.

[UnPrepare](#) (inherited from [TMemDataSet](#))

Frees the resources allocated for a previously prepared query on the server and client sides.

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

Used to indicate the update status for the current record when cached updates are enabled.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TDAMetaData Class](#)
- [TDAMetaData Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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 instantiated TCustomDAConnection object.

© 1997-2012 Devart. All Rights Reserved.

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
Databases	show metainformation about existing databases
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

If you provide a value that equals neither of the values described in the table, an error will be raised.

See Also

- [Restrictions](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TDAMetaData** class.

For a complete list of the **TDAMetaData** class members, see the [TDAMetaData Members](#) topic.

Public

Name	Description
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates 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](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetMetaDataKinds](#)

[GetRestrictions](#)

[IndexFieldNames](#) (inherited from [TMemDataSet](#))

[LocalConstraints](#) (inherited from [TMemDataSet](#))

[LocalUpdate](#) (inherited from [TMemDataSet](#))

[Locate](#) (inherited from [TMemDataSet](#))

[LocateEx](#) (inherited from [TMemDataSet](#))

[OnUpdateError](#) (inherited from [TMemDataSet](#))

[OnUpdateRecord](#) (inherited from [TMemDataSet](#))

[Prepare](#) (inherited from [TMemDataSet](#))

[Prepared](#) (inherited from [TMemDataSet](#))

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Makes permanent changes to the database server.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Used to get values acceptable in the MetaDataKind property.

Used to find out which restrictions are applicable to a certain MetaDataKind.

Used to get or set the list of fields on which the recordset is sorted.

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Used to prevent implicit update of rows on database server.

Overloaded. Searches a dataset for a specific record and positions the cursor on it.

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Occurs when an exception is generated while cached updates are applied to a database.

Occurs when a single update component can not handle the updates.

Allocates resources and creates field components for a dataset.

Determines whether a query is prepared for execution or not.

Marks all records in the cache of updates as unapplied.

Cancels changes made to the current record when cached updates are enabled.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Frees the resources allocated for a previously prepared query on the server and client sides.

Used to indicate the update status for the current record when cached updates are enabled.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Used to check the status of the cached updates buffer.

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TDAMetaData Class](#)
 - [TDAMetaData Class Members](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.11.1.14 DBAccess.TDAPParam Class

A class that forms objects to represent the values of the [parameters set](#).
For a list of all members of this type, see [TDAPParam](#) 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.

Inheritance Hierarchy

```
TObject
  TDAParam
```

See Also

- [TCustomDADataset](#)
- [TCustomDASQL](#)
- [TDAParams](#)

© 1997-2012 Devart. All Rights Reserved.

[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**[AssignField](#)[AssignFieldValue](#)[LoadFromFile](#)[LoadFromStream](#)[SetBlobData](#)**Description**

Assigns field name and field value to a param.

Assigns the specified field properties and value to a parameter.

Places the content of a specified file into a TDAParam object.

Places the content from a stream into a TDAParam object.

Overloaded. Writes the data from a specified buffer to BLOB.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TDAParam** class.

For a complete list of the **TDAParam** class members, see the [TDAParam Members](#) topic.

Public**Name**[AsBlob](#)[AsBlobRef](#)[AsFloat](#)[AsInteger](#)[AsLargeInt](#)[AsMemo](#)[AsMemoRef](#)[AsSQLTimeStamp](#)[AsString](#)[AsWideString](#)[IsNull](#)**Description**

Used to set and read the value of the BLOB parameter as string.

Used to set and read the value of the BLOB parameter as a TBlob object.

Used to assign the value for a float field to a parameter.

Used to assign the value for an integer field to the parameter.

Used to assign the value for a LargeInteger field to the parameter.

Used to assign the value for a memo field to the parameter.

Used to set and read the value of the memo parameter as a TBlob object.

Used to specify the value of the parameter when it represents a SQL timestamp field.

Used to assign the string value to the parameter.

Used to assign the Unicode string value to the parameter.

Used to indicate whether the value assigned to a parameter is NULL.

Published**Name**[DataType](#)[ParamType](#)[Size](#)**Description**

Indicates the data type of the parameter.

Used to indicate the type of use for a parameter.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to assign the value for an integer field to the parameter.

Class

[TDAParam](#)

Syntax

```
property AsInteger: integer;
```

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Writes the data from a specified buffer to BLOB.

Class

[TDAParam](#)

Overload List

Name	Description
SetBlobData	Writes the data from a specified buffer to BLOB.
SetBlobData(Buffer: TValueBuffer)	Writes the data from a specified buffer to BLOB.

© 1997-2012 Devart. All Rights Reserved.

Writes the data from a specified buffer to BLOB.

Unit

Syntax

Remarks

Call the SetBlobData method to write data from a specified buffer to BLOB.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.15 DBAccess.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.

Inheritance Hierarchy

TObject
TDAParams

See Also

- [TCustomDADataset.Params](#)
- [TCustomDASQL.Params](#)

- [TDAParam](#)

© 1997-2012 Devart. All Rights Reserved.

[TDAParams](#) class overview.

Properties

Name	Description
Items	Used to iterate through all parameters.

Methods

Name	Description
FindParam	Searches for a parameter with the specified name.
ParamByName	Searches for a parameter with the specified name.

© 1997-2012 Devart. All Rights Reserved.

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 iterate through all parameters.

See Also

- [TDAParams Class](#)
- [TDAParams Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to iterate 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.16 DBAccess.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.

Inheritance Hierarchy

TObject
TDATransaction

© 1997-2012 Devart. All Rights Reserved.

[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
OnError	Used to process errors that occur during executing a transaction.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Events of the **TDATransaction** class.

For a complete list of the **TDATransaction** class members, see the [TDATransaction Members](#) topic.

Public

Name	Description
------	-------------

[OnError](#)

Used to process errors that occur during executing a transaction.

See Also

- [TDATransaction Class](#)
- [TDATransaction Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.11.1.17 DBAccess.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.

Inheritance Hierarchy

```
TObject
  TMacro
```

See Also

- [TMacros](#)

© 1997-2012 Devart. All Rights Reserved.

[TMacro](#) class overview.

Properties

Name	Description
Active	Used to determine if the macro should be expanded.
AsDateTime	Used to set the TDateTime 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.

© 1997-2012 Devart. All Rights Reserved.

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 TDateTime 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](#)

© 1997-2012 Devart. All Rights Reserved.

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;
```

© 1997-2012 Devart. All Rights Reserved.

Used to set the TDateTime value to a macro.

Class

[TMacro](#)

Syntax

property AsDateTime: TDateTime;

Remarks

Use the AsDateTime property to set the TDateTime value to a macro.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to identify a particular macro.

Class

[TMacro](#)

Syntax

property Name: string;

Remarks

Use the Name property to identify a particular macro.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.18 DBAccess.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.

Inheritance Hierarchy

TObject
TMacros

See Also

- [TMacro](#)

© 1997-2012 Devart. All Rights Reserved.

[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	Searches for a TMacro object by its 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

Searches for a TMacro object by its 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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Searches for a TMacro object by its 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 was found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the name passed in Value. If a match is found, FindMacro returns the macro. 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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 rather than a direct reference to the Items property to avoid depending on the order of the entries. To locate a macro by name without raising an exception if the parameter is not found, use the FindMacro method.

© 1997-2012 Devart. All Rights Reserved.

Creates a macros from the passed SQL statement.

Class

[TMacros](#)

Syntax

```
procedure Scan(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.

© 1997-2012 Devart. All Rights Reserved.

17.11.1.19 DBAccess.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) ;
```

Inheritance Hierarchy

TObject
TPoolingOptions

© 1997-2012 Devart. All Rights Reserved.

[TPoolingOptions](#) class overview.

Properties

Name	Description
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 the connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

© 1997-2012 Devart. All Rights Reserved.

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 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 the connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

- [TPoolingOptions Class](#)
- [TPoolingOptions Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to specify the maximum time during which an opened connection can be used by connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property ConnectionLifetime: integer default 0;
```

Remarks

Use the ConnectionLifeTime property to specify the maximum time during which an opened connection can be used by connection pool. Measured in milliseconds. Pool deletes connections with exceeded connection lifetime when [TCustomDAConnection](#) is about to close. If the ConnectionLifetime property is set to 0 (by default), then the lifetime of connection is infinity. ConnectionLifetime concerns only inactive connections in the pool.

© 1997-2012 Devart. All Rights Reserved.

Used to specify the maximum number of connections that can be opened in connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property MaxPoolSize: integer default 100;
```

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.

© 1997-2012 Devart. All Rights Reserved.

Used to specify the minimum number of connections that can be opened in the connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property MinPoolSize: integer default 0;
```

Remarks

Use the MinPoolSize property to specify the minimum number of connections that can be opened in the connection pool.

© 1997-2012 Devart. All Rights Reserved.

Used for a connection to be validated when it is returned from the pool.

Class

[TPoolingOptions](#)

Syntax

```
property Validate: boolean default False;
```

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.2 Types

Types in the **DBAccess** unit.

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.2.1 DBAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.11.2.2 DBAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.11.2.3 DBAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.11.2.4 DBAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.11.2.5 DBAccess.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 .

© 1997-2012 Devart. All Rights Reserved.

17.11.2.6 DBAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.11.2.7 DBAccess.TRefreshOptions Set

Represents the set of [TRefreshOption](#).

Unit

[DBAccess](#)

Syntax

```
TRefreshOptions = set of TRefreshOption;
```

© 1997-2012 Devart. All Rights Reserved.

17.11.2.8 DBAccess.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.

17.11.3 Enumerations

Enumerations in the **DBAccess** unit.

Enumerations

Name	Description
TLabelSet	Sets the language of labels in the connect dialog.
TLockMode	Specifies when the locking of an editing record should be performed.
TRefreshOption	Indicates when the editing record will be refreshed.
TRetryMode	Specifies the application behavior when connection is lost.

© 1997-2012 Devart. All Rights Reserved.

17.11.3.1 DBAccess.TLabelSet Enumeration

Sets the language 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.

© 1997-2012 Devart. All Rights Reserved.

17.11.3.2 DBAccess.TLockMode Enumeration

Specifies when the locking of an editing record should be performed.

Unit

[DBAccess](#)

Syntax

```
TLockMode = (lmNone);
```

Values

Value	Meaning
lmLockDelayed	Locking is performed, then the user posts an edited record. After that the lock is released.
lmLockImmediate	Locking is performed then the user starts editing a record. The lock remains until the user posts or cancels the changes.

ImNone

No locking is performed. This should only be used in single user applications. The default value.

© 1997-2012 Devart. All Rights Reserved.

17.11.3.3 DBAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.11.3.4 DBAccess.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.

© 1997-2012 Devart. All Rights Reserved.

17.11.4 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.
MacroChar	Determinates what character is used for macros.
SQLGeneratorCompatibility	The value of the TCustomDADDataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.

© 1997-2012 Devart. All Rights Reserved.

17.11.4.1 DBAccess.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;
```

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.

© 1997-2012 Devart. All Rights Reserved.

17.11.4.2 DBAccess.ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

Unit

[DBAccess](#)

Syntax

```
ChangeCursor: boolean;
```

© 1997-2012 Devart. All Rights Reserved.

17.11.4.3 DBAccess.MacroChar Variable

Determinates what character is used for macros.

Unit

[DBAccess](#)

Syntax

```
MacroChar: _char;
```

© 1997-2012 Devart. All Rights Reserved.

17.11.4.4 DBAccess.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;
```

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.

© 1997-2012 Devart. All Rights Reserved.

17.12 Devart.Dac.DataAdapter

This unit contains implementation of the DADDataAdapter class.

Classes

Name	Description
DADDataAdapter	DataAdapter serves as a bridge between a System.Data.DataSet and a TDataSet component (data source) for retrieving and saving data.

© 1997-2012 Devart. All Rights Reserved.

17.12.1 Classes

Classes in the **Devart.Dac.DataAdapter** unit.

Classes

Name	Description
DADDataAdapter	DataAdapter serves as a bridge between a System.Data.DataSet and a TDataSet component (data source) for retrieving and saving data.

© 1997-2012 Devart. All Rights Reserved.

17.12.1.1 Devart.Dac.DataAdapter.DADDataAdapter Class

DataAdapter serves as a bridge between a System.Data.DataSet and a TDataSet component (data source) for retrieving and saving data.

For a list of all members of this type, see [DADDataAdapter](#) members.

Unit

[Devart.Dac.DataAdapter](#)

Syntax

```
DADDataAdapter = class (TComponent) ;
```

Remarks

DataAdapter serves as a bridge between a System.Data.DataSet and a TDataSet component (data source) for retrieving and saving data. DataAdapter provides this bridge by mapping [DADDataAdapter.Fill](#), which changes the data in the System.Data.DataSet to match the data in the data source, and [DADDataAdapter.Update](#), which changes the data in the data source to match the data in the System.Data.DataSet.

Inheritance Hierarchy

TObject
DADDataAdapter

© 1997-2012 Devart. All Rights Reserved.

[DADDataAdapter](#) class overview.

Properties

Name	Description
DataSet	Used to specify a TDataSet object which will be used as data source for DADDataAdapter component.

Methods

Name	Description
Fill	Adds or refreshes rows in the System.Data.DataSet to match those in the TDataSet and creates a DataTable.
Update	Performs Insert, Edit, Delete for each inserted, updated, or deleted row in the specified System.Data.DataSet due to the ordering of the rows in the DataTable.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **DADDataAdapter** class.

For a complete list of the **DADDataAdapter** class members, see the [DADDataAdapter Members](#) topic.

Public

Name

[DataSet](#)

Description

Used to specify a TDataSet object which will be used as data source for DADDataAdapter component.

See Also

- [DADDataAdapter Class](#)
- [DADDataAdapter Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to specify a TDataSet object which will be used as data source for DADDataAdapter component.

Class

[DADDataAdapter](#)

Syntax

property DataSet: TDataSet;

Remarks

Specify a TDataSet object which will be used as data source for DADDataAdapter component.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **DADDataAdapter** class.

For a complete list of the **DADDataAdapter** class members, see the [DADDataAdapter Members](#) topic.

Public

Name

[Fill](#)

Description

Adds or refreshes rows in the System.Data.DataSet to match those in the TDataSet and creates a DataTable.

[Update](#)

Performs Insert, Edit, Delete for each inserted, updated, or deleted row in the specified System.Data.DataSet due to the ordering of the rows in the DataTable.

See Also

- [DADDataAdapter Class](#)
- [DADDataAdapter Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Adds or refreshes rows in the System.Data.DataSet to match those in the TDataSet and creates a DataTable.

Class

[DADDataAdapter](#)

Syntax

function Fill(Data: DataSet; tableName: **string**): integer;

Parameters

Data

holds the dataset updates of which are to be commented to the database.

tableName
holds the name of the DataTable.

Return Value

the number of rows successfully inserted into DataSet.

Remarks

Adds or refreshes rows in the System.Data.DataSet to match those in the TDataSet using the DataSet parameter, and creates a DataTable named tableName. Function returns the number of rows successfully inserted into DataSet.

TDataSet object associated with DDataAdapter must be valid, but it does not need to be opened. If TDataSet is closed before Fill is called, it is opened to retrieve data, then closed. If TDataSet is opened before Fill is called, it remains opened.

If an error is encountered while populating the dataset, rows added prior to the occurrence of the error remain in the dataset. The remainder of the operation is aborted.

If TDataSet does not return any rows, fields are created and no rows are added to the DataSet, and no exception is raised.

See Also

- [Update](#)

© 1997-2012 Devart. All Rights Reserved.

Performs Insert, Edit, Delete for each inserted, updated, or deleted row in the specified System.Data.DataSet due to the ordering of the rows in the DataTable.

Class

[DDataAdapter](#)

Syntax

```
function Update(Data: DataSet; tableName: string): integer;
```

Parameters

Data
holds the dataset updates of which are to be commented to the database.

tableName
holds the name of the DataTable.

Return Value

the number of rows successfully updated from the DataSet.

Remarks

Performs Insert, Edit, Delete for each inserted, updated, or deleted row in the specified System.Data.DataSet due to the ordering of the rows in the DataTable. It should be noted that these statements are not performed as a batch process; each row is updated individually. Function returns the number of rows successfully updated from the DataSet.

See Also

- [Fill](#)

© 1997-2012 Devart. All Rights Reserved.

17.13 Devart.Odac.DataAdapter

This unit contains implementation of the OraDataAdapter class.

Classes

Name	Description
OraDataAdapter	A class for using with TCustomOraDataSet components and as data source for retrieving and saving data.

© 1997-2012 Devart. All Rights Reserved.

17.13.1 Classes

Classes in the **Devart.Odac.DataAdapter** unit.

Classes

Name	Description
OraDataAdapter	A class for using with TCustomOraDataSet components and as data source for retrieving and saving data.

© 1997-2012 Devart. All Rights Reserved.

17.13.1.1 Devart.Odac.DataAdapter.OraDataAdapter Class

A class for using with TCustomOraDataSet components and as data source for retrieving and saving data.

For a list of all members of this type, see [OraDataAdapter](#) members.

Unit

[Devart.Odac.DataAdapter](#)

Syntax

```
OraDataAdapter = class (DADDataAdapter) ;
```

Remarks

The OraDataAdapter class is designed for the use with [TOraDataSet](#) components and for the use as data source for retrieving and saving data. OraDataAdapter provides this bridge by mapping [Fill](#), which changes data in System.Data.DataSet to match data in data source, and Update, which changes data in data source to match data in System.Data.DataSet.

Inheritance Hierarchy

TObject
[DADDataAdapter](#)
OraDataAdapter

See Also

- [DADDataAdapter](#)

© 1997-2012 Devart. All Rights Reserved.

[OraDataAdapter](#) class overview.

Properties

Name	Description
DataSet (inherited from DADDataAdapter)	Used to specify a TDataSet object which will be used as data source for DADDataAdapter component.

Methods

Name	Description
Fill (inherited from DADDataAdapter)	Adds or refreshes rows in the System.Data.DataSet to match those in the TDataSet and creates a DataTable.

[Update](#) (inherited from [DADDataAdapter](#))

Performs Insert, Edit, Delete for each inserted, updated, or deleted row in the specified System.Data.DataSet due to the ordering of the rows in the DataTable.

17.14 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.
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
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.

17.14.1 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.
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.

© 1997-2012 Devart. All Rights Reserved.

17.14.1.1 MemData.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.Attributes to access individual attributes. To create TAttribute objects call the TOraType.Describe method. It is called implicitly when ODAC fetches Oracle objects.

Inheritance Hierarchy

```
TObject
  TAttribute
```

See Also

- [TObjectType](#)
- [TOraType](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.14.1.2 MemData.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

TObject
[TSharedObject](#)
TBlob

See Also

- TBlob in Delphi Help
- [TMemDataSet.GetBlob](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
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.

[Release](#) (inherited from [TSharedObject](#))

Decrements the reference count.

[Size](#)

Used to learn the size of the TBlob value in bytes.

See Also

- [TBlob Class](#)
- [TBlob Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.14.1.3 MemData.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, Borland Delphi 8 (for .NET) and Borland Delphi 7. To use BLOB compression under Borland Delphi 6, Borland Delphi 5 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: TCompressedBlob);
TUncompressProc = function(dest: IntPtr; destLen: IntPtr; source: TCompressedBlob);
```

var

```
CompressProc: TCompressProc;
UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

```

TObject
  TSharedObject
    TBlob
      TCompressedBlob
  
```

See Also

- [TBlob](#)
- [TMemDataSet.GetBlob](#)
- [TCustomDADataset.Options](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.

© 1997-2012 Devart. All Rights Reserved.

17.14.1.4 MemData.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

```
TObject
  TSharedObject
    TDBObject
```

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

17.14.1.5 MemData.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

```
TObject
  TSharedObject
    TObjectType
```

See Also

- [TOraType](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
AttributeByName	Retrieves attribute information for an attribute when only the attribute's name is known.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from TSharedObject)	Decrements the reference count.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
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.
Release (inherited from TSharedObject)	Decrements the reference count.
Size	Used to learn the size of an object instance.

See Also

- [TObjectType Class](#)
- [TObjectType Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

See Also

- [MemData](#)
 - [OraClasses](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
AttributeByName	Retrieves attribute information for an attribute when only the attribute's name is known.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

- [TObjectType Class](#)
- [TObjectType Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Retrieves attribute information for an attribute when only the attribute's name is known.

Class

[TObjectType](#)

Syntax

```
function AttributeByName (Name: string) : TAttribute;
```

Parameters

Name

Holds the name of an existing attribute.

Return Value

a TAttribute object for the specified attribute. Otherwise an exception is raised.

Remarks

Call the AttributeByName method to retrieve attribute information for an attribute when only the

attribute's name is known. Name is the name of an existing Attribute. AttributeByName returns a TAttribute object for the specified attribute. If the attribute can not be found, an exception is raised.

See Also

- [TAttribute](#)
- [FindAttribute](#)
- [Attributes](#)

© 1997-2012 Devart. All Rights Reserved.

Indicates whether a specified Attribute component is referenced in the TAttributes object.

Class

[TObjectType](#)

Syntax

```
function FindAttribute(Name: string): TAttribute;
```

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](#)
- [AttributeByName](#)
- [Attributes](#)

© 1997-2012 Devart. All Rights Reserved.

17.14.1.6 MemData.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.

Inheritance Hierarchy

TObject

TSharedObject

See Also

- [TBlob](#)
 - [TObjectType](#)
-

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.14.2 Types

Types in the **MemData** unit.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption .
TUpdateRecKinds	Represents the set of TUpdateRecKind.

© 1997-2012 Devart. All Rights Reserved.

17.14.2.1 MemData.TLocateExOptions Set

Represents the set of [TLocateExOption](#).

Unit

[MemData](#)

Syntax

```
TLocateExOptions = set of TLocateExOption;
```

© 1997-2012 Devart. All Rights Reserved.

17.14.2.2 MemData.TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

[MemData](#)

Syntax

```
TUpdateRecKinds = set of TUpdateRecKind;
```

© 1997-2012 Devart. All Rights Reserved.

17.14.3 Enumerations

Enumerations in the **MemData** unit.

Enumerations

Name	Description
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.

© 1997-2012 Devart. All Rights Reserved.

17.14.3.1 MemData.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.

© 1997-2012 Devart. All Rights Reserved.

17.14.3.2 MemData.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. Not supported for Delphi 5 and C++Builder 5.

© 1997-2012 Devart. All Rights Reserved.

17.14.3.3 MemData.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.

© 1997-2012 Devart. All Rights Reserved.

17.14.3.4 MemData.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.

© 1997-2012 Devart. All Rights Reserved.

17.14.3.5 MemData.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.

© 1997-2012 Devart. All Rights Reserved.

17.15 MemDS

This unit contains implementation of the TMemDataSet class.

Classes

Name	Description
<u>TMemDataSet</u>	A base class for working with data and manipulating data in memory.

Variables

Name	Description
<u>DoNotRaiseExcetionOnUaFail</u>	An exception will be raised if the value of the UpdateAction parameter is uaFail.
<u>SendDataSetChangeEventAfterOpen</u>	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.

17.15.1 Classes

Classes in the **MemDS** unit.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

© 1997-2012 Devart. All Rights Reserved.

17.15.1.1 MemDS.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.

Inheritance Hierarchy

```
TObject
  TMemDataSet
```

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
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
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.

[CancelUpdates](#)

Clears all pending cached updates from cache and restores dataset in its prior state.

[CommitUpdates](#)[DeferredPost](#)

Clears the cached updates buffer. Makes permanent changes to the database server.

[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.

[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.

© 1997-2012 Devart. All Rights Reserved.

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.

[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.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

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 [TCustomDADataset.Options](#) is set to True.

See Also

- [UpdatesPending](#)
- [TMemDataSet.ApplyUpdates](#)
- [RestoreUpdates](#)
- [CommitUpdates](#)
- [CancelUpdates](#)
- [UpdateStatus](#)
- [TCustomDADataset.Options](#)

© 1997-2012 Devart. All Rights Reserved.

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. Ordering of column names is significant. Separate names with semicolon. 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 direction for the field. If one of these keywords is not used, the default sort direction for the field is ascending.

Use CIS, CS or BIN keywords to specify a 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 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.

Ordering is processed locally.

Note: You cannot process ordering by BLOB fields.

Example

The following procedure illustrates how to set IndexFieldNames in response to a button click:

```
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

© 1997-2012 Devart. All Rights Reserved.

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 LocalConstraints 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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TMemDataSet** class.

For a complete list of the **TMemDataSet** class members, see the [TMemDataSet Members](#) topic.

Public

Name	Description
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.
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.
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.
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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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;
end;
```

See Also

- [TMemDataSet.CachedUpdates](#)
 - [TMemDataSet.CancelUpdates](#)
 - [TMemDataSet.CommitUpdates](#)
 - [TMemDataSet.UpdateStatus](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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 whether 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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

17.15.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.15.2.1 MemDS.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.

© 1997-2012 Devart. All Rights Reserved.

17.15.2.2 MemDS.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.

© 1997-2012 Devart. All Rights Reserved.

17.16 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.

17.16.1 Classes

Classes in the **OdacVcl** unit.

Classes

Name	Description
TConnectDialog	A component providing a dialog box for a user to supply login information.

© 1997-2012 Devart. All Rights Reserved.

17.16.1.1 OdacVcl.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

```
TObject
  TCustomConnectDialog
    TConnectDialog
```

See Also

- [TCustomDAConnection.ConnectDialog](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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.
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.
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**[ReadAliases](#)**Description**

Used to specify where the TConnectDialog object should acquire the names of database instances.

See Also

- [TConnectDialog Class](#)
- [TConnectDialog Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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[OdacVcl](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

17.17 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.
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.
TPISqlTraceMode	Specifies the level of PL/SQL trace.
TSqlTraceMode	Specifies the level of SQL trace statistics level.

Enumerations

Name	Description
TCheckMode	Specifies the action to take when another user makes modifications to a record.
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.

Routines

Name	Description
AddWhere	Appends condition to WHERE clause.
DefaultSession	Call this function to get pointer to default session object.
DeleteWhere	Removes WHERE clause from a SELECT statement.
GetOrderBy	Returns fields from ORDER BY.
SessionByName	Returns pointer to session object by its name.

[SetFieldList](#)
[SetGroupBy](#)
[SetOrderBy](#)
[SetSubscriptionPort](#)

Sets list of selecting fields.
Sets grouping fields.
Sets ordering fields.
Sets change notification subscription port. Before setting the subscription port all opened sessions close and oci.dll unloads.
Sets list of tables.
Call this function to replace WHERE clause of a SELECT statement to Condition. You should pass SELECT statement by SQL.If condition is blank SetWhere removes WHERE clause.

[SetTableList](#)
[SetWhere](#)

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.

Constants

Name	Description
OdacVersion	Read this constant to get current version number for ODAC.

17.17.1 Classes

Classes in the **Ora** unit.

Classes

Name	Description
<u>TBFileField</u>	A class representing BFile field in dataset.
<u>TCursorField</u>	A class representing REF CURSOR field in dataset.
<u>TCustomOraQuery</u>	A base class defining functionality for descendent classes which access database using SQL statements.
<u>TOraChangeNotification</u>	A component for keeping information in local dataset up-to-date through receiving notifications.
<u>TOraDataSet</u>	A class defining the Oracle functionality for a dataset.
<u>TOraDataSetField</u>	A class providing access to Oracle nested datasets.
<u>TOraDataSetOptions</u>	This class allows setting up the behaviour of the TOraDataSet class.
<u>TOraDataSetOptionsDS</u>	This class allows setting up the behaviour of the TOraDaatSet class (this property is obsolete).
<u>TOraDataSource</u>	TOraDataSource provides an interface between an ODAC dataset components and data-aware controls on a form.
<u>TOraEncryptor</u>	The class that performs encrypting and decrypting of data.
<u>TOraIntervalField</u>	A class providing access to the Oracle interval fields.
<u>TOraMetaData</u>	A component for obtaining metainformation about database objects from the server.
<u>TOraNestedTable</u>	A component for controlling nested table data.
<u>TOraNumberField</u>	A class providing access to the Oracle number fields.
<u>TOraParam</u>	A class that is used to set the values of individual parameters passed with queries or stored procedures.
<u>TOraParams</u>	Used to control TOraParam objects.
<u>TOraPoolingOptions</u>	This class allows setting up the behaviour of the connecton pool.
<u>TOraQuery</u>	A component for executing queries and operating record sets. It also provides flexible way to update data.
<u>TOraReferenceField</u>	A class representing an Oracle REF field in a dataset.
<u>TOraSession</u>	A component for maintaining connection to an Oracle database.
<u>TOraSessionOptions</u>	This class allows setting up the behaviour of the TOraSession class.
<u>TOraSQL</u>	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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.1 Ora.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.

Inheritance Hierarchy

```
TObject
  TBFileField
```

See Also

- [TOraFile](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Returns a TOraFile object.

Class

[TBFileField](#)

Syntax

property AsFile: [TOraFile](#);

Remarks

Returns a TOraFile object. Later you can open TOraDataSet once.

See Also

- [TOraFile](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 + '    (NotExist)';
```

© 1997-2012 Devart. All Rights Reserved.

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;
```

© 1997-2012 Devart. All Rights Reserved.

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;
```

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Reloads BFile.

Class

[TBFileField](#)

Syntax

```
procedure Refresh;
```

Remarks

Call the Refresh procedure to reload BFile.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.2 Ora.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

TObject
TDACursorField
TCursorField

See Also

- [TOraCursor](#)
- [TOraDataSet.Cursor](#)

© 1997-2012 Devart. All Rights Reserved.

[TCursorField](#) class overview.

Properties

Name	Description
AsCursor	Returns a TOraCursor object you can assign to the Cursor property of TOraDataSet .

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.3 Ora.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 [TCustomDADDataSet.BeforeUpdateExecute](#) event to assign the value to additional parameters and the [TCustomDADDataSet.AfterUpdateExecute](#) event for reading them.

TCustomOraQuery is read-only when none of the SQLInsert, SQLDelete, SQLUpdate properties are defined.

Inheritance Hierarchy

TObject

[TMemDataSet](#)

[TCustomDADDataSet](#)

[TOraDataSet](#)

TCustomOraQuery

See Also

- [TOraDataSet](#)
- [TOraQuery](#)
- [TSmartQuery](#)
- [TOraStoredProc](#)
- [TOraTable](#)

© 1997-2012 Devart. All Rights Reserved.

[TCustomOraQuery](#) class overview.

Properties

Name	Description
BaseSQL (inherited from TCustomDADDataSet)	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.

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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of 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 TCustomDADataset)	Used to specify the options of the data encryption 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.
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.

<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>NonBlocking</u> (inherited from <u>TOraDataSet</u>)	Used to execute a SQL statement and fetch rows by a separate thread.
<u>Options</u> (inherited from <u>TOraDataSet</u>)	Used to specify the behaviour of TOrDataSetObject.
<u>OptionsDS</u> (inherited from <u>TOraDataSet</u>)	Used to specify the behaviour of TOrDataSetObject.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TOraDataSet</u>)	Contains the parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify when to refresh an editing record.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>ReturnParams</u> (inherited from <u>TOraDataSet</u>)	Used to return a new fields value to dataset after insert or update.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>RowsProcessed</u> (inherited from <u>TOraDataSet</u>)	Returns the number of rows processed by a query.
<u>SequenceMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>Session</u> (inherited from <u>TOraDataSet</u>)	Used to specify the session in which dataset will be executed.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> 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 point to 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.
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.
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.
ErrorOffset (inherited from TOraDataSet)	Returns the parse error offset.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
Fetched (inherited from TOraDataSet)	Indicates if TCustomDADataset has already fetched all rows.
Fetching (inherited from TCustomDADataset)	Used to learn whether TCustomDADataset is still fetching rows.

[FetchingAll](#) (inherited from [TCustomDADataset](#))

[FindKey](#) (inherited from [TCustomDADataset](#))

[FindMacro](#) (inherited from [TCustomDADataset](#))

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TOraDataSet](#))

[GetArray](#) (inherited from [TOraDataSet](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetErrorPos](#) (inherited from [TOraDataSet](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetFile](#) (inherited from [TOraDataSet](#))

[GetInterval](#) (inherited from [TOraDataSet](#))

[GetKeyList](#) (inherited from [TOraDataSet](#))

[GetLob](#) (inherited from [TOraDataSet](#))

[GetLobLocator](#) (inherited from [TOraDataSet](#))

[GetObject](#) (inherited from [TOraDataSet](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GetRef](#) (inherited from [TOraDataSet](#))

[GetTable](#) (inherited from [TOraDataSet](#))

[GetTimeStamp](#) (inherited from [TOraDataSet](#))

[GotoCurrent](#) (inherited from [TCustomDADataset](#))

[Locate](#) (inherited from [TMemDataSet](#))

Used to learn whether TCustomDADataset is fetching all rows to the end.

Searches for a record which contains specified field values.

Indicates whether a specified macro exists in a dataset.

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.

Determines whether a parameter with the specified name exists in a dataset.

Retrieves a TORAArray object for a field when only its name is known.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Returns internal field types defined in the MemData and accompanying modules.

Returns a row and column of parse error for a SQL statement.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves a TORAFile object for a field with known name.

Retrieves a TORAInterval object for a field with known name.

Returns the list of table primary key fields.

Retrieves a TORALob object for a field with known name.

Retrieves a TORALob object for a field with known name.

Retrieves a TORAObject object for a field with known name.

Retrieves an ORDER BY clause from a SQL statement.

Retrieves a TORARef object for a field with known name.

Retrieve a TORANestTable object for a field with known name.

Retrieves a TORATimeStamp object for a field with known name.

Sets the current record in this dataset similar to the current record in another dataset.

Overloaded. Searches a dataset for a specific record and positions the cursor on it.

[LocateEx](#) (inherited from [TMemDataSet](#))

[Lock](#) (inherited from [TCustomDADataset](#))

[MacroByName](#) (inherited from [TCustomDADataset](#))

[ParamByName](#) (inherited from [TOraDataSet](#))

[Prepare](#) (inherited from [TCustomDADataset](#))

[RefreshRecord](#) (inherited from [TCustomDADataset](#))

[RestoreSQL](#) (inherited from [TCustomDADataset](#))

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[Resync](#) (inherited from [TCustomDADataset](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Locks the current record.

Finds a Macro with the name passed in Name.

Sets or uses parameter information for a specific parameter based on its name.

Allocates, opens, and parses cursor for a query.

Actualizes field values for the current record.

Restores the SQL property modified by AddWhere and SetOrderBy.

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Cancels changes made to the current record when cached updates are enabled.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Builds an ORDER BY clause of a SELECT statement.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.4 Ora.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.

Inheritance Hierarchy

```
TObject
  TOraChangeNotification
```

See Also

- Change Notification demo
- [TOraChangeNotification Component](#)

© 1997-2012 Devart. All Rights Reserved.

[TOraChangeNotification](#) class overview.

Properties

Name	Description
Active	Description is not available at the moment.
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.
Port	Used to subscribe for the database change notification.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraChangeNotification** class.

For a complete list of the **TOraChangeNotification** class members, see the [TOraChangeNotification Members](#) topic.

Public

Name	Description
Active	Description is not available at the moment.
Persistent	Used to store notifications in a database persistently.
Port	Used to subscribe for the database change notification.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Class

[TOraChangeNotification](#)

Syntax

property Active: boolean;

Remarks

Indicates whether there is an active dataset connected with the TOraChangeNotification component.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to subscribe for the database change notification.

Class

[TOraChangeNotification](#)

Syntax

property Port: integer;

Remarks

Set the Port property for the database change notification subscription.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.5 Ora.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

```

TObject
  TMemDataSet
    TCustomDADataset
      TOraDataSet
  
```

See Also

- [TOraQuery](#)
- [TSmartQuery](#)
- [TOraStoredProc](#)
- [TOraTable](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of 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 (inherited from TCustomDADataset)	Used to specify the options of the data encryption 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.
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.
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.
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.
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 point to 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.
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.
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.
ErrorOffset	Returns the parse error offset.

<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u>	Indicates if TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether a specified macro exists in a dataset.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u>	Determines whether a parameter with the specified name exists in a dataset.
<u>GetArray</u>	Retrieves a TOrArray object for a field when only its name is known.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetErrorPos</u>	Returns a row and column of parse error for a SQL statement.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetFile</u>	Retrieves a TOrFile object for a field with known name.
<u>GetInterval</u>	Retrieves a TOrInterval object for a field with known name.
<u>GetKeyList</u>	Returns the list of table primary key fields.
<u>GetLob</u>	Retrieves a TOrLob object for a field with known name.
<u>GetLobLocator</u>	Retrieves a TOrLob object for a field with known name.
<u>GetObject</u>	Retrieves a TOrObject object for a field with known name.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetRef</u>	Retrieves a TOrRef object for a field with known name.
<u>GetTable</u>	Retrieve a TOrNestTable object for a field with known name.

[GetTimeStamp](#)

[GotoCurrent](#) (inherited from [TCustomDADataset](#))

[Locate](#) (inherited from [TMemDataSet](#))

[LocateEx](#) (inherited from [TMemDataSet](#))

[Lock](#) (inherited from [TCustomDADataset](#))

[MacroByName](#) (inherited from [TCustomDADataset](#))

[ParamByName](#)

[Prepare](#) (inherited from [TCustomDADataset](#))

[RefreshRecord](#) (inherited from [TCustomDADataset](#))

[RestoreSQL](#) (inherited from [TCustomDADataset](#))

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[Resync](#) (inherited from [TCustomDADataset](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Retrieves a ToraTimeStamp object for a field with known name.

Sets the current record in this dataset similar to the current record in another dataset.

Overloaded. Searches a dataset for a specific record and positions the cursor on it.

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Locks the current record.

Finds a Macro with the name passed in Name.

Sets or uses parameter information for a specific parameter based on its name.

Allocates, opens, and parses cursor for a query.

Actualizes field values for the current record.

Restores the SQL property modified by AddWhere and SetOrderBy.

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Cancels changes made to the current record when cached updates are enabled.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Builds an ORDER BY clause of a SELECT statement.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
ChangeNotification	Used to receive database change notification messages to refresh dataset when required.
CheckMode	Used to define the check mode before editing a record.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.

Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
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.
Cursor	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.
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.
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 (inherited from TCustomDADataset)	Used to specify the options of the data encryption in a dataset.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
FetchAll	Used to request all records of the query from database server when a dataset is being opened.
Fetched (inherited from TCustomDADataset)	Used to learn whether TCustomDADataset has already 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.
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.
FindKey (inherited from TCustomDADataset)	Searches for a record which contains specified field values.
FindMacro (inherited from TCustomDADataset)	Indicates whether a specified macro exists in a dataset.

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TCustomDADataset](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GotoCurrent](#) (inherited from [TCustomDADataset](#))

[IndexFieldNames](#) (inherited from [TMemDataSet](#))

[IsPLSQL](#)

[IsQuery](#)

[KeyFields](#)

[KeySequence](#)

[LocalConstraints](#) (inherited from [TMemDataSet](#))

[LocalUpdate](#) (inherited from [TMemDataSet](#))

[Locate](#) (inherited from [TMemDataSet](#))

[LocateEx](#) (inherited from [TMemDataSet](#))

[Lock](#) (inherited from [TCustomDADataset](#))

[LockMode](#)

[MacroByName](#) (inherited from [TCustomDADataset](#))

[MacroCount](#) (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.

Determines if a parameter with the specified name exists in a dataset. Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Returns internal field types defined in the MemData and accompanying modules.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves an ORDER BY clause from a SQL statement.

Sets the current record in this dataset similar to the current record in another dataset.

Used to get or set the list of fields on which the recordset is sorted.

Indicates whether a SQL statement is a PL/SQL block.

Indicates whether SQL statement returns rows or not.

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.

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.

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Used to prevent implicit update of rows on database server.

Overloaded. Searches a dataset for a specific record and positions the cursor on it.

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Locks the current record.

Used to define when to perform the locking of an editing record.

Finds a Macro with the name passed in Name.

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.
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.
Options	Used to specify the behaviour of ToraDataSetObject.
OptionsDS	Used to specify the behaviour of ToraDataSetObject.
ParamByName (inherited from TCustomDADataset)	Sets or uses parameter information for a specific parameter based on its name.
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.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.
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.
Resync (inherited from TCustomDADataset)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
ReturnParams	Used to return a new fields value to dataset after insert or update.

[RevertRecord](#) (inherited from [TMemDataSet](#))

[RowsAffected](#) (inherited from [TCustomDADataset](#))

[RowsProcessed](#)

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SequenceMode](#)

[Session](#)

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQL](#) (inherited from [TCustomDADataset](#))

[SQLDelete](#) (inherited from [TCustomDADataset](#))

[SQLInsert](#) (inherited from [TCustomDADataset](#))

[SQLLock](#) (inherited from [TCustomDADataset](#))

[SQLRefresh](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[SQLType](#)

[SQLUpdate](#) (inherited from [TCustomDADataset](#))

[StrictUpdate](#)

[UniDirectional](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

Cancels changes made to the current record when cached updates are enabled.

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Returns the number of rows processed by a query.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Used to specify the methods used internally to generate a sequenced field.

Used to specify the session in which dataset will be executed.

Builds an ORDER BY clause of a SELECT statement.

Used to provide a SQL statement that a query component executes when its Open method is called.

Used to specify a SQL statement that will be used when applying a deletion to a record.

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Used to specify a SQL statement that will be used to perform a record lock.

Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataset.RefreshRecord](#) procedure.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Used to get the typecode of the SQL statement being processed by Oracle database server.

Used to specify a SQL statement that will be used when applying an update to a dataset.

Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.

Used if an application does not need bidirectional access to records in the result set.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

[UpdateObject](#)

Used to point to 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.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TOraDataSet Class](#)
- [TOraDataSet Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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 = ImLockImmediate, [CheckMode](#) = cmException. To set optimistic locking use LockMode = ImLockDelayed, [CheckMode](#) = cmException.

The default value is ImNone.

See Also

- [TCustomDADataset.Lock](#)
- [TCustomOraQuery.SQLLock](#)
- [CheckMode](#)

© 1997-2012 Devart. All Rights Reserved.

Used to execute a SQL statement and fetch rows by a separate thread.

Class

[TOraDataSet](#)

Syntax

property NonBlocking: boolean **default** False;

Remarks

Set the NonBlocking property to True to execute SQL statement and fetch rows by a separate thread.

© 1997-2012 Devart. All Rights Reserved.

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.
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.

[PrefetchRows](#)

Used to get or set the number of rows that OCI prefetches when executing a query.

[PrepareUpdateSQL](#)

Used to automatically prepare update queries before execution.

[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 associated 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](#)

©

1997-2012 Devart. All Rights Reserved.

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.
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](#)

©

1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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

- [TCustomDADDataSet.RowsAffected](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 [TCustomDADDataSet.Options](#) instead.

© 1997-2012 Devart. All Rights Reserved.

Used to point to 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 points to an update object component which provides SQL statements that perform updates of the read-only datasets when cached updates are enabled.

© 1997-2012 Devart. All Rights Reserved.

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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates 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.
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.
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.

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.
Encryption (inherited from TCustomDADataset)	Used to specify the options of the data encryption in a dataset.
ErrorOffset	Returns the parse error offset.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
Fetched	Indicates if TCustomDADataset has already 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.
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.
FindKey (inherited from TCustomDADataset)	Searches for a record which contains specified field values.
FindMacro (inherited from TCustomDADataset)	Indicates whether a specified macro exists in a dataset.
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.
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 TCustomDADDataSet)	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 TCustomDADDataSet)	Sets the current record in this dataset similar to the current record in another dataset.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADDataSet)	Used to check whether SQL statement returns rows.
KeyFields (inherited from TCustomDADDataSet)	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.
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 TCustomDADDataSet)	Locks the current record.
MacroByName (inherited from TCustomDADDataSet)	Finds a Macro with the name passed in Name.
MacroCount (inherited from TCustomDADDataSet)	Used to get the number of macros associated with the Macros property.
Macros (inherited from TCustomDADDataSet)	Makes it possible to change SQL queries easily.
MasterFields (inherited from TCustomDADDataSet)	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.

<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>OnUpdateError</u> (inherited from <u>TMemDataSet</u>)	Occurs when an exception is generated while cached updates are applied to a database.
<u>OnUpdateRecord</u> (inherited from <u>TMemDataSet</u>)	Occurs when a single update component can not handle the updates.
<u>Options</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the behaviour of TCustomDADataset object.
<u>ParamByName</u>	Sets or uses parameter information for a specific parameter based on its name.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomDADataset</u>)	Used to view and set parameter names, values, and data types dynamically.
<u>Prepare</u> (inherited from <u>TCustomDADataset</u>)	Allocates, opens, and parses cursor for a query.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>Resync</u> (inherited from <u>TCustomDADataset</u>)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Builds an ORDER BY clause of a SELECT statement.

[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.

[SQLRefresh](#) (inherited from [TCustomDADataset](#))

Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataset.RefreshRecord](#) procedure.

[SQLSaved](#) (inherited from [TCustomDADataset](#))

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

[SQLUpdate](#) (inherited from [TCustomDADataset](#))

Used to specify a SQL statement that will be used when applying an update to a dataset.

[UniDirectional](#) (inherited from [TCustomDADataset](#))

Used if an application does not need bidirectional access to records in the result set.

[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.

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

Used to indicate the update status for the current record when cached updates are enabled.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TOraDataSet Class](#)
- [TOraDataSet Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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 number 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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Indicates if TCustomDADataset has already fetched all rows.

Class

[TOraDataSet](#)

Syntax

```
function Fetched: boolean; override;
```

Remarks

Check the Fetched property to learn whether TCustomDADataset has already fetched all rows.

See Also

- [TCustomDADataset.Fetching](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
 - [TCustomDADDataSet.GetDataType](#)
 - [TOraParam.AsBLOBLocator](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
 - [TCustomDADDataSet.GetDataType](#)
 - [TOraParam.AsBLOBLocator](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.6 Ora.TOrDataSetField Class

A class providing access to Oracle nested datasets.

For a list of all members of this type, see [TOraDataSetField](#) members.

Unit

[Ora](#)

Syntax

```
TOrDataSetField = class (TDataSetField);
```

Remarks

TOrDataSetField provides access to Oracle nested datasets.

Inheritance Hierarchy

TObject

TOrDataSetField

© 1997-2012 Devart. All Rights Reserved.

[TOraDataSetField](#) class overview.

Properties

Name	Description
Modified	Indicates whether tafield was modified.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Indicates whether tafield was modified.

Class

[ToraDataSetField](#)

Syntax

```
property Modified: boolean;
```

Remarks

The Modified proptry indicates whether a field was modified. The property is writable.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.7 Ora.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 associated 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

TObject

[TDADatasetOptions](#)

[ToraDataSetOptionsDS](#)

ToraDataSetOptions

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
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.
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.
NumberRange (inherited from TDADatasetOptions)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
PrefetchRows	Used to get or set the number of rows that OCI prefetches when executing a query.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
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 associated 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.

© 1997-2012 Devart. All Rights Reserved.

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.
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.
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.
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.
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.
KeepPrepared (inherited from TOraDataSetOptionsDS)	Used to keep TOraDataSet prepared after closing.
PrefetchRows	Used to get or set the number of rows that OCI prefetches when executing a query.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
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 associated 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](#)

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to get or set the number of rows that OCI prefetches when executing a query.

Class

[TOraDataSetOptions](#)

Syntax

```
property PrefetchRows: integer default 1;
```

Remarks

Use the PrefetchRows property to get or set the number of rows that OCI prefetches when executing a query. Setting a value greater than 0 for this option minimizes server round trip count, what increases performance of the application. By default the value of this options is 1 - OCI always prefetches one extra row.

Note: Some queries (for example, query `SELECT * FROM DUAL CONNECT BY LEVEL <= 5` returns 1 row when prefetch is enabled, and 5 rows when it is disabled) can return invalid rows count when prefetch is enabled.

© 1997-2012 Devart. All Rights Reserved.

Used to automatically prepare update queries before execution.

Class

[TOraDataSetOptions](#)

Syntax

property PrepareUpdateSQL: boolean;

Remarks

If True, update queries are automatically prepared before executing.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to get a value indicating whether Oracle resources associated 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.StatementCache](#) to false if you don't want the statements to be cached.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.8 Ora.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

```
TObject
  TDADatasetOptions
    TOraDataSetOptionsDS
```

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
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.
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.

© 1997-2012 Devart. All Rights Reserved.

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.
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.
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.
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.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.9 Ora.TOrDataSource Class

TOrDataSource 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

TOrDataSource = **class** ([TCRDataSource](#)) ;

Remarks

TOrDataSource provides an interface between an ODAC dataset components and data-aware controls on a form.

TOrDataSource 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

TObject

[TCRDataSource](#)

TOrDataSource

© 1997-2012 Devart. All Rights Reserved.

[TOraDataSource](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.10 Ora.TOrEncryptor 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

TOrEncryptor = **class** ([TCREncryptor](#)) ;

Inheritance Hierarchy

TObject

[TCREncryptor](#)

TOrEncryptor

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.11 Ora.TOralIntervalField 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

```
TOralIntervalField = class (TField);
```

Remarks

TOralIntervalField provides access to the Oracle interval fields. Unlike other TField descendants the TOralIntervalField.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.

Inheritance Hierarchy

TObject
 TOralIntervalField

See Also

- [TOraInterval](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **ToraIntervalField** class.

For a complete list of the **ToraIntervalField** class members, see the [ToraIntervalField Members](#) topic.

Public**Name**[AsInterval](#)**Description**

Used to provide access to a ToraInterval object.

Published**Name**[FracPrecision](#)**Description**

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.12 Ora.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

```

TObject
  TMemDataSet
    TDAMetaData
      TOraMetaData

```

See Also

- [TCustomDADataset.Debug](#)
- [TCustomDASQL.Debug](#)
- [DBMonitor](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
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
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
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.
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.
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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.13 Ora.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

```
TObject
  TMemDataSet
    TOraNestedTable
```

See Also

- [TOraNestTable](#)
- [TOraRef](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
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
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
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.
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.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraNestedTable** class.

For a complete list of the **TOraNestedTable** class members, see the [TOraNestedTable Members](#) topic.

Public

Name	Description
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates 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.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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.
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.

[OnUpdateError](#) (inherited from [TMemDataSet](#))

[OnUpdateRecord](#) (inherited from [TMemDataSet](#))

[Prepare](#) (inherited from [TMemDataSet](#))

[Prepared](#) (inherited from [TMemDataSet](#))

[Ref](#)

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[Table](#)

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Occurs when an exception is generated while cached updates are applied to a database.

Occurs when a single update component can not handle the updates.

Allocates resources and creates field components for a dataset.

Determines whether a query is prepared for execution or not.

Used to assign reference data to TOraNestedTable.

Marks all records in the cache of updates as unapplied.

Cancels changes made to the current record when cached updates are enabled.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Used to assign nested table data to TOraNestedTable.

Frees the resources allocated for a previously prepared query on the server and client sides.

Used to indicate the update status for the current record when cached updates are enabled.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Used to check the status of the cached updates buffer.

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TOraNestedTable Class](#)
- [TOraNestedTable Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.14 Ora.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.

Inheritance Hierarchy

```
TObject  
  TOranumberField
```

See Also

- [TOraNumber](#)

© 1997-2012 Devart. All Rights Reserved.

[TOraNumberField](#) class overview.

Properties

Name	Description
AsNumber	Used to provide access to a TOraNumber object.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.15 Ora.TOruParam 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

TObject
[TDAParam](#)
TOriParam

See Also

- [TOraDataSet](#)
- [TOraSQL](#)
- [TOraParams](#)

© 1997-2012 Devart. All Rights Reserved.

[TOriParam](#) 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.

<u>AsOraBlob</u>	Used to specify the value of a parameter when it represents the value of the BLOB type.
<u>AsOraClob</u>	Used to specify the value of the parameter when it represents the value of CLOB type.
<u>AsRef</u>	Used to specify the value of the parameter when it represents the value of the Oracle reference type.
<u>AsSQLTimeStamp</u> (inherited from <u>TDAParam</u>)	Used to specify the value of the parameter when it represents a SQL timestamp field.
<u>AsString</u> (inherited from <u>TDAParam</u>)	Used to assign the string value to the parameter.
<u>AsTable</u>	Used to specify the value of the parameter when it represents the value of the Oracle nested table type.
<u>AsTimeStamp</u>	Used to specify parameter value when it represents Oracle 9 timestamp type.
<u>AsWideString</u> (inherited from <u>TDAParam</u>)	Used to assign the Unicode string value to the parameter.
<u>AsXML</u>	Used to specify the value of the parameter when it represents the value of Oracle SYS.XMLTYPE.
<u>DataType</u> (inherited from <u>TDAParam</u>)	Indicates the data type of the parameter.
<u>IsNull</u> (inherited from <u>TDAParam</u>)	Used to indicate whether the value assigned to a parameter is NULL.
<u>ItemAsDateTime</u>	Used to access a PL/SQL table item as TDateTime by Index.
<u>ItemAsFloat</u>	Used to access a PL/SQL table item as Double by Index.
<u>ItemAsInteger</u>	Used to access a PL/SQL table item as Integer by Index.
<u>ItemAsInterval</u>	Used to access a PL/SQL table item as TOraInterval by Index.
<u>ItemAsString</u>	Used to access a PL/SQL table item as String by Index.
<u>ItemAsTimeStamp</u>	Used to access a PL/SQL table item as TOraTimeStamp by Index.
<u>ItemIsNull</u>	Used to indicate if an item value is Null.
<u>ItemText</u>	Allows to modify data without changing its type.
<u>ItemValue</u>	Used to access the item value as variant.
<u>Length</u>	Used to determine the maximum length of a PL/SQL table parameter.
<u>National</u>	Used to determine whether the parameter is in National charset.
<u>ParamType</u> (inherited from <u>TDAParam</u>)	Used to indicate the type of use for a parameter.
<u>Size</u> (inherited from <u>TDAParam</u>)	Specifies the size of a string type parameter.
<u>Table</u>	Used to determine if the parameter is a PL/SQL table.
<u>Value</u> (inherited from <u>TDAParam</u>)	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.

© 1997-2012 Devart. All Rights Reserved.

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	Used to specify the value of the parameter when it represents the value of the Oracle reference type.
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.
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 specify 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.
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](#) (inherited from [TDAParam](#))

Overloaded. Writes the data from a specified buffer to BLOB.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 property 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 property 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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used 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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to specify 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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 charset.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TOraParam** class.

For a complete list of the **TOraParam** class members, see the [TOraParam Members](#) topic.

Public

Name	Description
AsBlob (inherited from TDAParam)	Used to set and read the value of the BLOB parameter as string.
AsBlobRef (inherited from TDAParam)	Used to set and read the value of the BLOB parameter as a TBlob object.

[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.

[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.

[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.

[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.

[AsWideString](#) (inherited from [TDAParam](#))

Used to assign the Unicode string value to the parameter.

[IsNull](#) (inherited from [TDAParam](#))

Used to indicate whether the value assigned to a parameter is NULL.

[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.

Published

Name	Description
DataType (inherited from TDAParam)	Indicates the data type of the parameter.
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.
Value (inherited from TDAParam)	Used to represent the value of the parameter as Variant.

See Also

- [TOraParam Class](#)
- [TOraParam Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#).

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.16 Ora.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

TObject

[TDAParams](#)

ToraParams

See Also

- [TOraParam](#)
- [TCustomDASQL.Params](#)
- [TCustomDADataset.Params](#)
- [TOraDataSet.Params](#)
- [TOraSQL.Params](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraParams** class.

For a complete list of the **TOraParams** class members, see the [TOraParams Members](#) topic.

Public

Name	Description
FindParam (inherited from TDAParams)	Searches for a parameter with the specified name.
Items	Used to iterate through all field parameters.
ParamByName (inherited from TDAParams)	Searches for a parameter with the specified name.

See Also

- [TOraParams Class](#)
- [TOraParams Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.17 Ora.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

TObject
 [TPoolingOptions](#)
 ToraPoolingOptions

© 1997-2012 Devart. All Rights Reserved.

[ToraPoolingOptions](#) class overview.

Properties

Name	Description
ConnectionLifetime (inherited from TPoolingOptions)	Used to specify the maximum time during which an opened 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.
PoolType	Used to specify the pool type.
ProxyPassword	Used to specify a password for proxy pooling.
ProxyUsername	Used 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.

© 1997-2012 Devart. All Rights Reserved.

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 opened 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.
PoolType	Used to specify the pool type.
ProxyPassword	Used to specify a password for proxy pooling.
ProxyUsername	Used 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](#)

© 1997-2012 Devart. All Rights Reserved.

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 [OraConnectionPool](#) unit to "uses" list to use the PoolType option at run-time.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used 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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.18 Ora.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

```

TObject
  TMemDataSet
    TCustomDADataset
      TORADataSet
        TCustomOraQuery
          TORAQuery
  
```

See Also

- Query demo project
- [Updating Data with ODAC Dataset Components](#)
- [Master/Detail Relationships](#)
- [ToraStoredProc](#)
- [ToraTable](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.

[DetailFields](#) (inherited from [TCustomDADataset](#))

[Disconnected](#) (inherited from [TCustomDADataset](#))

[DMLRefresh](#) (inherited from [TOraDataSet](#))

[Encryption](#) (inherited from [TCustomDADataset](#))

[FetchAll](#)

[FetchRows](#) (inherited from [TCustomDADataset](#))

[FilterSQL](#) (inherited from [TCustomDADataset](#))

[FinalSQL](#) (inherited from [TCustomDADataset](#))

[IndexFieldNames](#) (inherited from [TMemDataSet](#))

[IsPLSQL](#) (inherited from [TOraDataSet](#))

[IsQuery](#) (inherited from [TOraDataSet](#))

[KeyFields](#) (inherited from [TOraDataSet](#))

[KeySequence](#) (inherited from [TOraDataSet](#))

[LocalConstraints](#) (inherited from [TMemDataSet](#))

[LocalUpdate](#) (inherited from [TMemDataSet](#))

[LockMode](#)

[MacroCount](#) (inherited from [TCustomDADataset](#))

[Macros](#) (inherited from [TCustomDADataset](#))

[MasterFields](#) (inherited from [TCustomDADataset](#))

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

Used to keep dataset opened after connection is closed.

Used to refresh record by RETURNING clause when insert or update is performed.

Used to specify the options of the data encryption in a dataset.

Defines whether to request all records of the query from database server when the dataset is being opened.

Used to define the number of rows to be transferred across the network at the same time.

Used to change the WHERE clause of SELECT statement and reopen a query.

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Used to get or set the list of fields on which the recordset is sorted.

Indicates whether a SQL statement is a PL/SQL block.

Indicates whether SQL statement returns rows or not.

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.

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.

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Used to prevent implicit update of rows on database server.

Used to specify what kind of lock will be performed when editing a record.

Used to get the number of macros associated with the Macros property.

Makes it possible to change SQL queries easily.

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.

<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>NonBlocking</u> (inherited from <u>TOraDataSet</u>)	Used to execute a SQL statement and fetch rows by a separate thread.
<u>Options</u> (inherited from <u>TOraDataSet</u>)	Used to specify the behaviour of TOrDataSetObject.
<u>OptionsDS</u> (inherited from <u>TOraDataSet</u>)	Used to specify the behaviour of TOrDataSetObject.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TOraDataSet</u>)	Contains the parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify when to refresh an editing record.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>ReturnParams</u> (inherited from <u>TOraDataSet</u>)	Used to return a new fields value to dataset after insert or update.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>RowsProcessed</u> (inherited from <u>TOraDataSet</u>)	Returns the number of rows processed by a query.
<u>SequenceMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>Session</u> (inherited from <u>TOraDataSet</u>)	Used to specify the session in which dataset will be executed.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLType</u> (inherited from <u>TOraDataSet</u>)	Used to get the typecode of the SQL statement being processed by Oracle database server.

[SQLUpdate](#) (inherited from [TCustomDADataset](#))

[StrictUpdate](#) (inherited from [TOraDataSet](#))

[UniDirectional](#) (inherited from [TCustomDADataset](#))

[UpdateObject](#) (inherited from [TOraDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

[UpdatingTable](#)

Used to specify a SQL statement that will be used when applying an update to a dataset.

Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.

Used if an application does not need bidirectional access to records in the result set.

Used to point to an update object component which provides SQL statements that perform updates of the read-only datasets.

Used to indicate the update status for the current record when cached updates are enabled.

Used to check the status of the cached updates buffer.

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.
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.
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.
ErrorOffset (inherited from TOraDataSet)	Returns the parse error offset.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
Fetched (inherited from TOraDataSet)	Indicates if TCustomDADataset has already fetched all rows.
Fetching (inherited from TCustomDADataset)	Used to learn whether TCustomDADataset is still fetching rows.

<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether a specified macro exists in a dataset.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TOraDataSet</u>)	Determines whether a parameter with the specified name exists in a dataset.
<u>GetArray</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORAArray object for a field when only its name is known.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetErrorPos</u> (inherited from <u>TOraDataSet</u>)	Returns a row and column of parse error for a SQL statement.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetFile</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORAFile object for a field with known name.
<u>GetInterval</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORAInterval object for a field with known name.
<u>GetKeyList</u> (inherited from <u>TOraDataSet</u>)	Returns the list of table primary key fields.
<u>GetLob</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORAlob object for a field with known name.
<u>GetLobLocator</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORAlob object for a field with known name.
<u>GetObject</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORAobject object for a field with known name.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetRef</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORAref object for a field with known name.
<u>GetTable</u> (inherited from <u>TOraDataSet</u>)	Retrieve a TORAnestTable object for a field with known name.
<u>GetTimeStamp</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TORATimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.

[LocateEx](#) (inherited from [TMemDataSet](#))

[Lock](#) (inherited from [TCustomDADataset](#))

[MacroByName](#) (inherited from [TCustomDADataset](#))

[ParamByName](#) (inherited from [TOraDataSet](#))

[Prepare](#) (inherited from [TCustomDADataset](#))

[RefreshRecord](#) (inherited from [TCustomDADataset](#))

[RestoreSQL](#) (inherited from [TCustomDADataset](#))

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[Resync](#) (inherited from [TCustomDADataset](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Locks the current record.

Finds a Macro with the name passed in Name.

Sets or uses parameter information for a specific parameter based on its name.

Allocates, opens, and parses cursor for a query.

Actualizes field values for the current record.

Restores the SQL property modified by AddWhere and SetOrderBy.

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Cancels changes made to the current record when cached updates are enabled.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Builds an ORDER BY clause of a SELECT statement.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Indicates the current update status for the dataset when cached updates are enabled.

Events

Name

[AfterExecute](#) (inherited from [TCustomDADataset](#))

[AfterFetch](#) (inherited from [TCustomDADataset](#))

[AfterUpdateExecute](#) (inherited from [TCustomDADataset](#))

[BeforeFetch](#) (inherited from [TCustomDADataset](#))

Description

Occurs after a component has executed a query to database.

Occurs after dataset finishes fetching data from server.

Occurs after executing insert, delete, update, lock and refresh operations.

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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **ToraQuery** class.

For a complete list of the **ToraQuery** class members, see the [ToraQuery Members](#) topic.

Public

Name	Description
AddWhere (inherited from TCustomDADataset)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
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.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
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.

<u>Cursor</u> (inherited from <u>TOraDataSet</u>)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TOraDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>Encryption</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the options of the data encryption in a dataset.
<u>ErrorOffset</u> (inherited from <u>TOraDataSet</u>)	Returns the parse error offset.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TOraDataSet</u>)	Indicates if TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether a specified macro exists in a dataset.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TOraDataSet</u>)	Determines whether a parameter with the specified name exists in a dataset.
<u>GetArray</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrArray object for a field when only its name is known.

<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetErrorPos</u> (inherited from <u>TOraDataSet</u>)	Returns a row and column of parse error for a SQL statement.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetFile</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraFile object for a field with known name.
<u>GetInterval</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraInterval object for a field with known name.
<u>GetKeyList</u> (inherited from <u>TOraDataSet</u>)	Returns the list of table primary key fields.
<u>GetLob</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraLob object for a field with known name.
<u>GetLobLocator</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraLob object for a field with known name.
<u>GetObject</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraObject object for a field with known name.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetRef</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraRef object for a field with known name.
<u>GetTable</u> (inherited from <u>TOraDataSet</u>)	Retrieve a TOraNestTable object for a field with known name.
<u>GetTimeStamp</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsPLSQL</u> (inherited from <u>TOraDataSet</u>)	Indicates whether a SQL statement is a PL/SQL block.
<u>IsQuery</u> (inherited from <u>TOraDataSet</u>)	Indicates whether SQL statement returns rows or not.
<u>KeyFields</u> (inherited from <u>TOraDataSet</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
<u>KeySequence</u> (inherited from <u>TOraDataSet</u>)	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.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.

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 name passed in Name.
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.
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.
Options (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
ParamByName (inherited from TOraDataSet)	Sets or uses parameter information for a specific parameter based on its name.
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.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.

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.
Resync (inherited from TCustomDADataset)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
ReturnParams (inherited from TOraDataset)	Used to return a new fields value to dataset after insert or update.
RevertRecord (inherited from TMemDataset)	Cancels changes made to the current record when cached updates are enabled.
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.
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.
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.
SetOrderBy (inherited from TCustomDADataset)	Builds an ORDER BY clause of a SELECT statement.
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.
SQLRefresh (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.
SQLSaved (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.
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.

[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.

[UpdateObject](#) (inherited from [TOraDataSet](#))

Used to point to 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.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.19 Ora.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.

Inheritance Hierarchy

TObject

ToraReferenceField

© 1997-2012 Devart. All Rights Reserved.

[TOraReferenceField](#) class overview.

Properties

Name	Description
Modified	Used to indicate whether a field was modified.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.20 Ora.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

TObject

[TCustomDAConnection](#)

TOrasession

See Also

- [TOraDataSet.Session](#)
- [TOraSQL.Session](#)
- [TBDESession](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
ConnectionString	Used to assign the TOraSession.Username , TOraSession.Password and TOraSession.Server properties at once.
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	Not supported.
HomeName	Used to select the Oracle client to use with the application.
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.

[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.
ChangePassword	Changes the current user password for the session by a new one.
Commit (inherited from TCustomDAConnection)	Commits current transaction.
Connect (inherited from TCustomDAConnection)	Establishes a connection to the server.
CreateDataSet (inherited from TCustomDAConnection)	Creates a dataset component.
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.
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	Checks whether the connection to the server can be established.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
Commit (inherited from TCustomDAConnection)	Commits current transaction.
Connect (inherited from TCustomDAConnection)	Establishes a connection to the server.
ConnectDialog (inherited from TCustomDAConnection)	Allows to link a TCustomConnectDialog component.
ConvertEOL (inherited from TCustomDAConnection)	Allows customizing line breaks in string fields and parameters.
CreateDataSet (inherited from TCustomDAConnection)	Creates a dataset component.
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.
GetStoredProcNames (inherited from TCustomDAConnection)	Returns a list of stored procedures from the server.
GetTableNames (inherited from TCustomDAConnection)	Provides a list of available tables names.
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.
MonitorMessage (inherited from TCustomDAConnection)	Sends a specified message through the TCustomDASQLMonitor component.
OCICallStyle	Indicates the set of OCI routines used.
OCISvcCtx	Used to return Oracle 8 service context handle of the current connection.
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.
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.
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.
SQL	Uses embedded TOraSQL object to execute any SQL statement.
StartTransaction (inherited from TCustomDAConnection)	Begins a new user transaction.

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.
ConnectionString	Used to assign the TOraSession.Username , TOraSession.Password and TOraSession.Server properties at once.
Debug	Used to display SQL statements being executed with their parameter values and data types.
Home	Not supported.
HomeName	Used to select the Oracle client to use with the application.
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.
ThreadSafety	Used to allow the usage of the OCI in multi-threaded environment.
Username	Contains username.

See Also

- [TOraSession Class](#)
- [TOraSession Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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 TOraSession 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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to specify the system privileges to use when a user connects to the server.

Class

[TOraSession](#)

Syntax


```
property ConnectMode: TConnectMode default cmNORMAL;
```

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
- [ConnectionString](#)

© 1997-2012 Devart. All Rights Reserved.

Used to assign the [Username](#), [Password](#) and [Server](#) properties at once.

Class

[TOraSession](#)

Syntax

```
property ConnectString: string;
```

Remarks

Use the ConnectString property to assign such properties as [Username](#), [Password](#) and [Server](#) at once. Valid format is Username/Password@Server.

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

See Also

- [TCustomDAConnection.Password](#)

- [TCustomDAConnection.Username](#)
 - [TCustomDAConnection.Server](#)
 - [TCustomDAConnection.Connect](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Not supported.

Class

[TOraSession](#)

Syntax

```
property Home: TOracleHome stored False default ohDefault;
```

Remarks

This property is out of date and is not supported any more. 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.

The default value is ohDefault.

See Also

- [TCustomDAConnection.Connect](#)
 - [OraCall](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
- [OraCall](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to return Oracle 8 service context handle of the current connection.

Class

[TOraSession](#)

Syntax

property OCISvcCtx: pOCISvcCtx;

Remarks

Use the OCISvcCtx property to return Oracle 8 service context handle of the current connection.

See Also

- [LDA](#)
- [AssignConnect](#)

© 1997-2012 Devart. All Rights Reserved.

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.
EnableIntegers	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.
NeverConnect	Used to prevent an application from establishing a connection at the time of startup.
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.
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](#)

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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	Used to specify a user name for proxy pooling.

See Also

- [TCustomDAConnection.Pooling](#)

©

1997-2012 Devart. All Rights Reserved.

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 be 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](#)=False.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 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.

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:

If prefix sid= or sn= aren't set then connection will be established using SID.

See Also

- [Username](#)
 - [Password](#)
 - [TCustomDAConnection.Connect](#)
 - [HomeName](#)
 - [Connecting in Direct Mode](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
ChangePassword	Changes the current user password for the session by a new one.
Commit (inherited from TCustomDAConnection)	Commits current transaction.
Connect (inherited from TCustomDAConnection)	Establishes a connection to the server.
ConnectDialog (inherited from TCustomDAConnection)	Allows to link a TCustomConnectDialog component.
ConvertEOL (inherited from TCustomDAConnection)	Allows customizing line breaks in string fields and parameters.
CreateDataSet (inherited from TCustomDAConnection)	Creates a dataset component.
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.
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.
InTransaction (inherited from TCustomDAConnection)	Indicates whether the transaction is active.
LoginPrompt (inherited from TCustomDAConnection)	Specifies whether a login dialog appears immediately before opening a new connection.
MonitorMessage (inherited from TCustomDAConnection)	Sends a specified message through the TCustomDASQLMonitor component.
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.
Options (inherited from TCustomDAConnection)	Specifies the connection behavior.
ParamByName	Provides access to OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc.
Password (inherited from TCustomDAConnection)	Serves to supply a password for login.
Ping	Checks whether the connection to the server can be established.
Pooling (inherited from TCustomDAConnection)	Enables or disables using connection pool.
PoolingOptions (inherited from TCustomDAConnection)	Specifies the behaviour of connection pool.
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.
Server (inherited from TCustomDAConnection)	Serves to supply the server name for login.
StartTransaction	Overloaded. Begins a new user transaction against the database server.
Username (inherited from TCustomDAConnection)	Used to supply a user name for login.

See Also

- [TOraSession Class](#)
- [TOraSession Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Shares database connection between the TOraSession components.

Class

[TOraSession](#)

Syntax

```
procedure AssignConnect (Source: TOraSession);  
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(Null) or Disconnect method for all assigned sessions.

Example

```
OraSession1.Connect;
OraSession2.AssignConnect (OraSession1);
// OraSession2.Connected is True
OraSQL.Session := OraSession2;
OraSQL.Execute;
OraSession2.AssignConnect (Null);
// OraSession2.Connected is False
OraSession1.Disconnect;
```

See Also

- [LDA](#)
- [OCISvcCtx](#)
- [TCustomDAConnection.Connect](#)

© 1997-2012 Devart. All Rights Reserved.

Changes the current user password for the session by a new one.

Class

[TOraSession](#)

Syntax

```
procedure ChangePassword(NewPassword: string);
```

Parameters*NewPassword*

Holds the new password.

Remarks

Call the ChangePassword method to replace current user password for this session by a new one. The previous value for Password and UserName properties must be provided before calling ChangePassword.

ChangePassword method is used mainly when logging to the user account fails due to the password expiration or any other relevant reason accompanied by an exception with ORA-2800 Oracle error code family (see Oracle Error Messages).

See Also

- [Username](#)
- [Password](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.
Implicitly calls ParamByName function of TOraSQL.

See Also

- [SQL](#)
 - [TCustomDAConnection.ExecSQL](#)
 - [TCustomDAConnection.ExecSQLEx](#)
-

© 1997-2012 Devart. All Rights Reserved.

Checks whether the connection to the server can be established.

Class

[TOraSession](#)

Syntax

```
procedure Ping;
```

Remarks

Use the Ping method to check whether the connection to the server can be established. If it cannot, an exception is raised.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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: <u>TOraIsolationLevel</u>; const RollbackSegment: string; const Name: string)	Begins a new user transaction against the database server.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Events 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.
Commit (inherited from TCustomDAConnection)	Commits current transaction.
Connect (inherited from TCustomDAConnection)	Establishes a connection to the server.
ConnectDialog (inherited from TCustomDAConnection)	Allows to link a TCustomConnectDialog component.
ConvertEOL (inherited from TCustomDAConnection)	Allows customizing line breaks in string fields and parameters.
CreateDataSet (inherited from TCustomDAConnection)	Creates a dataset component.
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.
GetStoredProcNames (inherited from TCustomDAConnection)	Returns a list of stored procedures from the server.
GetTableNames (inherited from TCustomDAConnection)	Provides a list of available tables names.
InTransaction (inherited from TCustomDAConnection)	Indicates whether the transaction is active.
LoginPrompt (inherited from TCustomDAConnection)	Specifies whether a login dialog appears immediately before opening a new connection.
MonitorMessage (inherited from TCustomDAConnection)	Sends a specified message through the TCustomDASQLMonitor component.
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.
Options (inherited from TCustomDAConnection)	Specifies the connection behavior.
Password (inherited from TCustomDAConnection)	Serves to supply a password for login.
Pooling (inherited from TCustomDAConnection)	Enables or disables using connection pool.
PoolingOptions (inherited from TCustomDAConnection)	Specifies the behaviour of connection pool.

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.
Server (inherited from TCustomDAConnection)	Serves to supply the server name for login.
StartTransaction (inherited from TCustomDAConnection)	Begins a new user transaction.
Username (inherited from TCustomDAConnection)	Used to supply a user name for login.

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.

See Also

- [TOraSession Class](#)
- [TOraSession Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.21 Ora.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 (TDACConnectionOptions) ;
```

Inheritance Hierarchy

TObject

[TDACConnectionOptions](#)

ToraSessionOptions

© 1997-2012 Devart. All Rights Reserved.

[ToraSessionOptions](#) class overview.

Properties

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.
DefaultSortType (inherited from TDACConnectionOptions)	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 TDACConnectionOptions)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
EnableIntegers	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.
KeepDesignConnected (inherited from TDACConnectionOptions)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from TDACConnectionOptions)	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
NeverConnect	Used to prevent an application from establishing a connection at the time of startup.
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.
UseOCI7	Used to force TOraSession use OCI 7 call style only.
UseUnicode	Used to enable or disable Unicode support.

© 1997-2012 Devart. All Rights Reserved.

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 TDACConnectionOptions)	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 TDACConnectionOptions)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected (inherited from TDACConnectionOptions)	Used to prevent an application from establishing a connection at the time of startup.

[LocalFailover](#) (inherited from [TDACConnectionOptions](#))

If True, the [TCustomDAConnection.OnConnectionLost](#) event occurs and a failover operation can be performed after connection breaks.

Published

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.
EnableIntegers	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.
NeverConnect	Used to prevent an application from establishing a connection at the time of startup.
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.
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](#)

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.

© 1997-2012 Devart. All Rights Reserved.

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 only.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to specify the time to wait for a connection to open before raising an exception.

Class

[TOraSessionOptions](#)

Syntax

```
property ConnectionTimeout: integer default 0;
```

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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".

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 False;
```

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 Developer Editions.

© 1997-2012 Devart. All Rights Reserved.

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](#).

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 TSQLErrorTimestampField is created for columns of TIMESTAMP data type.

© 1997-2012 Devart. All Rights Reserved.

Used to prevent an application from establishing a connection at the time of startup.

Class

[TOraSessionOptions](#)

Syntax

```
property NeverConnect: boolean stored False default False;
```

Remarks

Use the NeverConnect property to prevent an application from establishing a connection at the time of startup even if the Connected property was set to True at design-time.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to enable or disable Unicode support.

Class

[TOraSessionOptions](#)

Syntax

```
property UseUnicode: boolean default False;
```

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 WideString and TStringField is replaced with TWideStringField. Supported starting with Oracle 8.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.22 Ora.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

```
TObject
  TCustomDASQL
    ToraSQL
```

See Also

- SQL demo project
- [TOraQuery](#)
- [TOraScript](#)

© 1997-2012 Devart. All Rights Reserved.

[TOraSQL](#) class overview.

Properties

Name	Description
ArrayLength	Used to set the Length property 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 executing statement, all its parameters' values, and the type of 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	Description is not available at the moment.
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 associated 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	Breaks execution of a SQL statement 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 SQL commands.
Executing (inherited from TCustomDASQL)	Checks whether TCustomDASQL still executes a SQL statement.

FindMacro (inherited from TCustomDASQL)	Searches for 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 name passed in 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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOrasQL** class.

For a complete list of the **TOrasQL** class members, see the [TOrasQL Members](#) topic.

Public

Name	Description
AfterExecute (inherited from TCustomDASQL)	Occurs after a SQL statement has been executed.
ArrayLength	Used to set the Length property 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 executing statement, all its parameters' values, and the type of parameters.
Execute (inherited from TCustomDASQL)	Overloaded. Executes SQL commands.
Executing (inherited from TCustomDASQL)	Checks whether TCustomDASQL still executes a SQL statement.
FinalSQL (inherited from TCustomDASQL)	Used to return a SQL statement with expanded macros.
FindMacro (inherited from TCustomDASQL)	Searches for a macro with the specified name.
FindParam (inherited from TCustomDASQL)	Finds a parameter with the specified name.
MacroByName (inherited from TCustomDASQL)	Finds a Macro with the name passed in Name.
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.
ParamByName (inherited from TCustomDASQL)	Finds a parameter with the specified name.

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.
Prepare (inherited from TCustomDASQL)	Allocates, opens, and parses cursor for a query.
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.
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.

Published

Name	Description
NonBlocking	Description is not available at the moment.
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 associated 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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to set the Length property to all parameters.

Class

[TOraSQL](#)

Syntax

property `ArrayLength: integer;`

Remarks

Use the `ArrayLength` property to set the `Length` property to all parameters. It is useful for DML array operations.

See Also

- [TOraParam.Length](#)

© 1997-2012 Devart. All Rights Reserved.

Class

[TOraSQL](#)

Syntax

property `NonBlocking: boolean default False;`

Remarks

Set the `NonBlocking` property to `True` to execute a SQL statement by a separate thread.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to get a value indicating whether Oracle resources associated 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.StatementCache](#) to false if you don't want the statements to be cached.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TOraSQL** class.

For a complete list of the **TOraSQL** class members, see the [TOraSQL Members](#) topic.

Public

Name	Description
AfterExecute (inherited from TCustomDASQL)	Occurs after a SQL statement has been executed.
BreakExec	Breaks execution of a SQL statement on the server.
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.
CreateProcCall	Assigns a PL/SQL block that calls stored procedure
Debug (inherited from TCustomDASQL)	Used to display executing statement, all its parameters' values, and the type of parameters.
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 SQL commands.
Executing (inherited from TCustomDASQL)	Checks whether TCustomDASQL still executes a SQL statement.
FinalSQL (inherited from TCustomDASQL)	Used to return a SQL statement with expanded macros.

FindMacro (inherited from TCustomDASQL)	Searches for 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 name passed in Name.
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.
ParamByName	Searches for and returns a parameter with the specified name.
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 (inherited from TCustomDASQL)	Used to contain 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.
Prepare (inherited from TCustomDASQL)	Allocates, opens, and parses cursor for a query.
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.
SQL (inherited from TCustomDASQL)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.
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](#)

© 1997-2012 Devart. All Rights Reserved.

Breaks execution of a SQL statement on the server.

Class

[TOraSQL](#)

Syntax

```
procedure BreakExec;
```

Remarks

Call the BreakExec method to break execution of the SQL statement on the server. It makes sense to call BreakExec only from another thread. Useful when NonBlocking is True.

See Also

- [TCustomDASQL.Execute](#)
 - [NonBlocking](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.23 Ora.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

```

TObject
  TMemDataSet
    TCustomDADataset
      TOraDataSet
        TCustomOraQuery
          TOraStoredProc

```

See Also

- Stored proc demo
- [TOraQuery](#)
- [TOraSQL](#)
- [Updating Data with ODAC Dataset Components](#)
- [TCustomDACConnection.ExecProc](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of 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 TCustomDADataset)	Used to specify the options of the data encryption 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.
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.
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.
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.
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 point to 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.
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.
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.
ErrorOffset (inherited from TOraDataSet)	Returns the parse error offset.
ExecProc	Executes a SQL statement on the server.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
Fetched (inherited from TOraDataSet)	Indicates if TCustomDADataset has already 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](#))

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TOraDataSet](#))

[GetArray](#) (inherited from [TOraDataSet](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetErrorPos](#) (inherited from [TOraDataSet](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetFile](#) (inherited from [TOraDataSet](#))

[GetInterval](#) (inherited from [TOraDataSet](#))

[GetKeyList](#) (inherited from [TOraDataSet](#))

[GetLob](#) (inherited from [TOraDataSet](#))

[GetLobLocator](#) (inherited from [TOraDataSet](#))

[GetObject](#) (inherited from [TOraDataSet](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GetRef](#) (inherited from [TOraDataSet](#))

[GetTable](#) (inherited from [TOraDataSet](#))

[GetTimeStamp](#) (inherited from [TOraDataSet](#))

[GotoCurrent](#) (inherited from [TCustomDADataset](#))

[Locate](#) (inherited from [TMemDataSet](#))

[LocateEx](#) (inherited from [TMemDataSet](#))

[Lock](#) (inherited from [TCustomDADataset](#))

[MacroByName](#) (inherited from [TCustomDADataset](#))

Indicates whether a specified macro exists in a dataset.

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.

Determines whether a parameter with the specified name exists in a dataset.

Retrieves a TOrArray object for a field when only its name is known.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Returns internal field types defined in the MemData and accompanying modules.

Returns a row and column of parse error for a SQL statement.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves a TOrFile object for a field with known name.

Retrieves a TOrInterval object for a field with known name.

Returns the list of table primary key fields.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrObject object for a field with known name.

Retrieves an ORDER BY clause from a SQL statement.

Retrieves a TOrRef object for a field with known name.

Retrieve a TOrNestTable object for a field with known name.

Retrieves a TOrTimeStamp object for a field with known name.

Sets the current record in this dataset similar to the current record in another dataset.

Overloaded. Searches a dataset for a specific record and positions the cursor on it.

Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Locks the current record.

Finds a Macro with the name passed in Name.

[ParamByName](#) (inherited from [TOraDataSet](#))

[Prepare](#) (inherited from [TCustomDADataset](#))

[PrepareSQL](#)

[RefreshRecord](#) (inherited from [TCustomDADataset](#))

[RestoreSQL](#) (inherited from [TCustomDADataset](#))

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[Resync](#) (inherited from [TCustomDADataset](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Sets or uses parameter information for a specific parameter based on its name.

Allocates, opens, and parses cursor for a query.

Describes the parameters of a stored procedure.

Actualizes field values for the current record.

Restores the SQL property modified by AddWhere and SetOrderBy.

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Cancels changes made to the current record when cached updates are enabled.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Builds an ORDER BY clause of a SELECT statement.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
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.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
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.
Cursor (inherited from TOraDataSet)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.

<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TOraDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>Encryption</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the options of the data encryption in a dataset.
<u>ErrorOffset</u> (inherited from <u>TOraDataSet</u>)	Returns the parse error offset.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>FetchAll</u> (inherited from <u>TOraDataSet</u>)	Used to request all records of the query from database server when a dataset is being opened.
<u>Fetched</u> (inherited from <u>TOraDataSet</u>)	Indicates if TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether a specified macro exists in a dataset.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TOraDataSet</u>)	Determines whether a parameter with the specified name exists in a dataset.
<u>GetArray</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrArray object for a field when only its name is known.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetErrorPos](#) (inherited from [TOraDataSet](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetFile](#) (inherited from [TOraDataSet](#))

[GetInterval](#) (inherited from [TOraDataSet](#))

[GetKeyList](#) (inherited from [TOraDataSet](#))

[GetLob](#) (inherited from [TOraDataSet](#))

[GetLobLocator](#) (inherited from [TOraDataSet](#))

[GetObject](#) (inherited from [TOraDataSet](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GetRef](#) (inherited from [TOraDataSet](#))

[GetTable](#) (inherited from [TOraDataSet](#))

[GetTimeStamp](#) (inherited from [TOraDataSet](#))

[GotoCurrent](#) (inherited from [TCustomDADataset](#))

[IndexFieldNames](#) (inherited from [TMemDataSet](#))

[IsPLSQL](#) (inherited from [TOraDataSet](#))

[IsQuery](#) (inherited from [TOraDataSet](#))

[KeyFields](#) (inherited from [TOraDataSet](#))

[KeySequence](#) (inherited from [TOraDataSet](#))

[LocalConstraints](#) (inherited from [TMemDataSet](#))

[LocalUpdate](#) (inherited from [TMemDataSet](#))

[Locate](#) (inherited from [TMemDataSet](#))

Returns internal field types defined in the MemData and accompanying modules.

Returns a row and column of parse error for a SQL statement.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves a TORAFile object for a field with known name.

Retrieves a TORAInterval object for a field with known name.

Returns the list of table primary key fields.

Retrieves a TORALob object for a field with known name.

Retrieves a TORALob object for a field with known name.

Retrieves a TORAObject object for a field with known name.

Retrieves an ORDER BY clause from a SQL statement.

Retrieves a TORARef object for a field with known name.

Retrieve a TORANestTable object for a field with known name.

Retrieves a TORATimeStamp object for a field with known name.

Sets the current record in this dataset similar to the current record in another dataset.

Used to get or set the list of fields on which the recordset is sorted.

Indicates whether a SQL statement is a PL/SQL block.

Indicates whether SQL statement returns rows or not.

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.

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.

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Used to prevent implicit update of rows on database server.

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 name passed in Name.
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.
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.
Options (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
ParamByName (inherited from TOraDataSet)	Sets or uses parameter information for a specific parameter based on its name.
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.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.
RefreshRecord (inherited from TCustomDADataset)	Actualizes field values for the current record.

[RestoreSQL](#) (inherited from [TCustomDADataset](#))

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[Resync](#) (inherited from [TCustomDADataset](#))

[ReturnParams](#) (inherited from [TOraDataSet](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[RowsAffected](#) (inherited from [TCustomDADataset](#))

[RowsProcessed](#) (inherited from [TOraDataSet](#))

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SequenceMode](#) (inherited from [TOraDataSet](#))

[Session](#) (inherited from [TOraDataSet](#))

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQL](#) (inherited from [TCustomDADataset](#))

[SQLDelete](#) (inherited from [TCustomDADataset](#))

[SQLInsert](#) (inherited from [TCustomDADataset](#))

[SQLRefresh](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[SQLType](#) (inherited from [TOraDataSet](#))

[SQLUpdate](#) (inherited from [TCustomDADataset](#))

[StrictUpdate](#) (inherited from [TOraDataSet](#))

Restores the SQL property modified by AddWhere and SetOrderBy.

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Used to return a new fields value to dataset after insert or update.

Cancels changes made to the current record when cached updates are enabled.

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Returns the number of rows processed by a query.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Used to specify the methods used internally to generate a sequenced field.

Used to specify the session in which dataset will be executed.

Builds an ORDER BY clause of a SELECT statement.

Used to provide a SQL statement that a query component executes when its Open method is called.

Used to specify a SQL statement that will be used when applying a deletion to a record.

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataset.RefreshRecord](#) procedure.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Used to get the typecode of the SQL statement being processed by Oracle database server.

Used to specify a SQL statement that will be used when applying an update to a dataset.

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.

[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.

[UpdateObject](#) (inherited from [TOraDataSet](#))

Used to point to 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.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.

<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>ChangeNotification</u> (inherited from <u>TOraDataSet</u>)	Used to receive database change notification messages to refresh dataset when required.
<u>CheckMode</u> (inherited from <u>TOraDataSet</u>)	Used to define the check mode before editing a record.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>CreateBlobStream</u> (inherited from <u>TCustomDADataset</u>)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
<u>CreateProcCall</u> (inherited from <u>TOraDataSet</u>)	Generates the stored procedure call.
<u>Cursor</u> (inherited from <u>TOraDataSet</u>)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TOraDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>Encryption</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the options of the data encryption in a dataset.
<u>ErrorOffset</u> (inherited from <u>TOraDataSet</u>)	Returns the parse error offset.
<u>ExecProc</u>	Executes a SQL statement on the server.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>FetchAll</u> (inherited from <u>TOraDataSet</u>)	Used to request all records of the query from database server when a dataset is being opened.
<u>Fetches</u> (inherited from <u>TOraDataSet</u>)	Indicates if TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.

[FetchRows](#) (inherited from [TCustomDADataset](#))

[FilterSQL](#) (inherited from [TCustomDADataset](#))

[FinalSQL](#) (inherited from [TCustomDADataset](#))

[FindKey](#) (inherited from [TCustomDADataset](#))

[FindMacro](#) (inherited from [TCustomDADataset](#))

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TOraDataSet](#))

[GetArray](#) (inherited from [TOraDataSet](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetErrorPos](#) (inherited from [TOraDataSet](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetFile](#) (inherited from [TOraDataSet](#))

[GetInterval](#) (inherited from [TOraDataSet](#))

[GetKeyList](#) (inherited from [TOraDataSet](#))

[GetLob](#) (inherited from [TOraDataSet](#))

[GetLobLocator](#) (inherited from [TOraDataSet](#))

[GetObject](#) (inherited from [TOraDataSet](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GetRef](#) (inherited from [TOraDataSet](#))

[GetTable](#) (inherited from [TOraDataSet](#))

[GetTimeStamp](#) (inherited from [TOraDataSet](#))

Used to define the number of rows to be transferred across the network at the same time.

Used to change the WHERE clause of SELECT statement and reopen a query.

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Searches for a record which contains specified field values.

Indicates whether a specified macro exists in a dataset.

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.

Determines whether a parameter with the specified name exists in a dataset.

Retrieves a TOrArray object for a field when only its name is known.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Returns internal field types defined in the MemData and accompanying modules.

Returns a row and column of parse error for a SQL statement.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves a TOrFile object for a field with known name.

Retrieves a TOrInterval object for a field with known name.

Returns the list of table primary key fields.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrObject object for a field with known name.

Retrieves an ORDER BY clause from a SQL statement.

Retrieves a TOrRef object for a field with known name.

Retrieve a TOrNestTable object for a field with known name.

Retrieves a TOrTimeStamp object for a field with known name.

<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsPLSQL</u> (inherited from <u>TOraDataSet</u>)	Indicates whether a SQL statement is a PL/SQL block.
<u>IsQuery</u> (inherited from <u>TOraDataSet</u>)	Indicates whether SQL statement returns rows or not.
<u>KeyFields</u> (inherited from <u>TOraDataSet</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
<u>KeySequence</u> (inherited from <u>TOraDataSet</u>)	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.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>LockMode</u> (inherited from <u>TOraDataSet</u>)	Used to define when to perform the locking of an editing record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>NonBlocking</u> (inherited from <u>TOraDataSet</u>)	Used to execute a SQL statement and fetch rows by a separate thread.
<u>OnUpdateError</u> (inherited from <u>TMemDataSet</u>)	Occurs when an exception is generated while cached updates are applied to a database.
<u>OnUpdateRecord</u> (inherited from <u>TMemDataSet</u>)	Occurs when a single update component can not handle the updates.
<u>Options</u> (inherited from <u>TOraDataSet</u>)	Used to specify the behaviour of TOraDataSetObject.

<u>OptionsDS</u> (inherited from <u>TOraDataSet</u>)	Used to specify the behaviour of TOraDataSetObject.
<u>ParamByName</u> (inherited from <u>TOraDataSet</u>)	Sets or uses parameter information for a specific parameter based on its name.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TOraDataSet</u>)	Contains the parameters for a query's SQL statement.
<u>Prepare</u> (inherited from <u>TCustomDADataset</u>)	Allocates, opens, and parses cursor for a query.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>PrepareSQL</u>	Describes the parameters of a stored procedure.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify when to refresh an editing record.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>Resync</u> (inherited from <u>TCustomDADataset</u>)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
<u>ReturnParams</u> (inherited from <u>TOraDataSet</u>)	Used to return a new fields value to dataset after insert or update.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>RowsProcessed</u> (inherited from <u>TOraDataSet</u>)	Returns the number of rows processed by a query.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SequenceMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>Session</u> (inherited from <u>TOraDataSet</u>)	Used to specify the session in which dataset will be executed.

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQL](#) (inherited from [TCustomDADataset](#))

[SQLDelete](#) (inherited from [TCustomDADataset](#))

[SQLInsert](#) (inherited from [TCustomDADataset](#))

[SQLRefresh](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[SQLType](#) (inherited from [TOraDataSet](#))

[SQLUpdate](#) (inherited from [TCustomDADataset](#))

[StrictUpdate](#) (inherited from [TOraDataSet](#))

[UniDirectional](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateObject](#) (inherited from [TOraDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Builds an ORDER BY clause of a SELECT statement.

Used to provide a SQL statement that a query component executes when its Open method is called.

Used to specify a SQL statement that will be used when applying a deletion to a record.

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataset.RefreshRecord](#) procedure.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Used to get the typecode of the SQL statement being processed by Oracle database server.

Used to specify a SQL statement that will be used when applying an update to a dataset.

Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.

Used if an application does not need bidirectional access to records in the result set.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Used to point to an update object component which provides SQL statements that perform updates of the read-only datasets.

Used to indicate the update status for the current record when cached updates are enabled.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Used to check the status of the cached updates buffer.

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TOraStoredProc Class](#)
- [TOraStoredProc Class Members](#)

Executes a SQL statement on the server.

Class

[TOraStoredProc](#)

Syntax

```
procedure ExecProc;
```

Remarks

The ExecProc method is the same as [TCustomDADDataSet.Execute](#) method. It is included for compatibility with TStoredProc.

See Also

- [TCustomDADDataSet.Execute](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.24 Ora.TOrTimeStampField 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

```
TOrTimeStampField = class(TField);
```

Remarks

TOrTimeStampField provides access to Oracle timestamp fields. The TOrTimeStampField class supports three data types: TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE. According to this, the TOrTimeStampField.DataType property has three valid values ftTimeStamp, ftTimeStampTZ and ftTimeStampLTZ.

You can access actual timestamp value using the AsDateTime, AsString and AsOraTimeStamp properties.

Inheritance Hierarchy

```
TObject
  TOrTimeStampField
```

See Also

- [TOraTimeStamp](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.25 Ora.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.

PISqlTraceMode <> [].

Inheritance Hierarchy

```
TObject
  TOraTrace
```

© 1997-2012 Devart. All Rights Reserved.

[TOraTrace](#) class overview.

Properties

Name	Description
Enabled	Used to enable the trace.
MaxTraceFileSize	Used to limit the size of the SQL trace dump file.
PISqlTraceMode	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.
PISqlTraceComment	Provides a comment on the PL/SQL trace.
PISqlTraceLimit	Limits the amount of storage used in the database for the PL/SQL trace data.
PISqlTracePause	makes a pause in PL/SQL tracing.
PISqlTraceResume	Resumes PL/SQL trace.
PISqlTraceRunNumber	Returns a run number for the current PL/SQL trace.
PISqlTraceStart	Starts PL/SQL trace data collection.
PISqlTraceStop	Stops the PL/SQL trace.
SqlTraceStart	Starts the SQL trace.
SqlTraceStop	Stops the SQL trace.

© 1997-2012 Devart. All Rights Reserved.

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 whether 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.
PISqlTraceMode	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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to define whether 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
PISqlTraceComment	Provides a comment on the PL/SQL trace.
PISqlTraceLimit	Limits the amount of storage used in the database for the PL/SQL trace data.
PISqlTracePause	makes a pause in PL/SQL tracing.
PISqlTraceResume	Resumes PL/SQL trace.
PISqlTraceRunNumber	Returns a run number for the current PL/SQL trace.
PISqlTraceStart	Starts PL/SQL trace data collection.
PISqlTraceStop	Stops the PL/SQL trace.
SqlTraceStart	Starts the SQL trace.

[SqlTraceStop](#)

Stops the SQL trace.

See Also

- [TOraTrace Class](#)
 - [TOraTrace Class Members](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Resumes PL/SQL trace.

Class

[TOraTrace](#)

Syntax

```
procedure PlSqlTraceResume;
```

Remarks

Call the PlSqlTraceResume method to resume PL/SQL trace.

See Also

- [PLSqlTracePause](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
- [PISqlTraceStart](#)
- [PISqlTracePause](#)
- [PISqlTraceResume](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.17.1.26 Ora.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

TObject

[TCustomDAUpdateSQL](#)

TOraUpdateSQL

See Also

- [TOraDataSet.UpdateObject](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

17.17.1.27 Ora.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.

Inheritance Hierarchy

TObject
ToraXMLField

See Also

- [ToraXML](#)

© 1997-2012 Devart. All Rights Reserved.

[ToraXMLField](#) class overview.

Properties

Name	Description
AsXML	Used to provide access to a ToraXML object.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.17.2 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.
TPISqlTraceMode	Specifies the level of PL/SQL trace.
TSqlTraceMode	Specifies the level of SQL trace statistics level.

© 1997-2012 Devart. All Rights Reserved.

17.17.2.1 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.2.2 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.2.3 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.2.4 Ora.TPISqlTraceMode Set

Specifies the level of PL/SQL trace.

Unit

[Ora](#)

Syntax

```
TPISqlTraceMode = set of ( pmAllCalls, pmEnabledCalls,  
    pmAllExceptions, pmEnabledExceptions, pmAllSql, pmEnabledSql,  
    pmAllLines, pmEnabledLines);
```

© 1997-2012 Devart. All Rights Reserved.

17.17.2.5 Ora.TSqlTraceMode Set

Specifies the level of SQL trace statistics level.

Unit

[Ora](#)

Syntax

```
TSqlTraceMode = set of ( smBasicStatistics, smTypicalStatistics,  
    smAllStatistics, smBindVariables, smWaitEvents,  
    smTimedStatistics);
```

© 1997-2012 Devart. All Rights Reserved.

17.17.3 Enumerations

Enumerations in the **Ora** unit.

Enumerations

Name	Description
TCheckMode	Specifies the action to take when another user makes modifications to a record.
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.

© 1997-2012 Devart. All Rights Reserved.

17.17.3.1 Ora.TCheckMode Enumeration

Specifies the action to take when another user makes modifications to a record.

Unit

[Ora](#)

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.

© 1997-2012 Devart. All Rights Reserved.

17.17.3.2 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.3.3 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.3.4 Ora.TOraIsolationLevel Enumeration

Specifies the way the transactions containing database modifications are handled.

Unit

[Ora](#)

Syntax

```
TOrasolationLevel = (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.

© 1997-2012 Devart. All Rights Reserved.

17.17.3.5 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.3.6 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.4 Routines

Routines in the **Ora** unit.

Routines

Name	Description
AddWhere	Appends condition to WHERE clause.
DefaultSession	Call this function to get pointer to default session object.
DeleteWhere	Removes WHERE clause from a SELECT statement.
GetOrderBy	Returns fields from ORDER BY.
SessionByName	Returns pointer to session object by its name.
SetFieldList	Sets list of selecting fields.
SetGroupBy	Sets grouping fields.
SetOrderBy	Sets ordering fields.
SetSubscriptionPort	Sets change notification subscription port. Before setting the subscription port all opened sessions close and oci.dll unloads.
SetTableList	Sets list of tables.
SetWhere	Call this function to replace WHERE clause of a SELECT statement to Condition. You should pass SELECT statement by SQL.If condition is blank SetWhere removes WHERE clause.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.1 Ora.AddWhere Function

Appends condition to WHERE clause.

Unit

[Ora](#)

Syntax

```
function AddWhere(SQL: string; Condition: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Condition

Holds the condition that will be added to the WHERE clause.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.2 Ora.DefaultSession Function

Call this function to get pointer to default session object.

Unit

[Ora](#)

Syntax

```
function DefaultSession: TOraSession;
```

© 1997-2012 Devart. All Rights Reserved.

17.17.4.3 Ora.DeleteWhere Function

Removes WHERE clause from a SELECT statement.

Unit

[Ora](#)

Syntax

```
function DeleteWhere(SQL: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.4 Ora.GetOrderBy Function

Returns fields from ORDER BY.

Unit

[Ora](#)

Syntax

```
function GetOrderBy(SQL: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.5 Ora.SessionByName Function

Returns pointer to session object by its name.

Unit

[Ora](#)

Syntax

```
function SessionByName(Name: string): TOraSession;
```

Parameters

Name

Holds the pointer name.

Return Value

pointer to session object.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.6 Ora.SetFieldList Function

Sets list of selecting fields.

Unit

[Ora](#)

Syntax

```
function SetFieldList(SQL: string; Fields: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Fields

Holds the list of fields.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.7 Ora.SetGroupBy Function

Sets grouping fields.

Unit

[Ora](#)

Syntax

```
function SetGroupBy(SQL: string; Fields: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Fields

Holds the fields.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.8 Ora.SetOrderBy Function

Sets ordering fields.

Unit

[Ora](#)

Syntax

```
function SetOrderBy(SQL: string; Fields: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Fields

Holds the fields.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.9 Ora.SetSubscriptionPort Procedure

Sets change notification subscription port. Before setting the subscription port all opened sessions close and oci.dll unloads.

Unit

[Ora](#)

Syntax

```
procedure SetSubscriptionPort(Value: integer);
```

Parameters

Value

Holds the of change notification subscription port.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.10 Ora.SetTableList Function

Sets list of tables.

Unit

[Ora](#)

Syntax

```
function SetTableList(SQL: string; Tables: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Tables

Holds the tables.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.4.11 Ora.SetWhere Function

Call this function to replace WHERE clause of a SELECT statement to Condition. You should pass SELECT statement by SQL.

If condition is blank SetWhere removes WHERE clause.

Unit

[Ora](#)

Syntax

```
function SetWhere(SQL: string; Condition: string): string;
```

Parameters

SQL

Holds a SQL to be modified.

Condition

Holds the condition that will be added to the WHERE clause.

Return Value

modified SQL.

© 1997-2012 Devart. All Rights Reserved.

17.17.5 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.

© 1997-2012 Devart. All Rights Reserved.

17.17.5.1 Ora.DefSession Variable

Read this variable to get pointer to default session object. Same as DefaultSession function.

Unit

[Ora](#)

Syntax

```
DefSession: TOraSession;
```

© 1997-2012 Devart. All Rights Reserved.

17.17.5.2 Ora.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.

© 1997-2012 Devart. All Rights Reserved.

17.17.5.3 Ora.Sessions Variable

Holds pointers to all TOraSession objects of an application.

Unit

[Ora](#)

Syntax

```
Sessions: TSessionList;
```

© 1997-2012 Devart. All Rights Reserved.

17.17.5.4 Ora.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;
```

© 1997-2012 Devart. All Rights Reserved.

17.17.6 Constants

Constants in the **Ora** unit.

Constants

Name	Description
OdacVersion	Read this constant to get current version number for ODAC.

© 1997-2012 Devart. All Rights Reserved.

17.17.6.1 Ora.OdacVersion Constant

Read this constant to get current version number for ODAC.

Unit

[Ora](#)

Syntax

```
ODACVersion = '8.5.9';
```

© 1997-2012 Devart. All Rights Reserved.

17.18 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.

17.18.1 Classes

Classes in the **OraAlerter** unit.

Classes

Name	Description
TOraAlerter	A component allowing transferring messages between sessions or client applications.

© 1997-2012 Devart. All Rights Reserved.

17.18.1.1 OraAlerter.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

TObject

[TDAAlerter](#)

TOraAlerter

© 1997-2012 Devart. All Rights Reserved.

[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 TOraAlerter 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.

© 1997-2012 Devart. All Rights Reserved.

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.

[OnError](#) (inherited from [TDAAlerter](#))

Occurs if an exception occurs in waiting process

[SendEvent](#) (inherited from [TDAAlerter](#))

Sends an event with Name and content Message.

[Start](#) (inherited from [TDAAlerter](#))

Starts waiting process.

[Stop](#) (inherited from [TDAAlerter](#))

Stops waiting process.

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 TOraAlerter 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](#)

© 1997-2012 Devart. All Rights Reserved.

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

- M:Devart.Odac.TOraAlerter.SendEvent(System.String,System.String)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to define the kind of messages used by the TOraAlerter component.

Class

[TOraAlerter](#)

Syntax

```
property EventType: TEventType;
```

Remarks

Use EventType to specify what kind of messages is used by the TOraAlerter component.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.
Internal TOraSession object is represented by AlerterSession property.

See Also

- [TOraSession](#)
 - [AlerterSession](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Methods 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.
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.
OnError (inherited from TDAAlerter)	Occurs if an exception occurs in waiting process
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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 designed 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.

© 1997-2012 Devart. All Rights Reserved.

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 designed to work only with the Oracle DBMS PIPE communication package which is the case only if the [EventType](#) property has been set to etPipe.

© 1997-2012 Devart. All Rights Reserved.

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 designed to work only with the Oracle DBMS PIPE communication package. So it can be called only if [EventType](#) property has been set to etPipe.

© 1997-2012 Devart. All Rights Reserved.

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 designed 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.

© 1997-2012 Devart. All Rights Reserved.

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 designed 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.

© 1997-2012 Devart. All Rights Reserved.

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 designed to work only with the Oracle DBMS PIPE communication package which is the case only if [EventType](#) property has been set to etPipe.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Events 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.
OnError (inherited from TDAAlerter)	Occurs if an exception occurs in waiting process
SendEvent (inherited from TDAAlerter)	Sends an event with Name and content Message.
Start (inherited from TDAAlerter)	Starts waiting process.
Stop (inherited from TDAAlerter)	Stops 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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.18.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.18.2.1 OraAlerter.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.

© 1997-2012 Devart. All Rights Reserved.

17.18.2.2 OraAlerter.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.

© 1997-2012 Devart. All Rights Reserved.

17.18.3 Enumerations

Enumerations in the **OraAlerter** unit.

Enumerations

Name	Description
TEventType	Specifies the kind of messages used by the TOraAlerter component.

© 1997-2012 Devart. All Rights Reserved.

17.18.3.1 OraAlerter.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.

© 1997-2012 Devart. All Rights Reserved.

17.19 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 Queueing.
TOraQueueAdmin	A component for managing queues in a database.
TOraQueueTable	A component managing queue tables in a database.
TQueueAgent	A class representing 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 column 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.

17.19.1 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 representing 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.

© 1997-2012 Devart. All Rights Reserved.

17.19.1.1 OraAQ.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.

Inheritance Hierarchy

```
TObject
  TDequeueOptions
```

See Also

- [TOraQueue.Dequeue](#)
- [TOraQueue.DequeueArray](#)
- [TOraQueue.DequeueOptions](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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).

© 1997-2012 Devart. All Rights Reserved.

17.19.1.2 OraAQ.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.EnqueueArray](#) method.

Inheritance Hierarchy

TObject

TEnqueueOptions

See Also

- [TOraQueue.Enqueue](#)
- [TOraQueue.EnqueueArray](#)
- [TOraQueue.EnqueueOptions](#)

© 1997-2012 Devart. All Rights Reserved.

[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	Description is not available at the moment.
Transformation	Used to specify the transformation that will be applied before enqueueing a message.
Visibility	Used to specify the transactional behavior of the enqueue request.

© 1997-2012 Devart. All Rights Reserved.

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	Description is not available at the moment.
Transformation	Used to specify the transformation that will be applied before enqueueing a message.
Visibility	Used to specify the transactional behavior of the enqueue request.

See Also

- [TEnqueueOptions Class](#)
- [TEnqueueOptions Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Class

[TEnqueueOptions](#)

Syntax

```
property SequenceDeviation: TQueueSequenceDeviation default  
qsdNone;
```

Remarks

Use SequenceDeviation property to specify whether the enqueued message should be dequeued before other messages that are in the queue already.

See Also

- [RelativeMessageId](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to specify the transformation that will be applied before enqueueing a message.

Class

[TEnqueueOptions](#)

Syntax

```
property Transformation: string;
```

Remarks

Use the Transformation property to specify a transformation that will be applied before enqueueing a message. Use it with Oracle 10 and higher.

See Also

- [TDequeueOptions.Transformation](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.19.1.3 OraAQ.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.

Inheritance Hierarchy

TObject

TOraQueue

© 1997-2012 Devart. All Rights Reserved.

[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 enqueueing message properties.
EnqueueOptions	Used to set the default message enqueueing options.
PayloadArrayTypeNames	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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraQueue** class.

For a complete list of the **TOraQueue** class members, see the [TOraQueue Members](#) topic.

Public

Name	Description
PayloadArrayTypeNames	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 enqueueing message properties.

[EnqueueOptions](#)

Used to set the default message enqueueing 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](#)

© 1997-2012 Devart. All Rights Reserved.

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 property 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](#)
- [Ora.GetSubscriptionPort](#)
- [Ora.SetSubscriptionPort](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to set the default enqueueing message properties.

Class

[TOraQueue](#)

Syntax

```
property EnqueueMessageProperties: TQueueMessageProperties;
```

Remarks

Use the EnqueueMessageProperties property to set the default enqueueing message properties.

See Also

- [TQueueMessageProperties](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to set the default message enqueueing options.

Class

[TOraQueue](#)

Syntax

```
property EnqueueOptions: TEnqueueOptions;
```

Remarks

Use the EnqueueOptions property to set the default message enqueueing options.

See Also

- [TEnqueueOptions](#)
 - [DequeueOptions](#)
-

© 1997-2012 Devart. All Rights Reserved.

Contains the type name of associative array, VARRAY, or nested table of queue payload type.

Class

[TOraQueue](#)

Syntax

```
property PayloadArrayTypeNames: string;
```

Remarks

The PayloadArrayTypeNames 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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Dequeues messages.

Class

[TOraQueue](#)

Overload List

Name	Description
Dequeue(Payload: TOraObject; MessageProperties: TQueueMessageProperties; DequeueOptions: TDequeueOptions)	Dequeues a message as a TOraObject.
Dequeue(Message: TQueueMessage; DequeueOptions: TDequeueOptions)	Dequeues messages.
Dequeue(out Payload: TBytes; MessageProperties: TQueueMessageProperties; DequeueOptions: TDequeueOptions)	Dequeues a message as array of Byte.
Dequeue(out Payload: string ; MessageProperties: TQueueMessageProperties; DequeueOptions: TDequeueOptions)	Dequeues string messages.

© 1997-2012 Devart. All Rights Reserved.

Dequeues 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 dequeuing 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.

© 1997-2012 Devart. All Rights Reserved.

Dequeues messages.

Class

[TOraQueue](#)

Syntax

```
function Dequeue (Message: TQueueMessage; DequeueOptions:
TDequeueOptions = nil): TMessageId; overload
```

Parameters

Message

Holds the message content.

DequeueOptions

Holds the options for dequeuing messages.

Return Value

message ID as string.

© 1997-2012 Devart. All Rights Reserved.

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; EnqueueOptions: TEnqueueOptions)	Enqueueus a message as array of Byte.
Enqueue(const Payload: string; MessageProperties: TQueueMessageProperties; EnqueueOptions: TEnqueueOptions)	Enqueues string messages.

© 1997-2012 Devart. All Rights Reserved.

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 MessageProperties 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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.19.1.4 OraAQ.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.

Inheritance Hierarchy

TObject

TOraQueueAdmin

© 1997-2012 Devart. All Rights Reserved.

[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
<u>AddSubscriber</u>	Adds a subscriber to the queue.
<u>AlterComment</u>	Alters the user-defined queue description.
<u>AlterMaxRetries</u>	Alters the number of attempts to dequeue a message.
<u>AlterPropagationSchedule</u>	Alters parameters for a propagation schedule.
<u>AlterQueue</u>	Alters queue properties.
<u>AlterRetentionTime</u>	Alters the time in seconds during which a message remains in the queue after being dequeued.
<u>AlterRetryDelay</u>	Alters the delay time in seconds before the message, which failed to be dequeued, will be scheduled to processing again.
<u>AlterSubscriber</u>	Alters the rule and transformation properties of a queue subscriber.
<u>CreateQueue</u>	Creates a queue with the name, specified by the <u>TOraQueueAdmin.QueueName</u> property.
<u>DisablePropagationSchedule</u>	Disables a propagation schedule.
<u>DropQueue</u>	Drops the queue specified by the <u>TOraQueueAdmin.QueueName</u> property.
<u>EnablePropagationSchedule</u>	Enables a previously disabled propagation schedule.
<u>GetSubscribers</u>	Provides the list of queue subscribers.
<u>GrantQueuePrivilege</u>	Grants queue privilege to the grantee.
<u>ReadQueueProperties</u>	Reads the information about a queue specified by the <u>TOraQueueAdmin.QueueName</u> property from the database to a TOraQueueAdmin component.
<u>RemoveSubscriber</u>	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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#).

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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` method to schedule the propagation of messages from the queue to a destination identified by a specific database link.

See Also

- [UnschedulePropagation](#)
- [AlterPropagationSchedule](#)
- [EnablePropagationSchedule](#)

© 1997-2012 Devart. All Rights Reserved.

Enables dequeuing on a queue.

Class

[TOraQueueAdmin](#)

Syntax

```
procedure StartDequeue;
```

Remarks

Call the StartDequeue method to enable dequeuing on a queue.

See Also

- [StartQueue](#)
 - [StartEnqueue](#)
 - [StopDequeue](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Enables enqueueing, dequeuing, 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 dequeuing is enabled. False otherwise.

Remarks

Call the StopQueue method to enable enqueueing, dequeuing, or both on a queue.

See Also

- [StartEnqueue](#)
 - [StartDequeue](#)
 - [StopQueue](#)
-

© 1997-2012 Devart. All Rights Reserved.

Disables dequeuing 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 dequeuing on a queue. The Wait parameter specifies whether to wait for the completion of the currently active queue transactions.

See Also

- [StopQueue](#)
- [StopEnqueue](#)
- [StartDequeue](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Stops enqueueing, dequeuing, 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](#)

© 1997-2012 Devart. All Rights Reserved.

Unschedule 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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.19.1.5 OraAQ.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.

Inheritance Hierarchy

TObject

TOraQueueTable

See Also

- [TOraQueue, TOraQueueAdmin and TOraQueueTable Components](#)
- [TOraQueueAdmin](#)
- [TOraQueue](#)

© 1997-2012 Devart. All Rights Reserved.

[TOraQueueTable](#) class overview.

Properties

Name	Description
Comment	Used to get or set a user-specified description of the 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.

© 1997-2012 Devart. All Rights Reserved.

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 get or set a user-specified description of the 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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to get or set a user-specified description of the 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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.
-

© 1997-2012 Devart. All Rights Reserved.

Drops the queue table.

Class

[TOraQueueTable](#)

Syntax

```
procedure DropQueueTable(Force: boolean = False);
```

Parameters*Force*

If True, stops and drops 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.19.1.6 OraAQ.TQueueAgent Class

A class representing 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.

Inheritance Hierarchy

TObject

TQueueAgent

See Also

- [TQueueAgents](#)
- [TQueueMessageProperties.SenderId](#)
- [TOraQueue.Listen](#)
- [TOraQueueAdmin.AddSubscriber](#)
- [TOraQueueAdmin.RemoveSubscriber](#)
- [TOraQueueAdmin.AlterSubscriber](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.19.1.7 OraAQ.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.

Inheritance Hierarchy

TObject

TQueueAgents

See Also

- [TQueueAgent](#)
- [TQueueMessageProperties.RecipientList](#)
- [TOraQueue.Listen](#)
- [TOraQueueAdmin.GetSubscribers](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.19.1.8 OraAQ.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.

Inheritance Hierarchy

TObject

TQueueMessage

See Also

- [TOraQueue.Dequeue](#)
- [TOraQueue.DequeueArray](#)
- [TOraQueue.Enqueue](#)
- [TOraQueue.EnqueueArray](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.19.1.9 OraAQ.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.

Inheritance Hierarchy

TObject

TQueueMessageProperties

See Also

- [TQueueMessage.MessageProperties](#)
- [TOraQueue.EnqueueMessageProperties](#)
- [TOraQueue.OnMessage](#)
- [TOraQueue.Dequeue](#)
- [TOraQueue.DequeueArray](#)
- [TOraQueue.Enqueue](#)
- [TOraQueue.EnqueueArray](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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 dequeuing messages. qdmPersistentOrBuffered value can be set only when dequeuing a message.

See Also

- [TEnqueueOptions.DeliveryMode](#)
- [TDequeueOptions.DeliveryMode](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Contains the state of a message.

Class

[TQueueMessageProperties](#)

Syntax

```
property State: TQueueMessageState;
```

Remarks

The State property contains the state of a message.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.19.2 Types

Types in the **OraAQ** unit.

Types

Name	Description
TQueueMessageEvent	This type is used for the TOraQueue.OnMessage event.

© 1997-2012 Devart. All Rights Reserved.

17.19.2.1 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3 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 column 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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.1 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.2 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.3 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.4 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.5 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.6 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.7 OraAQ.TQueueSortList Enumeration

Specifies the column 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.
qslEnqueueTimePriority	The table will be sorted by both enq time and priority columns. 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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.8 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.19.3.9 OraAQ.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.

© 1997-2012 Devart. All Rights Reserved.

17.20 OraCall

Defines Oracle Call Interface routines.

Routines

Name	Description
DetectOCI	Searches OCI library and sets OCIDLL, OCIVersion, OCICallStyle variables.
FreeOCI	Removes OCI library from memory.
GetSubscriptionPort	Returns the value of change notification subscription port.
InitOCI	Initializes OCI environment.
LoadedOCI	Returns True when OCI library has been loaded already.
LoadOCI	Loads OCI library to memory.
OraError	This unit contains the EOraError exception class.

Variables

Name	Description
OCICallStyle	Use OCICallStyle to set the set of routines to be used. Possible value is OCI73 and OCI80.
OCIDLL	Use OCIDLL to read and assign a name of OCI library file. OCIDLL variable is assigned automatically after OCI initialization.
OCIThreaded	Allows to initialize OCI environment in threaded mode. The default value is True.
OCIVersion	Read OCIVersion to learn the version of OCI.
OCIVersionSt	Read OCIVersionSt to learn version of OCI as string.
OracleErrorMaxLength	Allows to set the maximum length of returned Oracle error message. The default value is 512.

17.20.1 Routines

Routines in the **OraCall** unit.

Routines

Name	Description
DetectOCI	Searches OCI library and sets OCIDLL, OCIVersion, OCICallStyle variables.
FreeOCI	Removes OCI library from memory.
GetSubscriptionPort	Returns the value of change notification subscription port.
InitOCI	Initializes OCI environment.
LoadedOCI	Returns True when OCI library has been loaded already.
LoadOCI	Loads OCI library to memory.
OraError	This unit contains the EOraError exception class.

© 1997-2012 Devart. All Rights Reserved.

17.20.1.1 OraCall.DetectOCI Procedure

Searches OCI library and sets OCIDLL, OCIVersion, OCICallStyle variables.

Unit

[OraCall](#)

Syntax

```
procedure DetectOCI;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.1.2 OraCall.FreeOCI Procedure

Removes OCI library from memory.

Unit

[OraCall](#)

Syntax

```
procedure FreeOCI;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.1.3 OraCall.GetSubscriptionPort Function

Returns the value of change notification subscription port.

Unit

[OraCall](#)

Syntax

```
function GetSubscriptionPort: integer;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.1.4 OraCall.InitOCI Procedure

Initializes OCI environment.

Unit

[OraCall](#)

Syntax

```
procedure InitOCI;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.1.5 OraCall.LoadedOCI Function

Returns True when OCI library has been loaded already.

Unit

[OraCall](#)

Syntax

```
function LoadedOCI: boolean;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.1.6 OraCall.LoadOCI Procedure

Loads OCI library to memory.

Unit

[OraCall](#)

Syntax

```
procedure LoadOCI;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.1.7 OraCall.OraError Procedure

This unit contains the EOraError exception class.

Classes

Name	Description
EOraError	Raised when a component detects Oracle error.

© 1997-2012 Devart. All Rights Reserved.

17.20.2 Variables

Variables in the **OraCall** unit.

Variables

Name	Description
OCICallStyle	Use OCICallStyle to set the set of routines to be used. Possible value is OCI73 and OCI80.
OCIDLL	Use OCIDLL to read and assign a name of OCI library file. OCIDLL variable is assigned automatically after OCI initialization.
OCIThreaded	Allows to initialize OCI environment in threaded mode. The default value is True.
OCIVersion	Read OCIVersion to learn the version of OCI.
OCIVersionSt	Read OCIVersionSt to learn version of OCI as string.
OracleErrorMaxLength	Allows to set the maximum length of returned Oracle error message. The default value is 512.

© 1997-2012 Devart. All Rights Reserved.

17.20.2.1 OraCall.OCICallStyle Variable

Use OCICallStyle to set the set of routines to be used.
Possible value is OCI73 and OCI80.

Unit

[OraCall](#)

Syntax

```
OCICallStyle: TOCICallStyle;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.2.2 OraCall.OCIDLL Variable

Use OCIDLL to read and assign a name of OCI library file. OCIDLL variable is assigned automatically after OCI initialization.

Unit

[OraCall](#)

Syntax

```
OCIDLL: string;
```

Example

For example:

```
OCIDLL := 'C:\ORANT\BIN\OCI.DLL'
```

© 1997-2012 Devart. All Rights Reserved.

17.20.2.3 OraCall.OCIThreaded Variable

Allows to initialize OCI environment in threaded mode.
The default value is True.

Unit

[OraCall](#)

Syntax

```
OCIThreaded: boolean;
```

© 1997-2012 Devart. All Rights Reserved.

17.20.2.4 OraCall.OCIVersion Variable

Read OCIVersion to learn the version of OCI.

Unit

[OraCall](#)

Syntax

```
OCIVersion: word;
```

Example

7340 value corresponds to '7.3.4.0.0'.

© 1997-2012 Devart. All Rights Reserved.

17.20.2.5 OraCall.OCIVersionSt Variable

Read OCIVersionSt to learn version of OCI as string.

Unit

[OraCall](#)

Syntax

```
OCIVersionSt: string;
```

Example

For example:
'7.3.4.0.0'

© 1997-2012 Devart. All Rights Reserved.

17.20.2.6 OraCall.OracleErrorMaxLength Variable

Allows to set the maximum length of returned Oracle error message.
The default value is 512.

Unit

[OraCall](#)

Syntax

```
OracleErrorMaxLength: integer;
```

© 1997-2012 Devart. All Rights Reserved.

17.21 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
TOraCursor	A class holding the internal Oracle Call Interface data CDA for Oracle 7 and the OCISmt 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
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 IntegerPrecision and less than or equal to LargeIntPrecision as dtLargeInt.

17.21.1 Classes

Classes in the **OraClasses** unit.

Classes

Name	Description
TOraCursor	A class holding the internal Oracle Call Interface data CDA for Oracle 7 and the OCISstmt 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.

© 1997-2012 Devart. All Rights Reserved.

17.21.1.1 OraClasses.TOraCursor Class

A class holding the internal Oracle Call Interface data CDA for Oracle 7 and the OCISstmt 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 OCISstmt descriptor for Oracle 8.

Inheritance Hierarchy

```
TOject
  TSharedObject
    TCRCursor
      TOraCursor
```

See Also

- [TCursorField](#)
- [TOraDataSet.Cursor](#)
- [TOraParam.AsCursor](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

[OCISmt](#)

Used to return the OCISmt handler.

[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.
AllocCursor	Allocates CDA for Oracle 7 and OCISmt 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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
CDA	Used to return CDA.
OCICallStyle	Used to get or set the Oracle Client version for a subsequent cursor operations.
OCISmt	Used to return the OCISmt handler.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

- [TOraCursor Class](#)
- [TOraCursor Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to return the OCISmt handler.

Class

[TOraCursor](#)

Syntax

```
property OCISmt: pOCISmt;
```

Remarks

Use the OCISmt property to return the OCISmt handler. You can use this property for Oracle 8.

© 1997-2012 Devart. All Rights Reserved.

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 OCISmt handler for Oracle 8.
CanFetch	Verifies if the cursor state permits data fetching.
FreeCursor	Releases the cursor data.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

- [TOraCursor Class](#)
- [TOraCursor Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Allocates CDA for Oracle 7 and OCISmt handler for Oracle 8.

Class

[TOraCursor](#)

Syntax

```
procedure AllocCursor(StatementCache: boolean = False);
```

Parameters

StatementCache

Specifies whether OCI statement caching is used.

Remarks

Call the AllocCursor procedure to allocate CDA for Oracle 7 and OCISmt handler for Oracle 8.

See Also

- [FreeCursor](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Releases the cursor data.

Class

[TOraCursor](#)

Syntax

```
procedure FreeCursor;
```

Remarks

Call the FreeCursor method to release the cursor data.

See Also

- [AllocCursor](#)

© 1997-2012 Devart. All Rights Reserved.

17.21.1.2 OraClasses.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

```

TObject
  TSharedObject
    TBlob
      TCompressedBlob
        TOraLob
          ToraFile
  
```

See Also

- [TOraLob](#)
- [TBFileField](#)
- [ToraParam.AsBFile](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraFile** class.

For a complete list of the **TOraFile** class members, see the [TOraFile Members](#) topic.

Public

Name	Description
------	-------------

<u>AddRef</u> (inherited from <u>TSharedObject</u>)	Increments the reference count for the number of references dependent on the TSharedObject object.
<u>AllocLob</u> (inherited from <u>TOraLob</u>)	Allocates and initializes the LOB locator.
<u>Assign</u> (inherited from <u>TBlob</u>)	Sets BLOB value from another TBlob object.
<u>AsString</u> (inherited from <u>TBlob</u>)	Used to manipulate BLOB value as string.
<u>AsWideString</u> (inherited from <u>TBlob</u>)	Used to manipulate BLOB value as Unicode string.
<u>Cached</u> (inherited from <u>TOraLob</u>)	Used to indicate where the LOB data is.
<u>Clear</u> (inherited from <u>TBlob</u>)	Deletes the current value in TBlob object.
<u>CreateTemporary</u> (inherited from <u>TOraLob</u>)	Allocates and initializes a temporary LOB locator of the specified type.
<u>DisableBuffering</u> (inherited from <u>TOraLob</u>)	Disables the LOB buffering for the LOB locator.
<u>EnableBuffering</u> (inherited from <u>TOraLob</u>)	Enables the LOB buffering for the LOB locator.
<u>FileDir</u>	Determines the directory alias of where the file associated with the BFile field is stored.
<u>FileName</u>	Used to determine the name of a file associated with the BFile field.
<u>FreeLob</u> (inherited from <u>TOraLob</u>)	Releases the LOB locator descriptor.
<u>FreeTemporary</u> (inherited from <u>TOraLob</u>)	Frees a temporary LOB.
<u>Init</u> (inherited from <u>TOraLob</u>)	Initializes the OCILobLocator handle.
<u>IsInit</u> (inherited from <u>TOraLob</u>)	Verifies if the LOB locator is initialized.
<u>IsTemporary</u> (inherited from <u>TOraLob</u>)	Indicates whether the LOB is temporary.
<u>IsUnicode</u> (inherited from <u>TBlob</u>)	Gives choice of making TBlob store and process data in Unicode format or not.
<u>LengthLob</u> (inherited from <u>TOraLob</u>)	Returns the number of bytes contained in the LOB object.
<u>LoadFromFile</u> (inherited from <u>TBlob</u>)	Loads the contents of a file into a TBlob object.
<u>LoadFromStream</u> (inherited from <u>TBlob</u>)	Copies the contents of a stream into the TBlob object.
<u>OCILobLocator</u> (inherited from <u>TOraLob</u>)	Used to get or set the OCILobLocator handle.
<u>OCILobLocatorPtr</u> (inherited from <u>TOraLob</u>)	Used to retrieve the LOB value type locator.
<u>OCISvcCtx</u> (inherited from <u>TOraLob</u>)	Used to assign a service context handle.
<u>Read</u> (inherited from <u>TBlob</u>)	Acquires a raw sequence of bytes from the data stored in TBlob.
<u>ReadLob</u> (inherited from <u>TOraLob</u>)	Reads the LOB content from the server.
<u>RefCount</u> (inherited from <u>TSharedObject</u>)	Used to return the count of reference to a TSharedObject object.
<u>Release</u> (inherited from <u>TSharedObject</u>)	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.

[Size](#) (inherited from [TBlob](#))

Used to learn the size of the TBlob value in bytes.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TOraFile** class.

For a complete list of the **TOraFile** class members, see the [TOraFile Members](#) topic.

Public

Name	Description
------	-------------

<u>AddRef</u> (inherited from <u>TSharedObject</u>)	Increments the reference count for the number of references dependent on the TSharedObject object.
<u>AllocLob</u> (inherited from <u>TOraLob</u>)	Allocates and initializes the LOB locator.
<u>Assign</u> (inherited from <u>TBlob</u>)	Sets BLOB value from another TBlob object.
<u>AsString</u> (inherited from <u>TBlob</u>)	Used to manipulate BLOB value as string.
<u>AsWideString</u> (inherited from <u>TBlob</u>)	Used to manipulate BLOB value as Unicode string.
<u>Cached</u> (inherited from <u>TOraLob</u>)	Used to indicate where the LOB data is.
<u>Clear</u> (inherited from <u>TBlob</u>)	Deletes the current value in TBlob object.
<u>Close</u>	Closes a file opened by Open.
<u>CreateTemporary</u> (inherited from <u>TOraLob</u>)	Allocates and initializes a temporary LOB locator of the specified type.
<u>DisableBuffering</u> (inherited from <u>TOraLob</u>)	Disables the LOB buffering for the LOB locator.
<u>EnableBuffering</u> (inherited from <u>TOraLob</u>)	Enables the LOB buffering for the LOB locator.
<u>Exists</u>	Determines whether a file associated with the BFile field exists.
<u>FreeLob</u> (inherited from <u>TOraLob</u>)	Releases the LOB locator descriptor.
<u>FreeTemporary</u> (inherited from <u>TOraLob</u>)	Frees a temporary LOB.
<u>Init</u> (inherited from <u>TOraLob</u>)	Initializes the OCILobLocator handle.
<u>IsInit</u> (inherited from <u>TOraLob</u>)	Verifies if the LOB locator is initialized.
<u>IsOpen</u>	Determines whether the file associated with the BFile field is opened.
<u>IsTemporary</u> (inherited from <u>TOraLob</u>)	Indicates whether the LOB is temporary.
<u>IsUnicode</u> (inherited from <u>TBlob</u>)	Gives choice of making TBlob store and process data in Unicode format or not.
<u>LengthLob</u> (inherited from <u>TOraLob</u>)	Returns the number of bytes contained in the LOB object.
<u>LoadFromFile</u> (inherited from <u>TBlob</u>)	Loads the contents of a file into a TBlob object.
<u>LoadFromStream</u> (inherited from <u>TBlob</u>)	Copies the contents of a stream into the TBlob object.
<u>OCILobLocator</u> (inherited from <u>TOraLob</u>)	Used to get or set the OCILobLocator handle.
<u>OCILobLocatorPtr</u> (inherited from <u>TOraLob</u>)	Used to retrieve the LOB value type locator.
<u>OCISvcCtx</u> (inherited from <u>TOraLob</u>)	Used to assign a service context handle.
<u>Open</u>	Opens the file associated with the BFile field.
<u>Read</u> (inherited from <u>TBlob</u>)	Acquires a raw sequence of bytes from the data stored in TBlob.
<u>ReadLob</u> (inherited from <u>TOraLob</u>)	Reads the LOB content from the server.

[RefCount](#) (inherited from [TSharedObject](#))

[Refresh](#)

[Release](#) (inherited from [TSharedObject](#))

[SaveToFile](#) (inherited from [TBlob](#))

[SaveToStream](#) (inherited from [TBlob](#))

[Size](#) (inherited from [TBlob](#))

[Truncate](#) (inherited from [TBlob](#))

[Write](#) (inherited from [TBlob](#))

[WriteLob](#) (inherited from [TOraLob](#))

Used to return the count of reference to a TSharedObject object.

Reloads a file associated the BFile field opened by Open.

Decrements the reference count.

Saves the contents of the TBlob object to a file.

Copies the contents of a TBlob object to a stream.

Used to learn the size of the TBlob value in bytes.

Sets new TBlob size and discards all data over it.

Stores a raw sequence of bytes into a TBlob object.

Writes a LOB value to the server.

See Also

- [TOraFile Class](#)
- [TOraFile Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.21.1.3 OraClasses.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

TObject

[TSharedObject](#)

ToraInterval

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraInterval** class.

For a complete list of the **TOraInterval** class members, see the [TOraInterval Members](#) topic.

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
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.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

- [TOraInterval Class](#)
- [TOraInterval Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to get the OCIInterval type locator.

Class

[TOraInterval](#)

Syntax

```
property OCIIntervalPtr: ppOCIInterval;
```

Remarks

Use the OCIIntervalPtr property to get the OCIInterval type locator.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TOraInterval** class.

For a complete list of the **TOraInterval** class members, see the [TOraInterval Members](#) topic.

Public

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.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.21.1.4 OraClasses.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

TObject

[TSharedObject](#)

[TBlob](#)

[TCompressedBlob](#)

TOraLob

See Also

- [TBlob](#)
- [TOraDataSet.GetLob](#)

- [TOraParam.AsBLOBLocator](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
Assign (inherited from TBlob)	Sets BLOB value from another TBlob object.
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.
Clear (inherited from TBlob)	Deletes the current value in TBlob object.
IsUnicode (inherited from TBlob)	Gives choice of making TBlob store and process data in Unicode format or not.
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.
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.
Read (inherited from TBlob)	Acquires a raw sequence of bytes from the data stored in TBlob.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
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.

[Size](#) (inherited from [TBlob](#))

Used to learn the size of the TBlob value in bytes.

[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.

See Also

- [TOraLob Class](#)
- [TOraLob Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to assign a service context handle.

Class

[TOraLob](#)

Syntax

```
property OCISvcCtx: pOCISvcCtx;
```

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
AsString (inherited from TBlob)	Used to manipulate BLOB value as string.
AsWideString (inherited from TBlob)	Used to manipulate BLOB value as Unicode string.
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.
IsUnicode (inherited from TBlob)	Gives choice of making TBlob store and process data in Unicode format or not.
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.

[RefCount](#) (inherited from [TSharedObject](#))

Used to return the count of reference to a TSharedObject object.

[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.

[Size](#) (inherited from [TBlob](#))

Used to learn the size of the TBlob value in bytes.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

Allocates and initializes the LOB locator.

Class

[TOralob](#)

Syntax

```
procedure AllocLob; virtual;
```

Remarks

Call the AllocLob method to allocate and initialize the LOB locator.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Releases the LOB locator descriptor.

Class

[TOraLob](#)

Syntax

```
procedure FreeLob;
```

Remarks

Call the FreeLob method to release the LOB locator descriptor.

© 1997-2012 Devart. All Rights Reserved.

Frees a temporary LOB.

Class

[TOraLob](#)

Syntax

```
procedure FreeTemporary;
```

Remarks

Use the FreeTemporary method to free a temporary LOB.

See Also

- [FreeLob](#)

© 1997-2012 Devart. All Rights Reserved.

Initializes the OCILobLocator handle.

Class

[TOraLob](#)

Syntax

```
procedure Init;
```

Remarks

Call the Init method to initialize the OCILobLocator handle.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Reads the LOB content from the server.

Class

[TOraLob](#)

Syntax

```
procedure ReadLob(var SharedPiece: PPieceHeader); 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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.21.1.5 OraClasses.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

TObject

[TSharedObject](#)

TOranumber

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
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.

[Release](#) (inherited from [TSharedObject](#))

Decrements the reference count.

See Also

- [TOraNumber Class](#)
 - [TOraNumber Class Members](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to get and set the number value as integer.

Class

[TOraNumber](#)

Syntax

```
property AsInteger: integer;
```

Remarks

Use the AsInteger property to get and set the number value as integer.
Reading AsInteger property when [IsNull](#) is True returns 0.

See Also

- [IsNull](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

- [TOranumber Class](#)
- [TOranumber Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.21.1.6 OraClasses.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

TObject

[TSharedObject](#)

ToraTimeStamp

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraTimeStamp** class.

For a complete list of the **TOraTimeStamp** class members, see the [TOraTimeStamp Members](#) topic.

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
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.

[Release](#) (inherited from [TSharedObject](#))

Decrements the reference count.

[TimeZone](#)

Used to provide a time zone name portion of the datetime value.

See Also

- [TOraTimeStamp Class](#)
- [TOraTimeStamp Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to provide the OCIDateTime type locator.

Class

[TOraTimeStamp](#)

Syntax

```
property OCIDateTimePtr: ppOCIDateTime;
```

Remarks

Use the OCIDateTimePtr property to get the OCIDateTime type locator.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TOraTimeStamp** class.

For a complete list of the **TOraTimeStamp** class members, see the [TOraTimeStamp Members](#) topic.

Public

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.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.21.2 Enumerations

Enumerations in the **OraClasses** unit.

Enumerations

Name	Description
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.

© 1997-2012 Devart. All Rights Reserved.

17.21.2.1 OraClasses.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.

© 1997-2012 Devart. All Rights Reserved.

17.21.2.2 OraClasses.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.

© 1997-2012 Devart. All Rights Reserved.

17.21.2.3 OraClasses.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.

17.21.3 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 IntegerPrecision and less than or equal to LargeIntPrecision as dtLargeInt.

© 1997-2012 Devart. All Rights Reserved.

17.21.3.1 OraClasses.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;
```

© 1997-2012 Devart. All Rights Reserved.

17.21.3.2 OraClasses.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;
```

© 1997-2012 Devart. All Rights Reserved.

17.21.3.3 OraClasses.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;
```

© 1997-2012 Devart. All Rights Reserved.

17.22 OraConnectionPool

This unit contains the TOraConnectionPoolManager class for managing connection pool.

Enumerations

Name	Description
TOraPoolingType	Specifies the pool type.

© 1997-2012 Devart. All Rights Reserved.

17.22.1 Enumerations

Enumerations in the **OraConnectionPool** unit.

Enumerations

Name	Description
TOraPoolingType	Specifies the pool type.

© 1997-2012 Devart. All Rights Reserved.

17.22.1.1 OraConnectionPool.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.

© 1997-2012 Devart. All Rights Reserved.

17.23 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.

17.23.1 Classes

Classes in the **OraErrHand** unit.

Classes

Name	Description
TOraErrorHandler	A component allowing translating of error messages.

© 1997-2012 Devart. All Rights Reserved.

17.23.1.1 OraErrHand.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.

Inheritance Hierarchy

TObject

TOraErrorHandler

See Also

- [EOraError](#)

© 1997-2012 Devart. All Rights Reserved.

[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

[OnError](#)

Description

Occurs if an appropriate error can not be found in the error table or Active is False.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraErrorHandler** class.

For a complete list of the **TOraErrorHandler** class members, see the [TOraErrorHandler Members](#) topic.

Published

Name

[Active](#)

Description

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.23.2 Types

Types in the **OraErrHand** unit.

Types

Name	Description
TOnOraErrorEvent	This type is used for the TOraErrorHandler.OnError event.

© 1997-2012 Devart. All Rights Reserved.

17.23.2.1 OraErrHand.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).

© 1997-2012 Devart. All Rights Reserved.

17.24 OraError

This unit contains the EOraError exception class.

Classes

Name	Description
EOraError	Raised when a component detects Oracle error.

© 1997-2012 Devart. All Rights Reserved.

17.24.1 Classes

Classes in the **OraError** unit.

Classes

Name	Description
EOraError	Raised when a component detects Oracle error.

© 1997-2012 Devart. All Rights Reserved.

17.24.1.1 OraError.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

TObject
[EDAError](#)
EOraError

See Also

- [TOraErrorHandler](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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 [EDAEError](#))

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.25 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
TDPErrEvent	This type is used for the TOraLoader.OnError event.
TDPCGetColumnDataEvent	This type is used for the TOraLoader.OnGetColumnData event.
TDPPutDataEvent	This type is used for the TOraLoader.OnPutData event.

Enumerations

Name	Description
TDPErrAction	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.

17.25.1 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.

© 1997-2012 Devart. All Rights Reserved.

17.25.1.1 OraLoader.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

TObject

[TDAColumn](#)

TDPColumn

See Also

- [TOraLoader](#)
- [TQueueAgents](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.25.1.2 OraLoader.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 [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

TObject
[TDALoader](#)
TOraLoader

See Also

- [TOraLoader Component](#)
- [TDALoader](#)
- [TQueueAgents](#)
- [TOraSession](#)

© 1997-2012 Devart. All Rights Reserved.

[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)	Used to specify TCustomDAConnection in which TDALoader will be executed.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOracleLoader** class.

For a complete list of the **TOracleLoader** class members, see the [TOracleLoader 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)	Used to specify TCustomDAConnection in which TDALoader will be executed.
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.
OnGetColumnData (inherited from TDALoader)	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 (inherited from TDALoader)	Occurs when putting loading data by rows is needed.
PutColumnData (inherited from TDALoader)	Overloaded. Puts the value of individual columns.
TableName (inherited from TDALoader)	Used to specify the name of the table to which data will be loaded.

Published

Name	Description
LoadMode	Used to specify which access mode to use for a TOracleLoader object when a database table is being modified.
Session	Used to specify the session in which TOracleLoader will be executed.

See Also

- [TOracleLoader Class](#)
- [TOracleLoader Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to specify which access mode to use for a TOracleLoader object when a database table is being modified.

Class

[TOracleLoader](#)

Syntax

property LoadMode: [TLoadMode](#) **default** lmDirect;

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Events 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)	Used to specify TCustomDAConnection in which TDALoader will be executed.
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.
OnProgress (inherited from TDALoader)	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
PutColumnData (inherited from TDALoader)	Overloaded. Puts the value of individual columns.
TableName (inherited from TDALoader)	Used to specify the name of the table to which data will be loaded.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Occurs when processing errors that are raised during loading is needed.

Class

[TOraLoader](#)

Syntax

property OnError: [TDPErrrorEvent](#);

Remarks

Write the OnError event handler to process errors that occur during loading. Handler is used only if [LoadMode](#) = ImDirect, 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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

17.25.2 Types

Types in the **OraLoader** unit.

Types

Name	Description
TDPErrrorEvent	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.

© 1997-2012 Devart. All Rights Reserved.

17.25.2.1 OraLoader.TDPErrrorEvent Procedure Reference

This type is used for the [TOraLoader.OnError](#) event.

Unit

[OraLoader](#)

Syntax

```
TDPErrrorEvent = procedure (Sender: TOraLoader; E: Exception; Col: integer; Row: integer; var Action: TDPErrrorAction) 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.

© 1997-2012 Devart. All Rights Reserved.

17.25.2.2 OraLoader.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.

© 1997-2012 Devart. All Rights Reserved.

17.25.2.3 OraLoader.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.

© 1997-2012 Devart. All Rights Reserved.

17.25.3 Enumerations

Enumerations in the **OraLoader** unit.

Enumerations

Name	Description
TDPErrAction	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.

© 1997-2012 Devart. All Rights Reserved.

17.25.3.1 OraLoader.TDPErrAction Enumeration

Specifies the action for TOraLoader to take to process errors that occur during loading.

Unit

[OraLoader](#)

Syntax

```
TDPErrAction = (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.

© 1997-2012 Devart. All Rights Reserved.

17.25.3.2 OraLoader.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.

© 1997-2012 Devart. All Rights Reserved.

17.26 OraObjects

This unit contains classes for Oracle OBJECT, ARRAY, TABLE and XMLTYPE data types.

Classes

Name	Description
<u>TOraArray</u>	A class representing the value of the Oracle array data type.
<u>TOraNestTable</u>	A class representing a value of the Oracle nested table data type.
<u>TOraObject</u>	A class representing the Oracle object data type value.
<u>TOraRef</u>	A class representing the Oracle reference data type value.
<u>TOraType</u>	A class holding information about Oracle type required for TOraObject objects.
<u>TOraXML</u>	A class representing a value of the Oracle SYS.XMLTYPE type.

17.26.1 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.

© 1997-2012 Devart. All Rights Reserved.

17.26.1.1 OraObjects.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

```
TObject
  TSharedObject
    TDBObject
      TOraObject
        TOraArray
```

See Also

- [TOraObject](#)
 - [TOraDataSet.GetArray](#)
 - [TOraParam.AsArray](#)
-

© 1997-2012 Devart. All Rights Reserved.

[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.
AttrAsOCISString (inherited from TOraObject)	Used to read the attribute's data value as OCISString, or to assign an OCISString 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.
ItemAsDateTime	Used to read the value of the array's data into a TDateTime object or variable, or to assign a TDateTime value to the contents of the array's value.
ItemAsFloat	Used to read the value of the array's data into a Double, or to assign a Double value to the contents of the array's item.
ItemAsInteger	Used to read the value of the Item's data into an integer, or to assign an integer value to the contents of the item.
ItemAsObject	Used to read the value of the Item's data into a TOraObject , or to assign a TOraObject value to the contents of the item.

[ItemAsOCIString](#)

Used to get the value of the Item's data as a generic OCI string pointer, or to assign the OCI string value to the content of the item.

[ItemAsString](#)

Used to read the value of the Item's data into a string, or to assign a string value to the content of the item.

[ItemExists](#)

Indicates whether the item with Index exists.

[ItemIsNull](#)

Indicates whether the item contains a value.

[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 item to the end of an array.
Assign (inherited from TOraObject)	Copies properties or other attributes from another object.
Clear	Removes all items 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 item to the Index position.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
Assign (inherited from ToraObject)	Copies properties or other attributes from another object.
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.
AttrAsOCString (inherited from ToraObject)	Used to read the attribute's data value as OCString, or to assign an OCString 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.
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.
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.
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.
IsNull (inherited from TOraObject)	Used to verify if an object is empty.
ItemAsDateTime	Used to read the value of the array's data into a TDateTime object or variable, or to assign a TDateTime value to the contents of the array's value.
ItemAsFloat	Used to read the value of the array's data into a Double, or to assign a Double value to the contents of the array's item.
ItemAsInteger	Used to read the value of the Item's data into an integer, or to assign an integer value to the contents of the item.
ItemAsObject	Used to read the value of the Item's data into a TOraObject , or to assign a TOraObject value to the contents of the item.
ItemAsOCIStrng	Used to get the value of the Item's data as a generic OCI string pointer, or to assign the OCI string value to the content of the item.
ItemAsString	Used to read the value of the Item's data into a string, or to assign a string value to the content of the item.
ItemExists	Indicates whether the item with Index exists.
ItemIsNull	Indicates whether the item contains a value.
ItemType	Indicates the type of an array element.
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.
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.
Refresh (inherited from TOraObject)	Retrieves the latest database image for the object.
Release (inherited from TSharedObject)	Decrements the reference count.
Size	Holds the size of an array.
Unmark (inherited from TOraObject)	Marks an object as not being dirty.

See Also

- [TOraArray Class](#)
- [TOraArray Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to read the value of the array's data into a TDateTime object or variable, or to assign a TDateTime value to the contents of the array's value.

Class

[TOraArray](#)

Syntax

```
property ItemAsDateTime[Index: integer]: TDateTime;
```

Parameters

Index

Holds the index. ???

Remarks

Use the ItemAsDateTime property to read the value of the array's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the array's value.

© 1997-2012 Devart. All Rights Reserved.

Used to read the value of the array's data into a Double, or to assign a Double value to the contents of the array's item.

Class

[TOraArray](#)

Syntax

```
property ItemAsFloat[Index: integer]: double;
```

Parameters

Index

Holds the index of the array item.

Remarks

Use the ItemAsFloat property to read the value of the array's data into a Double, or to assign a Double value to the contents of the array's item.

© 1997-2012 Devart. All Rights Reserved.

Used to read the value of the Item's data into an integer, or to assign an integer value to the contents of the item.

Class

[TOraArray](#)

Syntax

```
property ItemAsInteger[Index: integer]: integer;
```

Parameters

Index

Holds the index of the item.

Remarks

Use the ItemAsInteger property to read the value of the Item's data into an integer, or to assign an integer value to the contents of the item.

© 1997-2012 Devart. All Rights Reserved.

Used to read the value of the Item's data into a [TOraObject](#), or to assign a TOraObject value to the contents of the item.

Class

[TOraArray](#)

Syntax

```
property ItemAsObject[Index: integer]: TOraObject;
```

Parameters

Index

Holds the index of the item.

Remarks

Use the ItemAsObject property to read the value of the Item's data into an [TOraObject](#), or to assign an TOraObject value to the contents of the item.

See Also

- [TOraObject](#)
-

© 1997-2012 Devart. All Rights Reserved.

Used to get the value of the Item's data as a generic OCI string pointer, or to assign the OCI string value to the content of the item.

Class

[TOraArray](#)

Syntax

```
property ItemAsOCIStrng[Index: integer]: pOCIStrng;
```

Parameters

Index

Holds the index of the item.

Remarks

Use the ItemAsOCIStrng property to get the value of the Item's data as a generic OCI string pointer, or to assign the OCI string value to the content of the item.

© 1997-2012 Devart. All Rights Reserved.

Used to read the value of the Item's data into a string, or to assign a string value to the content of the item.

Class

[TOraArray](#)

Syntax

```
property ItemAsString[Index: integer]: string;
```

Parameters

Index

Holds the index of the item.

Remarks

Use ItemAsString property to read the value of the Item's data into a string, or to assign a string value to the content of the item.

© 1997-2012 Devart. All Rights Reserved.

Indicates whether the item with Index exists.

Class

[TOraArray](#)

Syntax

```
property ItemExists[Index: integer]: boolean;
```

Parameters

Index

Holds the index of the item.

Remarks

Check ItemExists property to learn whether the item with Index exists.

See Also

- [Size](#)
-

© 1997-2012 Devart. All Rights Reserved.

Indicates whether the item contains a value.

Class

[TOraArray](#)

Syntax

```
property ItemIsNull[Index: integer]: boolean;
```

Parameters

Index

Holds the index of the item.

Remarks

Use IsNull to determine if the Item contains a value. If IsNull is True, the Item is blank. If IsNull is False, the Item has a value.

© 1997-2012 Devart. All Rights Reserved.

Indicates the type of an array element.

Class

[TOraArray](#)

Syntax

```
property ItemType: word;
```

Remarks

Read the ItemType property to return the type of an array element.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 item to the end of an array.
Assign (inherited from TOraObject)	Copies properties or other attributes from another object.
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.
AttrAsOCIStrng (inherited from TOraObject)	Used to read the attribute's data value as OCIStrng, or to assign an OCIStrng 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.
Clear	Removes all items 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.
Indicator (inherited from TOraObject)	Used to get a pointer to the indicator structure of an object.
InsertItem	Inserts an item to the Index position.
Instance (inherited from TOraObject)	Used to get a pointer to the internal representation of an object.
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.
IsNull (inherited from TOraObject)	Used to verify if an object is empty.
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.
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.
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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Appends an item to the end of an array.

Class

[TOraArray](#)

Syntax

```
function AppendItem: integer;  
Return Value
```

the index of an appended item.

Remarks

Call the AppendItem function to append an item to the end of an array. Returns the index of an appended item.

See Also

- [InsertItem](#)
-

© 1997-2012 Devart. All Rights Reserved.

Removes all items from an array.

Class

[TOraArray](#)

Syntax

```
procedure Clear;
```

Remarks

Call this method to remove all items from an array.

© 1997-2012 Devart. All Rights Reserved.

Inserts an item to the Index position.

Class

[TOraArray](#)

Syntax

```
procedure InsertItem(Index: integer);  
Parameters
```

Index
Holds the index position.

Remarks

Call the InsertItem procedure to insert an item to the Index position.

See Also

- [AppendItem](#)
-

© 1997-2012 Devart. All Rights Reserved.

17.26.1.2 OraObjects.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

```

TObject
  TSharedObject
    TDBObject
      TOraObject
        TOraArray
          TOraNestTable

```

See Also

- [TOraArray](#)
- [TOraNestedTable](#)
- [TOraDataSet.GetTable](#)
- [TOraParam.AsTable](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

<u>AttrAsOCIDate</u> (inherited from <u>TOraObject</u>)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
<u>AttrAsOCINumber</u> (inherited from <u>TOraObject</u>)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
<u>AttrAsOCISString</u> (inherited from <u>TOraObject</u>)	Used to read the attribute's data value as OCISString, or to assign an OCISString value to the contents of an attribute.
<u>AttrAsString</u> (inherited from <u>TOraObject</u>)	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
<u>AttrIsNull</u> (inherited from <u>TOraObject</u>)	Used to indicate if the attribute value is NULL.
<u>Indicator</u> (inherited from <u>TOraObject</u>)	Used to get a pointer to the indicator structure of an object.
<u>Instance</u> (inherited from <u>TOraObject</u>)	Used to get a pointer to the internal representation of an object.
<u>IsNull</u> (inherited from <u>TOraObject</u>)	Used to verify if an object is empty.
<u>ItemAsDateTime</u> (inherited from <u>TOraArray</u>)	Used to read the value of the array's data into a TDateTime object or variable, or to assign a TDateTime value to the contents of the array's value.
<u>ItemAsFloat</u> (inherited from <u>TOraArray</u>)	Used to read the value of the array's data into a Double, or to assign a Double value to the contents of the array's item.
<u>ItemAsInteger</u> (inherited from <u>TOraArray</u>)	Used to read the value of the Item's data into an integer, or to assign an integer value to the contents of the item.
<u>ItemAsObject</u> (inherited from <u>TOraArray</u>)	Used to read the value of the Item's data into a <u>TOraObject</u> , or to assign a TOraObject value to the contents of the item.
<u>ItemAsOCISString</u> (inherited from <u>TOraArray</u>)	Used to get the value of the Item's data as a generic OCI string pointer, or to assign the OCI string value to the content of the item.
<u>ItemAsString</u> (inherited from <u>TOraArray</u>)	Used to read the value of the Item's data into a string, or to assign a string value to the content of the item.
<u>ItemExists</u> (inherited from <u>TOraArray</u>)	Indicates whether the item with Index exists.
<u>ItemIsNull</u> (inherited from <u>TOraArray</u>)	Indicates whether the item contains a value.
<u>ItemType</u> (inherited from <u>TOraArray</u>)	Indicates the type of an array element.
<u>MaxSize</u> (inherited from <u>TOraArray</u>)	Defines the maximum number of elements that an array object may hold.
<u>ObjectType</u> (inherited from <u>TOraObject</u>)	Used to indicate the object type.
<u>OCISvcCtx</u> (inherited from <u>TOraObject</u>)	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 item to the end of an array.
Assign (inherited from TOraObject)	Copies properties or other attributes from another object.
Clear (inherited from TOraArray)	Removes all items 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 item to the Index position.
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.

© 1997-2012 Devart. All Rights Reserved.

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 item to the end of an array.

Assign (inherited from TOraObject)	Copies properties or other attributes from another object.
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.
AttrAsOCString (inherited from TOraObject)	Used to read the attribute's data value as OCString, or to assign an OCString 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.
Clear (inherited from TOraArray)	Removes all items 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.
Indicator (inherited from TOraObject)	Used to get a pointer to the indicator structure of an object.
InsertItem (inherited from TOraArray)	Inserts an item to the Index position.
Instance (inherited from TOraObject)	Used to get a pointer to the internal representation of an object.
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.

[IsNull](#) (inherited from [TOraObject](#))

Used to verify if an object is empty.

[ItemAsDateTime](#) (inherited from [TOraArray](#))

Used to read the value of the array's data into a [TDateTime](#) object or variable, or to assign a [TDateTime](#) value to the contents of the array's value.

[ItemAsFloat](#) (inherited from [TOraArray](#))

Used to read the value of the array's data into a [Double](#), or to assign a [Double](#) value to the contents of the array's item.

[ItemAsInteger](#) (inherited from [TOraArray](#))

Used to read the value of the Item's data into an integer, or to assign an integer value to the contents of the item.

[ItemAsObject](#) (inherited from [TOraArray](#))

Used to read the value of the Item's data into a [TOraObject](#), or to assign a [TOraObject](#) value to the contents of the item.

[ItemAsOCIString](#) (inherited from [TOraArray](#))

Used to get the value of the Item's data as a generic OCI string pointer, or to assign the OCI string value to the content of the item.

[ItemAsString](#) (inherited from [TOraArray](#))

Used to read the value of the Item's data into a string, or to assign a string value to the content of the item.

[ItemExists](#) (inherited from [TOraArray](#))

Indicates whether the item with Index exists.

[ItemIsNull](#) (inherited from [TOraArray](#))

Indicates whether the item contains a value.

[ItemType](#) (inherited from [TOraArray](#))

Indicates the type of an array element.

[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.

[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.

[Refresh](#) (inherited from [TOraObject](#))

Retrieves the latest database image for the object.

[Release](#) (inherited from [TSharedObject](#))

Decrements the reference count.

[Size](#) (inherited from [TOraArray](#))

Holds the size of an array.

[Unmark](#) (inherited from [TOraObject](#))

Marks an object as not being dirty.

See Also

- [TOraNestTable Class](#)
- [TOraNestTable Class Members](#)

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.

© 1997-2012 Devart. All Rights Reserved.

17.26.1.3 OraObjects.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

```
TObject
  TSharedObject
    TDBObject
      TOraObject
```

See Also

-
- [TOraType](#)
- [TOraDataSet.GetObject](#)
- [TOraParam.AsObject](#)

- [TOraRef](#)
- [TOraArray](#)
- [TOraNestTable](#)

© 1997-2012 Devart. All Rights Reserved.

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.
AttrAsOCISString	Used to read the attribute's data value as OCISString, or to assign an OCISString 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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TObject** class.

For a complete list of the **TObject** class members, see the [TObject Members](#) topic.

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
AttrAsArray	Used to get the TObjectArray 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 TObjectLob 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.
AttrAsOCISString	Used to read the attribute's data value as OCISString, or to assign an OCISString 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.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

- [ToraObject Class](#)
- [ToraObject Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to get the ToraArray type attribute value.

Class

[ToraObject](#)

Syntax

```
property AttrFromArray[Name: string]: ToraArray;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrFromArray property to get the value of an attribute of the ToraArray type. Provide an attribute name in the Name index parameter.

© 1997-2012 Devart. All Rights Reserved.

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[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.

© 1997-2012 Devart. All Rights Reserved.

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[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.

© 1997-2012 Devart. All Rights Reserved.

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[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.

© 1997-2012 Devart. All Rights Reserved.

Used to get reference to the TOraLob object that represents the LOBAttribute value.

Class

[TOraObject](#)

Syntax

```
property AttrAsLob[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](#)

© 1997-2012 Devart. All Rights Reserved.

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[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](#)

© 1997-2012 Devart. All Rights Reserved.

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[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.

© 1997-2012 Devart. All Rights Reserved.

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[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.

© 1997-2012 Devart. All Rights Reserved.

Used to read the attribute's data value as OCIStrng, or to assign an OCIStrng value to the contents of an attribute.

Class[TOraObject](#)**Syntax**

```
property AttrAsOCIStrng[Name: string]: pOCIStrng;
```

Parameters*Name*

Holds an attribute name.

Remarks

Use AttrAsOCIStrng property to read the attribute's data value as OCIStrng, or to assign an OCIStrng value to the contents of an attribute.

© 1997-2012 Devart. All Rights Reserved.

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[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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Used to assign a service context handle.

Class

[TOraObject](#)

Syntax

property OCISvcCtx: pOCISvcCtx;

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

[RefCount](#) (inherited from [TSharedObject](#))

Used to return the count of reference to a TSharedObject object.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance.

Class[TOraObject](#)**Overload List**

Name	Description
AllocObject	Allocates an object instance.
AllocObject(SvcCtx: pOCISvcCtx)	Allocates an object instance.
AllocObject(SvcCtx: pOCISvcCtx; TypeName: string)	Allocates an object instance.
AllocObject(TypeName: string)	Allocates an object instance.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance.

Class[TOraObject](#)**Syntax**

```
procedure AllocObject(SvcCtx: pOCISvcCtx); overload
```

Parameters

SvcCtx
Holds service context.

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance.

Class

[TOraObject](#)

Syntax

```
procedure AllocObject(SvcCtx: pOCISvcCtx; TypeName: string);
```

overload; virtual

Parameters

SvcCtx
Holds service context.

TypeName
Holds the type of a TOraObject.

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance.

Class

[TOraObject](#)

Syntax

```
procedure AllocObject(TypeName: string); overload; virtual
```

Parameters

TypeName
Holds the type of a TOraObject.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Creates an object.

Class

[TOraObject](#)

Syntax

```
procedure CreateObject (SvcCtx: pOCISvcCtx; TypeName: string);
```

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.26.1.4 OraObjects.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

```

TObject
  TSharedObject
    TDBObject
      ToraObject
        ToraRef
  
```

See Also

- [ToraObject](#)
- [ToraType](#)
- [ToraDataSet.GetRef](#)
- [ToraParam.AsRef](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
AttrAsOCISString (inherited from ToraObject)	Used to read the attribute's data value as OCISString, or to assign an OCISString 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 reference.
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 reference.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOracleRef** class.

For a complete list of the **TOracleRef** class members, see the [TOracleRef 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 TOracleObject)	Overloaded. Allocates an object instance.
AsHex	Used to convert the REF value converted to the hexadecimal string.
Assign (inherited from TOracleObject)	Copies properties or other attributes from another object.
AttrFromArray (inherited from TOracleObject)	Used to get the TOracleArray type attribute value.
AttrAsDateTime (inherited from TOracleObject)	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 TOracleObject)	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 TOracleObject)	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 TOracleObject)	Used to get reference to the TOracleLob object that represents the LOBAttribute value.
AttrAsObject (inherited from TOracleObject)	Used to read the attribute's data value as TOracleObject, or to assign a TOracleObject value to the contents of an attribute.
AttrAsOCIDate (inherited from TOracleObject)	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 TOracleObject)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCISString (inherited from TOracleObject)	Used to read the attribute's data value as OCISString, or to assign an OCISString value to the contents of an attribute.
AttrAsString (inherited from TOracleObject)	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 TOracleObject)	Used to indicate if the attribute value is NULL.
CreateObject (inherited from TOracleObject)	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.
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.
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.
IsNull (inherited from TOraObject)	Used to verify if an object is empty.
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.
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.
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

- [TOraRef Class](#)
- [TOraRef Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
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.
AttrAsOCISString (inherited from TOraObject)	Used to read the attribute's data value as OCISString, or to assign an OCISString 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.
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.
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.
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.
IsNull (inherited from TOraObject)	Used to verify if an object is empty.
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.
ObjectType (inherited from TOraObject)	Used to indicate the object type.
OCISvcCtx (inherited from TOraObject)	Used to assign a service context handle.
Pin	Pins reference.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject 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 reference.

See Also

- [TOraRef Class](#)
- [TOraRef Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Pins reference.

Class

[TOraRef](#)

Syntax

```
procedure Pin;
```

Remarks

Call the Pin method to pin reference.

See Also

- [TOraRef](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Unpins reference.

Class

[TOraRef](#)

Syntax

```
procedure Unpin;
```

Remarks

Call the Unpin method to unpin reference.

See Also

- [TOraRef](#)

© 1997-2012 Devart. All Rights Reserved.

17.26.1.5 OraObjects.TObjectType 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

```
TObjectType = class (TOBJECTTYPE) ;
```

Remarks

The TOraType class represents Oracle type and holds information about type required for TOraObject objects.

Inheritance Hierarchy

```
TObject  
  TSharedObject  
    TObjectType  
      TOraType
```

See Also

- [TOBJECTTYPE](#)
- [TOraObject](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
AttributeByName (inherited from TObjectType)	Retrieves attribute information for an attribute when only the attribute's name is known.
Describe	Overloaded. Retrieves type information from Oracle.
FindAttribute (inherited from TObjectType)	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from TSharedObject)	Decrements the reference count.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraType** class.

For a complete list of the **TOraType** class members, see the [TOraType Members](#) topic.

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
AttributeByName (inherited from TObjectType)	Retrieves attribute information for an attribute when only the attribute's name is known.
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.

[FindAttribute](#) (inherited from [TObjectType](#))

Indicates whether a specified Attribute component is referenced in the TAttributes object.

[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.

[Release](#) (inherited from [TSharedObject](#))

Decrements the reference count.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

See Also

- [TOraType.Describe](#)

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TOraType** class.

For a complete list of the **TOraType** class members, see the [TOraType Members](#) topic.

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
AttributeByName (inherited from TObjectType)	Retrieves attribute information for an attribute when only the attribute's name is known.
AttributeCount (inherited from TObjectType)	Used to indicate the number of attributes of type.
Attributes (inherited from TObjectType)	Used to access separate attributes.
DataType (inherited from TObjectType)	Used to indicate the type of object dtObject, dtArray or dtTable.
Describe	Overloaded. Retrieves type information from Oracle.
FindAttribute (inherited from TObjectType)	Indicates whether a specified Attribute component is referenced in the TAttributes object.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.
Size (inherited from TObjectType)	Used to learn the size of an object instance.

See Also

- [TOraType Class](#)
- [TOraType Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Retrieves type information from Oracle.

Class

[TOraType](#)

Overload List

Name	Description
Describe(SvcCtx: pOCISvcCtx)	Retrieves type information from Oracle.
Describe(SvcCtx: pOCISvcCtx; Name: string)	Retrieves type information from Oracle.

© 1997-2012 Devart. All Rights Reserved.

Retrieves type information from Oracle.

Class

[TOraType](#)

Syntax

```
procedure Describe (SvcCtx: pOCISvcCtx); overload
```

Parameters

SvcCtx

Holds service context.

Remarks

Call the Describe method to retrieve type information from Oracle. Pass service context by SvcCtx parameter and name of type by the Name parameter.

© 1997-2012 Devart. All Rights Reserved.

Retrieves type information from Oracle.

Class

[TOraType](#)

Syntax

```
procedure Describe (SvcCtx: pOCISvcCtx; Name: string); overload
```

Parameters

SvcCtx

Holds service context.

Name

Holds the type name.

Example

```
var
  OraType: TOraType;
  . . .
  OraType := TOraType.Create();
  OraType.Describe(Session.OCISvcCtx, 'Scott.TCustomer');
```

See Also

- [TOraType.TDO](#)

© 1997-2012 Devart. All Rights Reserved.

17.26.1.6 OraObjects.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><nodel>value</nodel></root>';
...

```

Inheritance Hierarchy

```

TObject
  TSharedObject
    TDBObject
      ToraObject
        ToraXML

```

See Also

- [ToraType](#)
- [ToraXMLField](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

[AttrAsOCISString](#) (inherited from [TOraObject](#))

Used to read the attribute's data value as OCISString, or to assign an OCISString 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 the 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.

© 1997-2012 Devart. All Rights Reserved.

Properties 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 (inherited from TOraObject)	Overloaded. Allocates an object instance.
Assign (inherited from TOraObject)	Copies properties or other attributes from another object.
AsString	Used to get and set the XML document value.
AttrFromArray (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.

[AttrAsOCIStrng](#) (inherited from [TOraObject](#))

Used to read the attribute's data value as OCIStrng, or to assign an OCIStrng 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.

[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.

[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.

[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.

[IsNull](#) (inherited from [TOraObject](#))

Used to verify if an object is empty.

[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.

[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.

[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

- [TOraXML Class](#)
- [TOraXML Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 <code>TSharedObject</code> object.
AllocObject	Overloaded. Allocates an object instance from the existing TOraLob object.
Assign (inherited from TOraObject)	Copies properties or other attributes from another object.
AttrAsArray (inherited from TOraObject)	Used to get the <code>TOraArray</code> type attribute value.
AttrAsDateTime (inherited from TOraObject)	Used to read the attribute's data value into <code>DateTime</code> , or to assign a <code>DateTime</code> 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 <code>TOraLob</code> object that represents the <code>LOBAttribute</code> value.
AttrAsObject (inherited from TOraObject)	Used to read the attribute's data value as <code>TOraObject</code> , or to assign a <code>TOraObject</code> value to the contents of an attribute.
AttrAsOCIDate (inherited from TOraObject)	Used to read the attribute's data value as <code>OCIDate</code> , or to assign an <code>OCIDate</code> value to the contents of an attribute.
AttrAsOCINumber (inherited from TOraObject)	Used to read the attribute's data value into <code>OCINumber</code> , or to assign an <code>OCINumber</code> value to the contents of an attribute.
AttrAsOCISString (inherited from TOraObject)	Used to read the attribute's data value as <code>OCISString</code> , or to assign an <code>OCISString</code> 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.
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.
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.
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.
IsNull (inherited from TOraObject)	Used to verify if an object is empty.
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.
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.
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](#)

Allocates an object instance from the existing [TOraLob](#) object.

Class

[TOraXML](#)

Overload List

Name	Description
AllocObject	Allocates an object instance from the existing TOraLob object.
AllocObject(SvcCtx: pOCISvcCtx; OraLob: TOraLob)	Allocates an object instance from the existing TOraLob object.
AllocObject(SvcCtx: pOCISvcCtx; TypeName: string)	Allocates an object instance from the existing TOraLob object.
AllocObject(TypeName: string)	Allocates an object instance from the existing TOraLob object.

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance from the existing [TOraLob](#) object.

Class

[TOraXML](#)

Syntax

```
procedure AllocObject; overload; override
```

Remarks

Call the AllocObject method to allocate an object instance from the existing [TOraLob](#) object. The procedure modifies service context handle before allocating the object.

See Also

- [TOraObject.AllocObject](#)
- [TOraLob](#)

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance from the existing [TOraLob](#) object.

Class

[TOraXML](#)

Syntax

```
procedure AllocObject(SvcCtx: pOCISvcCtx; OraLob: TOraLob);
```

overload

Parameters

SvcCtx

Holds service context.

OraLob

Holds a TOraLob object the data from which is copied to the XML object.

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance from the existing [TOraLob](#) object.

Class

[TOraXML](#)

Syntax

```
procedure AllocObject(SvcCtx: pOCISvcCtx; TypeName: string);  
overload; override
```

Parameters

SvcCtx

Holds service context.

TypeName

Holds the type of a TOraObject.

© 1997-2012 Devart. All Rights Reserved.

Allocates an object instance from the existing [TOraLob](#) object.

Class

[TOraXML](#)

Syntax

```
procedure AllocObject(TypeName: string); overload; override
```

Parameters

TypeName

Holds the type of a TOraObject.

© 1997-2012 Devart. All Rights Reserved.

Checks if the given set of nodes in the TOraXML exists.

Class

[TOraXML](#)

Syntax

```
function Exists(XPathExpr: string; 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](#)

© 1997-2012 Devart. All Rights Reserved.

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/s
    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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Checks if the input instance conforms to a specified XML schema.

Class

[TOraXML](#)

Syntax

```
function Validate(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
```

See Also

- [Extract](#)
 - [GetSchema](#)
 - [IsSchemaBased](#)
 - [Transform](#)
-

17.27 OraPackage

This unit contains implementation of the TOraPackage component.

Classes

Name	Description
<u>TCustomOraPackage</u>	A a base class for components that provide access to packages stored in an Oracle database.
<u>TOraPackage</u>	A component providing access to packages stored in an Oracle database.

17.27.1 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.

© 1997-2012 Devart. All Rights Reserved.

17.27.1.1 OraPackage.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 (TComponent) ;
```

Inheritance Hierarchy

```
TObject
  TCustomOraPackage
```

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

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**[Session](#)**Description**

Used to specify the session component which is associated with this TOraPackage object.

See Also

- [TCustomOraPackage Class](#)
- [TCustomOraPackage Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Used to specify the session component which is associated with this TOraPackage object.

Class[TCustomOraPackage](#)**Syntax**

property Session: [TOraSession](#);

Remarks

Use the Session property to specify the session component which is associated with this TOraPackage object.
At design-time select a session instance from a dropdown listbox.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TCustomOraPackage** class.

For a complete list of the **TCustomOraPackage** class members, see the [TCustomOraPackage Members](#) topic.

Public**Name**[ExecProc](#)[ExecProcEx](#)[VariableByName](#)**Description**

Calls the stored procedures defined for a given package.

Calls the stored procedures defined for a given package.

Provides access to package variables.

See Also

- [TCustomOraPackage Class](#)

- [TCustomOraPackage Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 stored 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", "MyMessage"]);
```

It is different from the way [ExecProc](#) is used.

See Also

- [ExecProc](#)
- [TCustomDAConnection.ExecProcEx](#)
- [TCustomDAConnection.ExecSQLEx](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.27.1.2 OraPackage.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

TObject
[TCustomOraPackage](#)
TOraPackage

© 1997-2012 Devart. All Rights Reserved.

[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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraPackage** class.

For a complete list of the **TOraPackage** class members, see the [TOraPackage Members](#) topic.

Public

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.
Params (inherited from TCustomOraPackage)	Used to receive values that are the output parameters of package stored procedures.
VariableByName (inherited from TCustomOraPackage)	Provides access to package variables.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.28 OraProvider

This unit contains implementation of the TOraProvider component.

Classes

Name	Description
TOraProvider	A component for loading data to and from a dataset.

© 1997-2012 Devart. All Rights Reserved.

17.28.1 Classes

Classes in the **OraProvider** unit.

Classes

Name	Description
TOraProvider	A component for loading data to and from a dataset.

© 1997-2012 Devart. All Rights Reserved.

17.28.1.1 OraProvider.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** .

Inheritance Hierarchy

TObject

TOraProvider

© 1997-2012 Devart. All Rights Reserved.

[TOraProvider](#) class overview.

© 1997-2012 Devart. All Rights Reserved.

17.29 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.

© 1997-2012 Devart. All Rights Reserved.

17.29.1 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.

© 1997-2012 Devart. All Rights Reserved.

17.29.1.1 OraScript.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

TObject
 [TDAScript](#)
 TOraScript

See Also

- [TOraSQL](#)

© 1997-2012 Devart. All Rights Reserved.

[TOraScript](#) class overview.

Properties

Name	Description
Connection (inherited from TDAScript)	Used to specify the connection in which the script will be executed.
DataSet	Description is not available at the moment.
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)	Indicates whether a specified macro exists in a dataset.
MacroByName (inherited from TDAScript)	Finds a Macro with the name passed in 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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TOraScript** class.

For a complete list of the **TOraScript** class members, see the [TOraScript Members](#) topic.

Public

Name	Description
BreakExec (inherited from TDAScript)	Stops script execution.
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](#))

[EndPos](#) (inherited from [TDAScript](#))

[ErrorOffset](#) (inherited from [TDAScript](#))

[Execute](#) (inherited from [TDAScript](#))

[ExecuteFile](#) (inherited from [TDAScript](#))

[ExecuteNext](#) (inherited from [TDAScript](#))

[ExecuteStream](#) (inherited from [TDAScript](#))

[FindMacro](#) (inherited from [TDAScript](#))

[MacroByName](#) (inherited from [TDAScript](#))

[StartLine](#) (inherited from [TDAScript](#))

[StartOffset](#) (inherited from [TDAScript](#))

[StartPos](#) (inherited from [TDAScript](#))

[Statements](#)

Used to get the offset in the last line of the current statement.

Used to get the end position of the current statement.

Used to get the offset of the statement if the Execute method raised an exception.

Executes a script.

Executes SQL statements contained in a file.

Executes the next statement in the script and then stops.

Executes SQL statements contained in a stream object.

Indicates whether a specified macro exists in a dataset.

Finds a Macro with the name passed in Name.

Used to get the current statement start line number in a script.

Used to get the offset in the first line of the current statement.

Used to get the start position of the current statement in a script.

Contains the list of statements obtained from the SQL property.

Published

Name

[AfterExecute](#) (inherited from [TDAScript](#))

[BeforeExecute](#) (inherited from [TDAScript](#))

[DataSet](#)

[Debug](#) (inherited from [TDAScript](#))

[Delimiter](#) (inherited from [TDAScript](#))

[Macros](#) (inherited from [TDAScript](#))

[OnError](#) (inherited from [TDAScript](#))

[Session](#)

[SQL](#) (inherited from [TDAScript](#))

Description

Occurs after a SQL script execution.

Occurs when taking a specific action before executing the current SQL statement is needed.

Description is not available at the moment.

Used to display the script execution and all its parameter values.

Used to set the delimiter string that separates script statements.

Used to change SQL script text in design- or run-time easily.

Occurs when Oracle raises an error.

Used to specify the session in which the script will be executed.

Used to get or set script text.

See Also

- [TOraScript Class](#)
- [TOraScript Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Class

[TOraScript](#)

Syntax

property DataSet: [TOraDataSet](#);

Remarks

Use the DataSet property to assign a component that will be used by TOraScript to execute statements, and obtain results of execution.

See Also

- [TOraDataSet](#)
 - [TDAScript.ExecuteNext](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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;
        end;
    end;
end;

```

See Also

-

[TOraStatements](#)

© 1997-2012 Devart. All Rights Reserved.

17.29.1.2 OraScript.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(TDASstatement);
```

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

```

TObject
  TDASstatement
    ToraStatement

```

See Also

- [TOraScript](#)
- [TOraStatements](#)
- [TOraStatements](#)

© 1997-2012 Devart. All Rights Reserved.

[TOraStatement](#) class overview.

Properties

Name	Description
EndLine (inherited from TDASstatement)	Used to determine the number of the last statement line in a script.
EndOffset (inherited from TDASstatement)	Used to get the offset in the last line of the statement.
EndPos (inherited from TDASstatement)	Used to get the end position of the statement in a script.
Omit (inherited from TDASstatement)	Used to avoid execution of a statement.

Params	Contains parameters for an SQL statement.
Script (inherited from TDASStatement)	Used to determine the TDAScript object the SQL Statement belongs to.
SQL (inherited from TDASStatement)	Used to get or set the text of an SQL statement.
StartLine (inherited from TDASStatement)	Used to determine the number of the first statement line in a script.
StartOffset (inherited from TDASStatement)	Used to get the offset in the first line of a statement.
StartPos (inherited from TDASStatement)	Used to get the start position of the statement in a script.

© 1997-2012 Devart. All Rights Reserved.

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 TDASStatement)	Used to determine the number of the last statement line in a script.
EndOffset (inherited from TDASStatement)	Used to get the offset in the last line of the statement.
EndPos (inherited from TDASStatement)	Used to get the end position of the statement in a script.
Omit (inherited from TDASStatement)	Used to avoid execution of a statement.
Params	Contains parameters for an SQL statement.
Script (inherited from TDASStatement)	Used to determine the TDAScript object the SQL Statement belongs to.
SQL (inherited from TDASStatement)	Used to get or set the text of an SQL statement.
StartLine (inherited from TDASStatement)	Used to determine the number of the first statement line in a script.
StartOffset (inherited from TDASStatement)	Used to get the offset in the first line of a statement.
StartPos (inherited from TDASStatement)	Used to get the start position of the statement in a script.

See Also

- [TOraStatement Class](#)
- [TOraStatement Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

17.29.1.3 OraScript.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

TObject

[TDAStatements](#)

ToraStatements

See Also

- [TDAScript](#)
- [TDASatement](#)

© 1997-2012 Devart. All Rights Reserved.

[TOraStatements](#) class overview.

Properties

Name	Description
Items	Used to access individual script statements.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

17.30 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 and 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.

17.30.1 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 and 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.

© 1997-2012 Devart. All Rights Reserved.

17.30.1.1 OraSmart.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

```

TObject
  TMemDataSet
    TCustomDADDataSet
      TOraDataSet
        TCustomOraQuery
          TCustomSmartQuery

```

See Also

- [TOraQuery](#)
- [TSmartQuery](#)

- [TOraStoredProc](#)
- [TOraTable](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of 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 TCustomDADataset)	Used to specify the options of the data encryption 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.
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.
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.
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.
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 point to 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
------	-------------

<u>AddWhere</u> (inherited from <u>TCustomDADataset</u>)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>BreakExec</u> (inherited from <u>TCustomDADataset</u>)	Breaks execution of a SQL statement on the server.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.
<u>CreateBlobStream</u> (inherited from <u>TCustomDADataset</u>)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
<u>CreateProcCall</u> (inherited from <u>TOraDataSet</u>)	Generates the stored procedure call.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>ErrorOffset</u> (inherited from <u>TOraDataSet</u>)	Returns the parse error offset.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TOraDataSet</u>)	Indicates if TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether a specified macro exists in a dataset.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TOraDataSet</u>)	Determines whether a parameter with the specified name exists in a dataset.
<u>GetArray</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrArray object for a field when only its name is known.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetErrorPos</u> (inherited from <u>TOraDataSet</u>)	Returns a row and column of parse error for a SQL statement.

<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetFile</u> (inherited from <u>TOraDataSet</u>)	Retrieves a ToraFile object for a field with known name.
<u>GetInterval</u> (inherited from <u>TOraDataSet</u>)	Retrieves a ToraInterval object for a field with known name.
<u>GetKeyList</u> (inherited from <u>TOraDataSet</u>)	Returns the list of table primary key fields.
<u>GetLob</u> (inherited from <u>TOraDataSet</u>)	Retrieves a ToraLob object for a field with known name.
<u>GetLobLocator</u> (inherited from <u>TOraDataSet</u>)	Retrieves a ToraLob object for a field with known name.
<u>GetObject</u> (inherited from <u>TOraDataSet</u>)	Retrieves a ToraObject object for a field with known name.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetRef</u> (inherited from <u>TOraDataSet</u>)	Retrieves a ToraRef object for a field with known name.
<u>GetTable</u> (inherited from <u>TOraDataSet</u>)	Retrieve a ToraNestTable object for a field with known name.
<u>GetTimeStamp</u> (inherited from <u>TOraDataSet</u>)	Retrieves a ToraTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>ParamByName</u> (inherited from <u>TOraDataSet</u>)	Sets or uses parameter information for a specific parameter based on its name.
<u>Prepare</u> (inherited from <u>TCustomDADataset</u>)	Allocates, opens, and parses cursor for a query.
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>Resync</u> (inherited from <u>TCustomDADataset</u>)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	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.
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 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	Occurs before an application posts changes for the current record to the database or cache.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TCustomSmartQuery** class.

For a complete list of the **TCustomSmartQuery** class members, see the [TCustomSmartQuery Members](#) topic.

Public

Name	Description
------	-------------

<u>AddWhere</u> (inherited from <u>TCustomDADataset</u>)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
<u>AfterExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after a component has executed a query to database.
<u>AfterFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs after dataset finishes fetching data from server.
<u>AfterUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after executing insert, delete, update, lock and refresh operations.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>BaseSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>BeforeFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs before dataset is going to fetch block of records from the server.
<u>BeforeUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs before executing insert, delete, update, lock, and refresh operations.
<u>BreakExec</u> (inherited from <u>TCustomDADataset</u>)	Breaks execution of a SQL statement on the server.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>ChangeNotification</u> (inherited from <u>TOraDataSet</u>)	Used to receive database change notification messages to refresh dataset when required.
<u>CheckMode</u> (inherited from <u>TOraDataSet</u>)	Used to define the check mode before editing a record.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>CreateBlobStream</u> (inherited from <u>TCustomDADataset</u>)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
<u>CreateProcCall</u> (inherited from <u>TOraDataSet</u>)	Generates the stored procedure call.
<u>Cursor</u> (inherited from <u>TOraDataSet</u>)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>DependEvents</u>	Used to get or set the names of the events that dataset will depend on in <u>TCustomSmartQuery</u> . <u>SmartRefresh</u> mode.

[DetailFields](#) (inherited from [TCustomDADataset](#))

[Disconnected](#) (inherited from [TCustomDADataset](#))

[DMLRefresh](#) (inherited from [TOraDataSet](#))

[Encryption](#) (inherited from [TCustomDADataset](#))

[ErrorOffset](#) (inherited from [TOraDataSet](#))

[Execute](#) (inherited from [TCustomDADataset](#))

[Executing](#) (inherited from [TCustomDADataset](#))

[Expand](#)

[FetchAll](#) (inherited from [TOraDataSet](#))

[Fetched](#) (inherited from [TOraDataSet](#))

[Fetching](#) (inherited from [TCustomDADataset](#))

[FetchingAll](#) (inherited from [TCustomDADataset](#))

[FetchRows](#) (inherited from [TCustomDADataset](#))

[FilterSQL](#) (inherited from [TCustomDADataset](#))

[FinalSQL](#) (inherited from [TCustomDADataset](#))

[FindKey](#) (inherited from [TCustomDADataset](#))

[FindMacro](#) (inherited from [TCustomDADataset](#))

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TOraDataSet](#))

[GetArray](#) (inherited from [TOraDataSet](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

Used to keep dataset opened after connection is closed.

Used to refresh record by RETURNING clause when insert or update is performed.

Used to specify the options of the data encryption in a dataset.

Returns the parse error offset.

Executes a SQL statement on the server.

Indicates whether SQL statement is still being executed.

Lets all data controls be aware of all the fields belonging to updating table.

Used to request all records of the query from database server when a dataset is being opened.

Indicates if TCustomDADataset has already fetched all rows.

Used to learn whether TCustomDADataset is still fetching rows.

Used to learn whether TCustomDADataset is fetching all rows to the end.

Used to define the number of rows to be transferred across the network at the same time.

Used to change the WHERE clause of SELECT statement and reopen a query.

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Searches for a record which contains specified field values.

Indicates whether a specified macro exists in a dataset.

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.

Determines whether a parameter with the specified name exists in a dataset.

Retrieves a TOrArray object for a field when only its name is known.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

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.
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.
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.
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.
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.

LockMode (inherited from TOraDataSet)	Used to define when to perform the locking of an editing record.
MacroByName (inherited from TCustomDADataset)	Finds a Macro with the name passed in Name.
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.
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.
Options	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
ParamByName (inherited from TOraDataSet)	Sets or uses parameter information for a specific parameter based on its name.
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.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.
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.
Resync (inherited from TCustomDADataset)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
ReturnParams (inherited from TOraDataSet)	Used to return a new fields value to dataset after insert or update.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
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.
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.
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.
SetOrderBy (inherited from TCustomDADataset)	Builds an ORDER BY clause of a SELECT statement.
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.
SQLRefresh (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.
SQLSaved (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.
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.

[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.

[UpdateObject](#) (inherited from [TOraDataSet](#))

Used to point to 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.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TCustomSmartQuery Class](#)
- [TCustomSmartQuery Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 and 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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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 occurs, dataset performs the Refresh procedure.

See Also

- [SmartRefresh](#)
- [DependEvents](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
BeforeFetch (inherited from TCustomDADataset)	Occurs before dataset is going to fetch block of records from the server.

<u>BeforeUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs before executing insert, delete, update, lock, and refresh operations.
<u>BreakExec</u> (inherited from <u>TCustomDADataset</u>)	Breaks execution of a SQL statement on the server.
<u>CachedUpdates</u> (inherited from <u>TMemDataset</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataset</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>ChangeNotification</u> (inherited from <u>TOraDataset</u>)	Used to receive database change notification messages to refresh dataset when required.
<u>CheckMode</u> (inherited from <u>TOraDataset</u>)	Used to define the check mode before editing a record.
<u>CommitUpdates</u> (inherited from <u>TMemDataset</u>)	Clears the cached updates buffer.
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>CreateBlobStream</u> (inherited from <u>TCustomDADataset</u>)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
<u>CreateProcCall</u> (inherited from <u>TOraDataset</u>)	Generates the stored procedure call.
<u>Cursor</u> (inherited from <u>TOraDataset</u>)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DeferredPost</u> (inherited from <u>TMemDataset</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TOraDataset</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>Encryption</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the options of the data encryption in a dataset.
<u>ErrorOffset</u> (inherited from <u>TOraDataset</u>)	Returns the parse error offset.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>FetchAll</u> (inherited from <u>TOraDataset</u>)	Used to request all records of the query from database server when a dataset is being opened.
<u>Fetched</u> (inherited from <u>TOraDataset</u>)	Indicates if TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.

[FetchingAll](#) (inherited from [TCustomDADataset](#))

[FetchRows](#) (inherited from [TCustomDADataset](#))

[FilterSQL](#) (inherited from [TCustomDADataset](#))

[FinalSQL](#) (inherited from [TCustomDADataset](#))

[FindKey](#) (inherited from [TCustomDADataset](#))

[FindMacro](#) (inherited from [TCustomDADataset](#))

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TOraDataSet](#))

[GetArray](#) (inherited from [TOraDataSet](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetErrorPos](#) (inherited from [TOraDataSet](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetFile](#) (inherited from [TOraDataSet](#))

[GetInterval](#) (inherited from [TOraDataSet](#))

[GetKeyList](#) (inherited from [TOraDataSet](#))

[GetLob](#) (inherited from [TOraDataSet](#))

[GetLobLocator](#) (inherited from [TOraDataSet](#))

[GetObject](#) (inherited from [TOraDataSet](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GetRef](#) (inherited from [TOraDataSet](#))

[GetTable](#) (inherited from [TOraDataSet](#))

Used to learn whether TCustomDADataset is fetching all rows to the end.

Used to define the number of rows to be transferred across the network at the same time.

Used to change the WHERE clause of SELECT statement and reopen a query.

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Searches for a record which contains specified field values.

Indicates whether a specified macro exists in a dataset.

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.

Determines whether a parameter with the specified name exists in a dataset.

Retrieves a TOrArray object for a field when only its name is known.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Returns internal field types defined in the MemData and accompanying modules.

Returns a row and column of parse error for a SQL statement.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves a TOrFile object for a field with known name.

Retrieves a TOrInterval object for a field with known name.

Returns the list of table primary key fields.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrObject object for a field with known name.

Retrieves an ORDER BY clause from a SQL statement.

Retrieves a TOrRef object for a field with known name.

Retrieve a TOrNestTable object for a field with known name.

<u>GetTimeStamp</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsPLSQL</u> (inherited from <u>TOraDataSet</u>)	Indicates whether a SQL statement is a PL/SQL block.
<u>IsQuery</u> (inherited from <u>TOraDataSet</u>)	Indicates whether SQL statement returns rows or not.
<u>KeyFields</u> (inherited from <u>TOraDataSet</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
<u>KeySequence</u> (inherited from <u>TOraDataSet</u>)	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.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>LockMode</u> (inherited from <u>TOraDataSet</u>)	Used to define when to perform the locking of an editing record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>NonBlocking</u> (inherited from <u>TOraDataSet</u>)	Used to execute a SQL statement and fetch rows by a separate thread.
<u>OnUpdateError</u> (inherited from <u>TMemDataSet</u>)	Occurs when an exception is generated while cached updates are applied to a database.
<u>OnUpdateRecord</u> (inherited from <u>TMemDataSet</u>)	Occurs when a single update component can not handle the updates.

Options (inherited from TOraDataSet)	Used to specify the behaviour of ToraDataSetObject.
OptionsDS (inherited from TOraDataSet)	Used to specify the behaviour of ToraDataSetObject.
ParamByName (inherited from TOraDataSet)	Sets or uses parameter information for a specific parameter based on its name.
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.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.
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.
Resync (inherited from TCustomDADataset)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
ReturnParams (inherited from TOraDataSet)	Used to return a new fields value to dataset after insert or update.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
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.
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.
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.

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQL](#) (inherited from [TCustomDADataset](#))

[SQLDelete](#) (inherited from [TCustomDADataset](#))

[SQLInsert](#) (inherited from [TCustomDADataset](#))

[SQLRefresh](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[SQLType](#) (inherited from [TOraDataSet](#))

[SQLUpdate](#) (inherited from [TCustomDADataset](#))

[StrictUpdate](#) (inherited from [TOraDataSet](#))

[UniDirectional](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateObject](#) (inherited from [TOraDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

[View](#)

Builds an ORDER BY clause of a SELECT statement.

Used to provide a SQL statement that a query component executes when its Open method is called.

Used to specify a SQL statement that will be used when applying a deletion to a record.

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataset.RefreshRecord](#) procedure.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Used to get the typecode of the SQL statement being processed by Oracle database server.

Used to specify a SQL statement that will be used when applying an update to a dataset.

Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.

Used if an application does not need bidirectional access to records in the result set.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Used to point to an update object component which provides SQL statements that perform updates of the read-only datasets.

Used to indicate the update status for the current record when cached updates are enabled.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Used to check the status of the cached updates buffer.

Indicates the current update status for the dataset when cached updates are enabled.

Allows viewing all fields of the updating table.

See Also

- [TCustomSmartQuery Class](#)
- [TCustomSmartQuery Class Members](#)

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](#)

© 1997-2012 Devart. All Rights Reserved.

Events 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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
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.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
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.
Cursor (inherited from TOraDataSet)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.
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.
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 TCustomDADataset)	Used to specify the options of the data encryption in a dataset.
ErrorOffset (inherited from TOraDataSet)	Returns the parse error offset.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
FetchAll (inherited from TOraDataSet)	Used to request all records of the query from database server when a dataset is being opened.
Fetched (inherited from TOraDataSet)	Indicates if TCustomDADataset has already 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.
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.

[FindKey](#) (inherited from [TCustomDADataset](#))

[FindMacro](#) (inherited from [TCustomDADataset](#))

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TOraDataSet](#))

[GetArray](#) (inherited from [TOraDataSet](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetErrorPos](#) (inherited from [TOraDataSet](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetFile](#) (inherited from [TOraDataSet](#))

[GetInterval](#) (inherited from [TOraDataSet](#))

[GetKeyList](#) (inherited from [TOraDataSet](#))

[GetLob](#) (inherited from [TOraDataSet](#))

[GetLobLocator](#) (inherited from [TOraDataSet](#))

[GetObject](#) (inherited from [TOraDataSet](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GetRef](#) (inherited from [TOraDataSet](#))

[GetTable](#) (inherited from [TOraDataSet](#))

[GetTimeStamp](#) (inherited from [TOraDataSet](#))

[GotoCurrent](#) (inherited from [TCustomDADataset](#))

[IndexFieldNames](#) (inherited from [TMemDataSet](#))

[IsPLSQL](#) (inherited from [TOraDataSet](#))

[IsQuery](#) (inherited from [TOraDataSet](#))

Searches for a record which contains specified field values.

Indicates whether a specified macro exists in a dataset.

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.

Determines whether a parameter with the specified name exists in a dataset.

Retrieves a TOrArray object for a field when only its name is known.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Returns internal field types defined in the MemData and accompanying modules.

Returns a row and column of parse error for a SQL statement.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves a TOrFile object for a field with known name.

Retrieves a TOrInterval object for a field with known name.

Returns the list of table primary key fields.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrObject object for a field with known name.

Retrieves an ORDER BY clause from a SQL statement.

Retrieves a TOrRef object for a field with known name.

Retrieve a TOrNestTable object for a field with known name.

Retrieves a TOrTimeStamp object for a field with known name.

Sets the current record in this dataset similar to the current record in another dataset.

Used to get or set the list of fields on which the recordset is sorted.

Indicates whether a SQL statement is a PL/SQL block.

Indicates whether SQL statement returns rows or not.

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.
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.
LockMode (inherited from TOraDataSet)	Used to define when to perform the locking of an editing record.
MacroByName (inherited from TCustomDADataset)	Finds a Macro with the name passed in Name.
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.
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.
Options (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
ParamByName (inherited from TOraDataSet)	Sets or uses parameter information for a specific parameter based on its name.
ParamCheck (inherited from TCustomDADataset)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TOraDataSet</u>)	Contains the parameters for a query's SQL statement.
<u>Prepare</u> (inherited from <u>TCustomDADataset</u>)	Allocates, opens, and parses cursor for a query.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshFields</u>	Occurs before an application posts changes for the current record to the database or cache.
<u>RefreshMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify when to refresh an editing record.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>Resync</u> (inherited from <u>TCustomDADataset</u>)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
<u>ReturnParams</u> (inherited from <u>TOraDataSet</u>)	Used to return a new fields value to dataset after insert or update.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>RowsProcessed</u> (inherited from <u>TOraDataSet</u>)	Returns the number of rows processed by a query.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SequenceMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>Session</u> (inherited from <u>TOraDataSet</u>)	Used to specify the session in which dataset will be executed.
<u>SetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Builds an ORDER BY clause of a SELECT statement.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	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.
SQLRefresh (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.
SQLSaved (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.
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.
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.
UpdateObject (inherited from TOraDataSet)	Used to point to 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.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.
UpdateStatus (inherited from TMemDataSet)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TCustomSmartQuery Class](#)
- [TCustomSmartQuery Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Occurs after the Smart Refresh procedure was performed by TCustomOraQuery.

Class

[TCustomSmartQuery](#)

Syntax

property AfterSmartRefresh: TDataSetNotifyEvent;

Remarks

Occurs every time after TCustomOraQuery performs Smart Refresh procedure.

See Also

- [SmartRefresh](#)

© 1997-2012 Devart. All Rights Reserved.

Occurs 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.FieldName('DNAME').AsString:=
        quDept.FieldName('DNAME').AsString;
end;
```

© 1997-2012 Devart. All Rights Reserved.

17.30.1.2 OraSmart.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

```
TObject
  TMemDataSet
    TCustomDADataset
      TOraDataSet
        TCustomOraQuery
          TCustomSmartQuery
            TOraTable
```

See Also

- [TSmartQuery](#)

- [Updating Data with ODAC Dataset Components](#)
- [Master/Detail Relationships](#)

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of 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 TCustomDADataset)	Used to specify the options of the data encryption 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.
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.
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.
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.
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 point to 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.
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.
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.
EmptyTable	Truncates the current table content on the server.
ErrorOffset (inherited from TOraDataSet)	Returns the parse error offset.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
Fetched (inherited from TOraDataSet)	Indicates if TCustomDADataset has already 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)	Indicates whether a specified macro exists in a dataset.
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 TOrFile object for a field with known name.
GetInterval (inherited from TOraDataSet)	Retrieves a TOrInterval object for a field with known name.
GetKeyList (inherited from TOraDataSet)	Returns the list of table primary key fields.
GetLob (inherited from TOraDataSet)	Retrieves a TOrLob object for a field with known name.
GetLobLocator (inherited from TOraDataSet)	Retrieves a TOrLob object for a field with known name.
GetObject (inherited from TOraDataSet)	Retrieves a TOrObject 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 TOrRef object for a field with known name.
GetTable (inherited from TOraDataSet)	Retrieve a TOrNestTable object for a field with known name.
GetTimeStamp (inherited from TOraDataSet)	Retrieves a TOrTimeStamp 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 name passed in Name.
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 TOrTable.
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](#))

[Resync](#) (inherited from [TCustomDADataset](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

[View](#) (inherited from [TCustomSmartQuery](#))

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Cancels changes made to the current record when cached updates are enabled.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Builds an ORDER BY clause of a SELECT statement.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Indicates the current update status for the dataset when cached updates are enabled.

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)	Occurs before an application posts changes for the current record to the database or cache.

© 1997-2012 Devart. All Rights Reserved.

Properties 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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
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.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
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.
Cursor (inherited from ToraDataSet)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.

<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>DependEvents</u> (inherited from <u>TCustomSmartQuery</u>)	Used to get or set the names of the events that dataset will depend on in <u>TCustomSmartQuery.SmartRefresh</u> mode.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TOraDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>Encryption</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the options of the data encryption in a dataset.
<u>ErrorOffset</u> (inherited from <u>TOraDataSet</u>)	Returns the parse error offset.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Expand</u> (inherited from <u>TCustomSmartQuery</u>)	Lets all data controls be aware of all the fields belonging to updating table.
<u>Fetched</u> (inherited from <u>TOraDataSet</u>)	Indicates if TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether a specified macro exists in a dataset.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TOraDataSet</u>)	Determines whether a parameter with the specified name exists in a dataset.
<u>GetArray</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrArray object for a field when only its name is known.

<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetErrorPos</u> (inherited from <u>TOraDataSet</u>)	Returns a row and column of parse error for a SQL statement.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetFile</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrFile object for a field with known name.
<u>GetInterval</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrInterval object for a field with known name.
<u>GetKeyList</u> (inherited from <u>TOraDataSet</u>)	Returns the list of table primary key fields.
<u>GetLob</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrLob object for a field with known name.
<u>GetLobLocator</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrLob object for a field with known name.
<u>GetObject</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrObject object for a field with known name.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetRef</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrRef object for a field with known name.
<u>GetTable</u> (inherited from <u>TOraDataSet</u>)	Retrieve a TOrNestTable object for a field with known name.
<u>GetTimeStamp</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOrTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsPLSQL</u> (inherited from <u>TOraDataSet</u>)	Indicates whether a SQL statement is a PL/SQL block.
<u>IsQuery</u> (inherited from <u>TOraDataSet</u>)	Indicates whether SQL statement returns rows or not.
<u>KeyFields</u> (inherited from <u>TOraDataSet</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
<u>KeySequence</u> (inherited from <u>TOraDataSet</u>)	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.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.

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 name passed in Name.
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.
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.
Options (inherited from TCustomSmartQuery)	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
ParamByName (inherited from TOraDataSet)	Sets or uses parameter information for a specific parameter based on its name.
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.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.

<u>RefreshFields</u> (inherited from <u>TCustomSmartQuery</u>)	Occurs before an application posts changes for the current record to the database or cache.
<u>RefreshMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify when to refresh an editing record.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>Resync</u> (inherited from <u>TCustomDADataset</u>)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
<u>ReturnParams</u> (inherited from <u>TOraDataSet</u>)	Used to return a new fields value to dataset after insert or update.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>RowsProcessed</u> (inherited from <u>TOraDataSet</u>)	Returns the number of rows processed by a query.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SequenceMode</u> (inherited from <u>TOraDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>Session</u> (inherited from <u>TOraDataSet</u>)	Used to specify the session in which dataset will be executed.
<u>SetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Builds an ORDER BY clause of a SELECT statement.
<u>SmartRefresh</u> (inherited from <u>TCustomSmartQuery</u>)	Let TCustomSmartQuery components work in a concurrent environment.
<u>SmartState</u> (inherited from <u>TCustomSmartQuery</u>)	Defines if TCustomSmartQuery is in view mode.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[SQLType](#) (inherited from [TOraDataSet](#))

[SQLUpdate](#) (inherited from [TCustomDADataset](#))

[StrictUpdate](#) (inherited from [TOraDataSet](#))

[UniDirectional](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateObject](#) (inherited from [TOraDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

[View](#) (inherited from [TCustomSmartQuery](#))

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Used to get the typecode of the SQL statement being processed by Oracle database server.

Used to specify a SQL statement that will be used when applying an update to a dataset.

Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.

Used if an application does not need bidirectional access to records in the result set.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Used to point to an update object component which provides SQL statements that perform updates of the read-only datasets.

Used to indicate the update status for the current record when cached updates are enabled.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Used to check the status of the cached updates buffer.

Indicates the current update status for the dataset when cached updates are enabled.

Allows viewing all fields of the updating table.

Published

Name

[FetchAll](#)

[LockMode](#)

[OrderFields](#)

[TableName](#)

Description

Defines whether to request all records of the query from database server when the dataset is being opened.

Used to specify what kind of lock will be performed when editing a record.

Used to build ORDER BY clause of SQL statements.

Used to specify the name of the database table this component encapsulates.

See Also

- [TOraTable Class](#)
- [TOraTable Class Members](#)

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.
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.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
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.
BreakExec (inherited from TCustomDADataset)	Breaks execution of a SQL statement on the server.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.

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.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
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.
Cursor (inherited from TOraDataSet)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.
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.
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.
EmptyTable	Truncates the current table content on the server.
Encryption (inherited from TCustomDADataset)	Used to specify the options of the data encryption in a dataset.
ErrorOffset (inherited from TOraDataSet)	Returns the parse error offset.
Execute (inherited from TCustomDADataset)	Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
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.
Fetched (inherited from TOraDataSet)	Indicates if TCustomDADataset has already fetched all rows.
Fetching (inherited from TCustomDADataset)	Used to learn whether TCustomDADataset is still fetching rows.

[FetchingAll](#) (inherited from [TCustomDADataset](#))

[FetchRows](#) (inherited from [TCustomDADataset](#))

[FilterSQL](#) (inherited from [TCustomDADataset](#))

[FinalSQL](#) (inherited from [TCustomDADataset](#))

[FindKey](#) (inherited from [TCustomDADataset](#))

[FindMacro](#) (inherited from [TCustomDADataset](#))

[FindNearest](#) (inherited from [TCustomDADataset](#))

[FindParam](#) (inherited from [TOraDataSet](#))

[GetArray](#) (inherited from [TOraDataSet](#))

[GetBlob](#) (inherited from [TMemDataSet](#))

[GetDataType](#) (inherited from [TCustomDADataset](#))

[GetErrorPos](#) (inherited from [TOraDataSet](#))

[GetFieldObject](#) (inherited from [TCustomDADataset](#))

[GetFieldPrecision](#) (inherited from [TCustomDADataset](#))

[GetFieldScale](#) (inherited from [TCustomDADataset](#))

[GetFile](#) (inherited from [TOraDataSet](#))

[GetInterval](#) (inherited from [TOraDataSet](#))

[GetKeyList](#) (inherited from [TOraDataSet](#))

[GetLob](#) (inherited from [TOraDataSet](#))

[GetLobLocator](#) (inherited from [TOraDataSet](#))

[GetObject](#) (inherited from [TOraDataSet](#))

[GetOrderBy](#) (inherited from [TCustomDADataset](#))

[GetRef](#) (inherited from [TOraDataSet](#))

[GetTable](#) (inherited from [TOraDataSet](#))

Used to learn whether TCustomDADataset is fetching all rows to the end.

Used to define the number of rows to be transferred across the network at the same time.

Used to change the WHERE clause of SELECT statement and reopen a query.

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Searches for a record which contains specified field values.

Indicates whether a specified macro exists in a dataset.

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.

Determines whether a parameter with the specified name exists in a dataset.

Retrieves a TOrArray object for a field when only its name is known.

Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Returns internal field types defined in the MemData and accompanying modules.

Returns a row and column of parse error for a SQL statement.

Returns a multireference shared object from field.

Retrieves the precision of a number field.

Retrieves the scale of a number field.

Retrieves a TOrFile object for a field with known name.

Retrieves a TOrInterval object for a field with known name.

Returns the list of table primary key fields.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrLob object for a field with known name.

Retrieves a TOrObject object for a field with known name.

Retrieves an ORDER BY clause from a SQL statement.

Retrieves a TOrRef object for a field with known name.

Retrieve a TOrNestTable object for a field with known name.

<u>GetTimeStamp</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsPLSQL</u> (inherited from <u>TOraDataSet</u>)	Indicates whether a SQL statement is a PL/SQL block.
<u>IsQuery</u> (inherited from <u>TOraDataSet</u>)	Indicates whether SQL statement returns rows or not.
<u>KeyFields</u> (inherited from <u>TOraDataSet</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
<u>KeySequence</u> (inherited from <u>TOraDataSet</u>)	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.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>LockMode</u> (inherited from <u>TOraDataSet</u>)	Used to define when to perform the locking of an editing record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>NonBlocking</u> (inherited from <u>TOraDataSet</u>)	Used to execute a SQL statement and fetch rows by a separate thread.
<u>OnUpdateError</u> (inherited from <u>TMemDataSet</u>)	Occurs when an exception is generated while cached updates are applied to a database.
<u>OnUpdateRecord</u> (inherited from <u>TMemDataSet</u>)	Occurs when a single update component can not handle the updates.

Options (inherited from TCustomSmartQuery)	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from TOraDataSet)	Used to specify the behaviour of TOraDataSetObject.
ParamByName (inherited from TOraDataSet)	Sets or uses parameter information for a specific parameter based on its name.
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.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
PrepareSQL	Determines KeyFields and builds query of TOraTable.
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.
RefreshFields (inherited from TCustomSmartQuery)	Occurs before an application posts changes for the current record to the database or cache.
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.
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.
Resync (inherited from TCustomDADataset)	Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.
ReturnParams (inherited from TOraDataSet)	Used to return a new fields value to dataset after insert or update.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
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.
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.
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.
SetOrderBy (inherited from TCustomDADataset)	Builds an ORDER BY clause of a SELECT statement.
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.
SQLRefresh (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.
SQLSaved (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.
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.
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.
UpdateObject (inherited from TOraDataSet)	Used to point to 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.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.30.1.3 OraSmart.TSmartQuery Class

A component providing [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 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 and not only those requested in SELECT clause, and [TCustomSmartQuery.SmartRefresh](#) feature (in Professional and Developer editions only).

Inheritance Hierarchy

[TObject](#)
[TMemDataSet](#)
[TCustomDADataset](#)
[TOraDataSet](#)
[TCustomOraQuery](#)
[TCustomSmartQuery](#)
TSmartQuery

See Also

- [TOraQuery](#)
- [TOraTable](#)
- [Updating Data with ODAC Dataset Components](#)

© 1997-2012 Devart. All Rights Reserved.

[TSmartQuery](#) 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.
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.
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of 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 TCustomDADataset)	Used to specify the options of the data encryption 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](#))

[FilterSQL](#) (inherited from [TCustomDADataset](#))

[FinalSQL](#) (inherited from [TCustomDADataset](#))

[IndexFieldNames](#) (inherited from [TMemDataSet](#))

[IsPLSQL](#) (inherited from [TOraDataSet](#))

[IsQuery](#) (inherited from [TOraDataSet](#))

[KeyFields](#) (inherited from [TOraDataSet](#))

[KeySequence](#) (inherited from [TOraDataSet](#))

[LocalConstraints](#) (inherited from [TMemDataSet](#))

[LocalUpdate](#) (inherited from [TMemDataSet](#))

[LockMode](#) (inherited from [TOraDataSet](#))

[MacroCount](#) (inherited from [TCustomDADataset](#))

[Macros](#) (inherited from [TCustomDADataset](#))

[MasterFields](#) (inherited from [TCustomDADataset](#))

[MasterSource](#) (inherited from [TCustomDADataset](#))

[NonBlocking](#) (inherited from [TOraDataSet](#))

[Options](#) (inherited from [TCustomSmartQuery](#))

[OptionsDS](#) (inherited from [TOraDataSet](#))

[ParamCheck](#) (inherited from [TCustomDADataset](#))

[ParamCount](#) (inherited from [TCustomDADataset](#))

Used to define the number of rows to be transferred across the network at the same time.

Used to change the WHERE clause of SELECT statement and reopen a query.

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Used to get or set the list of fields on which the recordset is sorted.

Indicates whether a SQL statement is a PL/SQL block.

Indicates whether SQL statement returns rows or not.

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.

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.

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Used to prevent implicit update of rows on database server.

Used to define when to perform the locking of an editing record.

Used to get the number of macros associated with the Macros property.

Makes it possible to change SQL queries easily.

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.

Used to specify the data source component which binds current dataset to the master one.

Used to execute a SQL statement and fetch rows by a separate thread.

Used to specify the behaviour of TCustomSmartQuery object.

Used to specify the behaviour of TOraDataSetObject.

Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

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.
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.
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.
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 point to 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.

[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.

[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.

[ErrorOffset](#) (inherited from [TOraDataSet](#))

Returns the parse error offset.

[Execute](#) (inherited from [TCustomDADataset](#))

Executes a SQL statement on the server.

[Executing](#) (inherited from [TCustomDADataset](#))

Indicates whether SQL statement is still being executed.

[Fetched](#) (inherited from [TOraDataSet](#))

Indicates if TCustomDADataset has already 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](#))

Indicates whether a specified macro exists in a dataset.

[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.

<u>GetArray</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraArray object for a field when only its name is known.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADDataSet</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetErrorPos</u> (inherited from <u>TOraDataSet</u>)	Returns a row and column of parse error for a SQL statement.
<u>GetFieldObject</u> (inherited from <u>TCustomDADDataSet</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADDataSet</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADDataSet</u>)	Retrieves the scale of a number field.
<u>GetFile</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraFile object for a field with known name.
<u>GetInterval</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraInterval object for a field with known name.
<u>GetKeyList</u> (inherited from <u>TOraDataSet</u>)	Returns the list of table primary key fields.
<u>GetLob</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraLob object for a field with known name.
<u>GetLobLocator</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraLob object for a field with known name.
<u>GetObject</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraObject object for a field with known name.
<u>GetOrderBy</u> (inherited from <u>TCustomDADDataSet</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetRef</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraRef object for a field with known name.
<u>GetTable</u> (inherited from <u>TOraDataSet</u>)	Retrieve a TOraNestTable object for a field with known name.
<u>GetTimeStamp</u> (inherited from <u>TOraDataSet</u>)	Retrieves a TOraTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADDataSet</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADDataSet</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADDataSet</u>)	Finds a Macro with the name passed in Name.
<u>ParamByName</u> (inherited from <u>TOraDataSet</u>)	Sets or uses parameter information for a specific parameter based on its name.
<u>Prepare</u> (inherited from <u>TCustomDADDataSet</u>)	Allocates, opens, and parses cursor for a query.
<u>RefreshRecord</u> (inherited from <u>TCustomDADDataSet</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADDataSet</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.

[RestoreUpdates](#) (inherited from [TMemDataSet](#))

[Resync](#) (inherited from [TCustomDADataset](#))

[RevertRecord](#) (inherited from [TMemDataSet](#))

[SaveSQL](#) (inherited from [TCustomDADataset](#))

[SaveToXML](#) (inherited from [TMemDataSet](#))

[SetOrderBy](#) (inherited from [TCustomDADataset](#))

[SQLSaved](#) (inherited from [TCustomDADataset](#))

[UnLock](#) (inherited from [TCustomDADataset](#))

[UnPrepare](#) (inherited from [TMemDataSet](#))

[UpdateResult](#) (inherited from [TMemDataSet](#))

[UpdateStatus](#) (inherited from [TMemDataSet](#))

[View](#) (inherited from [TCustomSmartQuery](#))

Marks all records in the cache of updates as unapplied.

Resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position.

Cancels changes made to the current record when cached updates are enabled.

Saves the SQL property value to BaseSQL.

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Builds an ORDER BY clause of a SELECT statement.

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Releases a record lock.

Frees the resources allocated for a previously prepared query on the server and client sides.

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Indicates the current update status for the dataset when cached updates are enabled.

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)	Occurs before an application posts changes for the current record to the database or cache.

© 1997-2012 Devart. All Rights Reserved.

17.30.1.4 OraSmart.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

```

TObject
  TDADatasetOptions
    TOraDataSetOptionsDS
      TOraDataSetOptions
        TSmartQueryOptions

```

© 1997-2012 Devart. All Rights Reserved.

[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.
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.
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.

<u>KeepPrepared</u> (inherited from <u>TOraDataSetOptionsDS</u>)	Used to keep TOraDataSet prepared after closing.
<u>LocalMasterDetail</u> (inherited from <u>TDADatasetOptions</u>)	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
<u>LongStrings</u> (inherited from <u>TDADatasetOptions</u>)	Used to represent string fields with the length that is greater than 255 as TStringField.
<u>NumberRange</u> (inherited from <u>TDADatasetOptions</u>)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
<u>PrefetchRows</u> (inherited from <u>TOraDataSetOptions</u>)	Used to get or set the number of rows that OCI prefetches when executing a query.
<u>PrepareUpdateSQL</u> (inherited from <u>TOraDataSetOptions</u>)	Used to automatically prepare update queries before execution.
<u>QueryRecCount</u> (inherited from <u>TDADatasetOptions</u>)	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.
<u>RawAsString</u> (inherited from <u>TOraDataSetOptions</u>)	Used to treat all RAW fields as being of string datatype.
<u>ReflectChangeNotify</u> (inherited from <u>TOraDataSetOptions</u>)	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.
<u>RemoveOnRefresh</u> (inherited from <u>TDADatasetOptions</u>)	Used for a dataset to locally remove a record that can not be found on the server.
<u>RequiredFields</u> (inherited from <u>TDADatasetOptions</u>)	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
<u>ReturnParams</u> (inherited from <u>TDADatasetOptions</u>)	Used to return the new value of fields to dataset after insert or update.
<u>ScrollableCursor</u> (inherited from <u>TOraDataSetOptions</u>)	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.
<u>SetFieldsReadOnly</u> (inherited from <u>TDADatasetOptions</u>)	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.
<u>StatementCache</u> (inherited from <u>TOraDataSetOptions</u>)	Used to get a value indicating whether Oracle resources associated with the current statement will be cached inside a session.
<u>StrictUpdate</u> (inherited from <u>TDADatasetOptions</u>)	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.

17.30.2 Enumerations

Enumerations in the **OraSmart** unit.

Enumerations

Name	Description
TSmartState	Specifies if TCustomSmartQuery is in view mode.

© 1997-2012 Devart. All Rights Reserved.

17.30.2.1 OraSmart.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.

© 1997-2012 Devart. All Rights Reserved.

17.31 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.

17.31.1 Classes

Classes in the **OraSQLMonitor** unit.

Classes

Name	Description
TOraSQLMonitor	This component serves for monitoring dynamic SQL execution in ODAC-based applications.

© 1997-2012 Devart. All Rights Reserved.

17.31.1.1 OraSQLMonitor.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

```

TObject
  TCustomDASQLMonitor
    TOraSQLMonitor

```

See Also

- [TCustomDADataset.Debug](#)
- [TCustomDASQL.Debug](#)
- [DBMonitor](#)

© 1997-2012 Devart. All Rights Reserved.

[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.

17.32 OraTransaction

This unit contains implementation of the TOraTransaction component.

Classes

Name	Description
<u>TOraTransaction</u>	A component for managing transactions in an application.

Enumerations

Name	Description
<u>TGlobalCoordinator</u>	Specifies with what distributed transaction, perform two-phase commit or rollback on all sessions will be coordinated.

17.32.1 Classes

Classes in the **OraTransaction** unit.

Classes

Name	Description
TOraTransaction	A component for managing transactions in an application.

© 1997-2012 Devart. All Rights Reserved.

17.32.1.1 OraTransaction.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

TObject
[TDATransaction](#)
TOraTransaction

See Also

- Transaction demo project
- [TOraTransaction Component](#)
- [TCustomDAConnection.StartTransaction](#)
- [TCustomDAConnection.Commit](#)
- [TCustomDAConnection.Rollback](#)
- [TOraTransaction Component](#)

© 1997-2012 Devart. All Rights Reserved.

[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
<u>AddSession</u>	Overloaded. Associates a TOraSession component with the transaction component.
<u>ClearSessions</u>	Disassociates the transaction component from all its session components.
<u>Commit</u> (inherited from <u>TDATransaction</u>)	Commits the current transaction.
<u>Detach</u>	Deactivates a transaction.
<u>RemoveSession</u>	Disassociates the specified session from the transaction.
<u>Resume</u>	Resumes a detached transaction.
<u>Rollback</u> (inherited from <u>TDATransaction</u>)	Discards all modifications of data associated with the current transaction and ends the transaction.
<u>RollbackToSavepoint</u>	Discards all modifications made during the current transaction and restores its state to the moment of the savepoint.
<u>Savepoint</u>	Defines a savepoint in the transaction.
<u>StartTransaction</u>	Overloaded. Begins a new user transaction against the database server.

Events

Name	Description
<u>OnError</u>	Occurs for processing errors that can be arised during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others.

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.
Commit (inherited from TDATransaction)	Commits the current transaction.
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.
OnError (inherited from TDATransaction)	Used to process errors that occur during executing a transaction.
Rollback (inherited from TDATransaction)	Discards all modifications of data associated with the current transaction and ends the transaction.
Sessions	Used to specify a session for the given index.
SessionsCount	Used to provide the number of sessions associated with the transaction component.
StartTransaction (inherited from TDATransaction)	Begins a new transaction.
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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TOraTransaction** class.

For a complete list of the **TOraTransaction** class members, see the [TOraTransaction Members](#) topic.

Public

Name	Description
Active (inherited from TDATransaction)	Used to determine if the transaction is active.
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.
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.
Detach	Deactivates a transaction.
OnError (inherited from TDATransaction)	Used to process errors that occur during executing 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](#)

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

Deactivates a transaction.

Class

[TOraTransaction](#)

Syntax

```
procedure Detach;
```

Remarks

Call the Detach method to make a transaction inactive.

See Also

- [Resume](#)

© 1997-2012 Devart. All Rights Reserved.

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](#)

© 1997-2012 Devart. All Rights Reserved.

Resumes a detached transaction.

Class

[TOraTransaction](#)

Syntax

```
procedure Resume;
```

Remarks

Call the Resume method to resume a detached transaction.

See Also

- [ResumeTimeOut](#)
 - [Detach](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Begins a new user transaction against the database server.

Class

[TOraTransaction](#)

Syntax

```
procedure StartTransaction; overload; override
```

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

Events of the **TOraTransaction** class.

For a complete list of the **TOraTransaction** class members, see the [TOraTransaction Members](#) topic.

Public

Name	Description
Active (inherited from TDATransaction)	Used to determine if the transaction is active.
Commit (inherited from TDATransaction)	Commits the current transaction.
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.
Rollback (inherited from TDATransaction)	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction (inherited from TDATransaction)	Begins a new transaction.

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](#)
-

© 1997-2012 Devart. All Rights Reserved.

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.

© 1997-2012 Devart. All Rights Reserved.

17.32.2 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.

© 1997-2012 Devart. All Rights Reserved.

17.32.2.1 OraTransaction.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.

© 1997-2012 Devart. All Rights Reserved.

17.33 VirtualTable

This unit contains implementation of the TVirtualTable component.

Classes

Name	Description
TVirtualTable	A base class for storing data in memory.

Types

Name	Description
TVirtualTableOptions	Represents the set of TVirtualTableOption .

Enumerations

Name	Description
TVirtualTableOption	Specifies the actions to take on fields data at the time of opening or closing TVirtualTable dataset.

17.33.1 Classes

Classes in the **VirtualTable** unit.

Classes

Name	Description
TVirtualTable	A base class for storing data in memory.

© 1997-2012 Devart. All Rights Reserved.

17.33.1.1 VirtualTable.TVirtualTable Class

A base class for storing data in memory.

For a list of all members of this type, see [TVirtualTable](#) members.

Unit

[VirtualTable](#)

Syntax

```
TVirtualTable = class (TMemDataSet) ;
```

Remarks

TVirtualTable is inherited from the TMemDataSet component. TVirtualTable stores data in memory and does not have linked data files. To add fields to virtual table at design time use Fields Editor. Call the [TVirtualTable.AddField](#) method to add fields at run time.

Immediately after creating, virtual table will be empty. Then you define new fields or load existing table files so that the virtual table object becomes initialized and ready to be opened.

When you close virtual table it will discard its record set. To keep the data you entered at design-time for later use you may wish to include the voStored option in the [TVirtualTable.Options](#) property. At run time you will need to call the [TVirtualTable.SaveToFile](#) method explicitly to store modifications to the file that may be retrieved back into the virtual table by calling the [TVirtualTable.LoadFromFile](#) method later.

Note: TVirtualTable component is added to the Data Access page of the component palette, not to the Oracle Access page.

Inheritance Hierarchy

TObject

[TMemDataSet](#)

TVirtualTable

© 1997-2012 Devart. All Rights Reserved.

[TVirtualTable](#) 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.
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.
Options	Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

[Prepared](#) (inherited from [TMemDataSet](#))

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Determines whether a query is prepared for execution or not.

Used to indicate the update status for the current record when cached updates are enabled.

Used to check the status of the cached updates buffer.

Methods

Name	Description
AddField	Adds a new TFieldDef object with the name determined by Name.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
Assign	Copies fields and data from another TDataSet component.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
Clear	Removes all records from TVirtualTable.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteField	Deletes a field specified by name.
DeleteFields	Deletes all fields.
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 file into a TVirtualTable component.
LoadFromStream	Copies data of 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.
SaveToFile	Saves data of a TVirtualTable component to a file.
SaveToStream	Copies data from a TVirtualTable component to a stream.
SaveToXML (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
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.

© 1997-2012 Devart. All Rights Reserved.

Properties of the **TVirtualTable** class.

For a complete list of the **TVirtualTable** class members, see the [TVirtualTable Members](#) topic.

Public

Name	Description
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates 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.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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.
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.
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.

Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.
UnPrepare (inherited from TMemDataSet)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.
UpdateStatus (inherited from TMemDataSet)	Indicates the current update status for the dataset when cached updates are enabled.

Published

Name	Description
Options	Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

See Also

- [TVirtualTable Class](#)
- [TVirtualTable Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

Class

[TVirtualTable](#)

Syntax

```
property Options: TVirtualTableOptions default [voPersistentData, voStored];
```

Remarks

The Options property specifies what actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

© 1997-2012 Devart. All Rights Reserved.

Methods of the **TVirtualTable** class.

For a complete list of the **TVirtualTable** class members, see the [TVirtualTable Members](#) topic.

Public

Name	Description
AddField	Adds a new TFieldDef object with the name determined by Name.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
Assign	Copies fields and data from another TDataSet component.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
Clear	Removes all records from TVirtualTable.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteField	Deletes a field specified by name.
DeleteFields	Deletes all fields.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
LoadFromFile	Loads data from file into a TVirtualTable component.
LoadFromStream	Copies data of a stream into a TVirtualTable component.
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.
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.
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.
Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
Prepared (inherited from TMemDataSet)	Determines whether a query is prepared for execution or not.
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.

[SaveToFile](#)

Saves data of a TVirtualTable component to a file.

[SaveToStream](#)

Copies data from a TVirtualTable component to a stream.

[SaveToXML](#) (inherited from [TMemDataSet](#))

Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

[UnPrepare](#) (inherited from [TMemDataSet](#))

Frees the resources allocated for a previously prepared query on the server and client sides.

[UpdateRecordTypes](#) (inherited from [TMemDataSet](#))

Used to indicate the update status for the current record when cached updates are enabled.

[UpdateResult](#) (inherited from [TMemDataSet](#))

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

[UpdatesPending](#) (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

[UpdateStatus](#) (inherited from [TMemDataSet](#))

Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TVirtualTable Class](#)
- [TVirtualTable Class Members](#)

© 1997-2012 Devart. All Rights Reserved.

Adds a new TFieldDef object with the name determined by Name.

Class

[TVirtualTable](#)

Syntax

```
procedure AddField(Name: string; FieldType: TFieldType; Size:
integer = 0; Required: boolean = False);
```

Parameters

Name

Holds the name of the TFieldDef object to add.

FieldType

Holds the type of the TFieldDef object to add.

Size

Holds the size of the string (if the type of TFieldDef object was specified as ftString or ftWideString).

Required

Holds an indicator that determines whether filling the Size parameter is required.

Remarks

Call the AddField method to add a new TFieldDef object with the name determined by Name. FieldType can be ftString, ftWideString, ftSmallint, ftInteger, ftAutoInc, ftWord, ftBoolean, ftLargeint, ftFloat, ftCurrency, ftDate, ftTime, ftDateTime, ftBlob, or ftMemo. When you add ftString or ftWideString field you should specify Size of the string.

Example

```
VirtualTable1.AddField('CODE', ftInteger, 0);
VirtualTable1.AddField('NAME', ftString, 30);
```

See Also

- [DeleteField](#)
- [DeleteFields](#)

© 1997-2012 Devart. All Rights Reserved.

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 SQL server oriented dataset.

Example

```
OraQuery1.SQL.Text := 'SELECT * FROM DEPT';  
OraQuery1.Active := True;  
VirtualTable1.Assign(OraQuery1);  
VirtualTable1.Active := True;
```

See Also

- [TVirtualTable](#)

© 1997-2012 Devart. All Rights Reserved.

Removes all records from TVirtualTable.

Class

[TVirtualTable](#)

Syntax

```
procedure Clear;
```

Remarks

Call the Clear method to remove all records from TVirtualTable.

© 1997-2012 Devart. All Rights Reserved.

Deletes a field specified by name.

Class

[TVirtualTable](#)

Syntax

```
procedure DeleteField(Name: string);
```

Parameters

Name

Holds the name of the field to delete.

Remarks

Call the DeleteField method to delete a field specified by Name.

See Also

- [AddField](#)
 - [DeleteFields](#)
-

© 1997-2012 Devart. All Rights Reserved.

Deletes all fields.

Class

[TVirtualTable](#)

Syntax

```
procedure DeleteFields;
```

Remarks

Call the DeleteFields method to delete all fields.

See Also

- [DeleteField](#)
-

© 1997-2012 Devart. All Rights Reserved.

Loads data from file into a TVirtualTable component.

Class

[TVirtualTable](#)

Syntax

```
procedure LoadFromFile(const FileName: string; LoadFields: boolean  
= True);
```

Parameters

FileName

Holds the name of the file to load data from.

LoadFields

Indicates whether to load fields from the file.

Remarks

Call the LoadFromFile method to load data from 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. File format will be detected automatically.

© 1997-2012 Devart. All Rights Reserved.

Copies data of a stream into a TVirtualTable component.

Class

[TVirtualTable](#)

Syntax

```
procedure LoadFromStream(Stream: TStream; LoadFields: boolean =  
True);
```

Parameters

Stream

Holds the stream from which the field's value is copied.

LoadFields

Indicates whether to load fields from the stream.

Remarks

Call the LoadFromStream method to copy data of 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. Data format will be detected automatically.

© 1997-2012 Devart. All Rights Reserved.

Saves data of a TVirtualTable component to a file.

Class

[TVirtualTable](#)

Syntax

```
procedure SaveToFile(const FileName: string; StoreFields: boolean  
= True);
```

Parameters

FileName

Holds the name of the file to save data to.

StoreFields

Indicates whether to save fields to a file.

Remarks

Call the SaveToFile method to save data of a TVirtualTable component to a file. Specify the name of the file as the value of the FileName parameter.

© 1997-2012 Devart. All Rights Reserved.

Copies data from a TVirtualTable component to a stream.

Class

[TVirtualTable](#)

Syntax

```
procedure SaveToStream(Stream: TStream; StoreFields: boolean =  
True);
```

Parameters

Stream

Holds the name of the stream to which the field's value is saved.

StoreFields

Indicates whether to save the fields names to a file.

Remarks

Call the `SaveToStream` method to copy data from a `TVirtualTable` component to a stream. Specify the name of the stream to which the field's value is saved as the value of the `Stream` parameter.

© 1997-2012 Devart. All Rights Reserved.

17.33.2 Types

Types in the **VirtualTable** unit.

Types

Name	Description
TVirtualTableOptions	Represents the set of TVirtualTableOption .

© 1997-2012 Devart. All Rights Reserved.

17.33.2.1 VirtualTable.TVirtualTableOptions Set

Represents the set of [TVirtualTableOption](#).

Unit

[VirtualTable](#)

Syntax

```
TVirtualTableOptions = set of TVirtualTableOption;
```

© 1997-2012 Devart. All Rights Reserved.

17.33.3 Enumerations

Enumerations in the **VirtualTable** unit.

Enumerations

Name	Description
TVirtualTableOption	Specifies the actions to take on fields data at the time of opening or closing TVirtualTable dataset.

© 1997-2012 Devart. All Rights Reserved.

17.33.3.1 VirtualTable.TVirtualTableOption Enumeration

Specifies the actions to take on fields data at the time of opening or closing TVirtualTable dataset.

Unit

[VirtualTable](#)

Syntax

```
TVirtualTableOption = (voPersistentData, voStored);
```

Values

Value	Meaning
voPersistentData	Dataset will not dispose of its data at the time of dataset closing.
voStored	Dataset will keep its data set at design-time in DFM file along with other form's stored properties.

© 1997-2012 Devart. All Rights Reserved.

Index

- 6 -

64-bit Development with Embarcadero RAD Studio XE2 111

- A -

AbortOnKeyViol Property 133

AbortOnProblem Property 133

Active Property

TCustomDASQLMonitor 208

TDAAlerter 157

TDATransaction 341

TMacro 344

TOraChangeNotification 433

TOraErrorHandler 738

TOraTransaction 895

Add Method 671

AddDBTypeRule Method 315

AddDrop Property 171

AddField Method 912

AddFieldNameRule Method 319

AddRef Method 387

Address Property 669

AddRule Method 321

AddSession Method 899

AddSubscriber Method 645

AddWhere Method 265

AfterExecute Event

TCustomDADDataSet 278

TCustomDASQL 292

TDAScript 197

AfterFetch Event 279

AfterSmartRefresh Event 857

AfterUpdateExecute Event 279

AllocCursor Method 693

AllocDateTime Method 726

AllocInterval Method 706

AllocLob Method 713

AllocObject Method

AllocObject 806, 807

TOraObject 783

TOraXML 806

AlterComment Method

TOraQueueAdmin 646

TOraQueueTable 663

AlterMaxRetries Method 646

AlterPrimaryInstance Method 664

AlterPropagationSchedule Method 646

AlterQueue Method 647

AlterQueueTable Method 664

AlterRetentionTime Method 648

AlterRetryDelay Method 648

AlterSecondaryInstance Method 665

AlterSubscriber Method 648

AppendItem Method 768

Apply Method 298

ApplyUpdates Method

ApplyUpdates 399, 400

TCustomDAConnection 233

TMemDataSet 399

ArrayLength Property 557

AsArray Property 497

AsBFile Property 497

AsBlob Property 331

AsBLOBLocator Property 498

AsBlobRef Property 331

AsCLOBLocator Property 498

AsCursor Property

TCursorField 425

TOraParam 498

AsDateTime Property

TMacro 345

TOraTimeStamp 723

AsFile Property 422

AsFloat Property

TDAParam 331

TMacro 345

TOraNumber 718

AsHex Property 791

AsInteger Property

TDAParam 331

TMacro 345

TOraNumber 718

AsInterval Property

TOraIntervalField 484

TOraParam 499

AsLargeInt Property

TDAParam 332

TOraNumber 718

AsMemo Property 332

AsMemoRef Property 332
 AsNumber Property
 TOraNumberField 492
 TOraParam 499
 AsObject Property 499
 AsOraBlob Property 499
 AsOraClob Property 500
 AsRef Property 500
 Assign Method
 TBlob 376
 TOraObject 784
 TVirtualTable 913
 AssignConnect Method 540
 AssignField Method 335
 AssignFieldValue Method 335
 AssignTo Method 720
 AssignValues Method 348
 AsSQLTimeStamp Property 333
 AsString Property
 TBlob 374
 TDAParam 333
 TMacro 345
 TOraInterval 703
 TOraNumber 719
 TOraTimeStamp 723
 TOraXML 803
 AsTable Property 500
 AsTimeStamp Property
 TOraParam 500
 TOraTimeStampField 583
 AsWideString Property
 TBlob 374
 TDAParam 333
 AsXML Property
 TOraParam 501
 TOraXMLField 593
 AsyncNotification Property 631
 Attempts Property 676
 AttrAsArray Property(Indexer) 777
 AttrAsDateTime Property(Indexer) 778
 AttrAsFloat Property(Indexer) 778
 AttrAsInteger Property(Indexer) 778
 AttrAsLob Property(Indexer) 778
 AttrAsObject Property(Indexer) 779
 AttrAsOCIDate Property(Indexer) 779
 AttrAsOCINumber Property(Indexer) 780
 AttrAsOCISString Property(Indexer) 780
 AttrAsString Property(Indexer) 780

AttributeByName Method 384
 AttributeCount Property 383
 AttributeNo Property 369
 Attributes Property(Indexer) 383
 AttrIsNull Property(Indexer) 781
 AutoClose Property
 TOraDataSetOptions 473
 TOraDataSetOptionsDS 480
 AutoCommit Property
 TOraAlerter 610
 TOraSession 529
 Automatic Key Field Value Generation 50
 AutoPrepare Property 304
 AutoRefresh Property 422
 AutoRegister Property 157

- B -

Backup Method 166
 BackupQuery Method 166
 BackupToFile Method 166
 BackupToStream Method 167
 BaseSQL Property 250
 BaseSQLOldBehavior Variable 358
 BDESession Unit Members 118
 BeforeExecute Event 197
 BeforeFetch Event 279
 BeforeUpdateExecute Event 280
 BLOB and CLOB Data Types 58
 BlobType Property 423
 bmAppend 140
 bmAppendUpdate 140
 bmDelete 140
 bmUpdate 140
 BranchQualifiers Property(Indexer) 896
 BreakExec Method
 TCustomDADDataSet 265
 TDAScript 193
 TOraSQL 561

- C -

CacheCalcFields Property 304
 Cached Property 711
 CachedUpdates Property 395
 CacheLobs Property
 TOraDataSetOptions 473

- CacheLobs Property
 - TOraDataSetOptionsDS 480
- CancelButton Property 222
- CancelUpdates Method 400
- CanFetch Method 694
- Caption Property 222
- CDA Property 692
- ChangeCursor Property 282
- ChangeCursor Variable 358
- ChangedCount Property 134
- ChangeNotification Property 447
- ChangePassword Method 541
- CharLength Property 550
- Charset Property 550
- CheckMode Property 447
- clApply 389
- clConnect 389
- clConnectionApply 389
- Clear Method
 - TBlob 376
 - TOraArray 768
 - TOraRef 794
 - TVirtualTable 913
- ClearSessions Method 900
- clExecute 389
- ClientIdentifier Property 550
- clOpen 389
- Close Method
 - TOraErrorHandler 739
 - TOraFile 700
- clRefresh 389
- clServiceQuery 389
- clTransStart 389
- clUnknown 389
- cmException 597
- cmNone 597
- cmNormal 730
- cmRefresh 597
- cmSysASM 730
- cmSysDBA 730
- cmSysOper 730
- Columns Property 178
- Comment Property
 - TOraQueueAdmin 642
 - TOraQueueTable 659
- Commit Method
 - TCustomDACConnection 234
 - TDATransaction 341
- CommitCount Property 134
- CommitUpdates Method 401
- Compare Method
 - TOraInterval 706
 - TOraNumber 720
 - TOraTimeStamp 727
- Compatibility 28
- Compatibility with Previous Versions 99
- Compatible Property 659
- Component List 24
- Component Property 219
- Connect Method 234
- ConnectButton Property 223
- ConnectDialog Property 228
- Connected Property 530
- Connecting in Direct Mode 45
- Connection Property
 - TCustomDADataset 251
 - TCustomDASQL 282
 - TDAAlerter 157
 - TDADump 163
 - TDALoader 178
 - TDAMetaData 325
 - TDAScript 189
- ConnectionLifetime Property 351
- ConnectionTimeout Property 550
- ConnectMode Property 530
- ConnectPrompt Property 531
- ConnectionString Property 531
- Construct Method 727
- ConsumerName Property 624
- ConvertEOL Property
 - TCustomDACConnection 228
 - TOraSessionOptions 551
- Correlation Property
 - TDequeueOptions 624
 - TQueueMessageProperties 676
- CRAccess Unit Members 126
- CRBatchMove Unit Members 130
- CRDataTypeMap Unit Members 141
- CreateBlobStream Method 266
- CreateColumns Method 179
- CreateDataSet Method 235
- CreateObject Method 784
- CreateProcCall Method
 - TOraDataSet 460
 - TOraSQL 562
- CreateQueue Method 649

CreateQueueTable Method 665
 CreateSQL Method 235
 CreateTemporary Method 713
 CREncryption Unit Members 148
 Cursor Property 448

- D -

DAAlerter Unit Members 155
 DADDataAdapter Class 361
 DADDataAdapter.DataSet Property 362
 DADDataAdapter.Fill Method 362
 DADDataAdapter.Update Method 363
 DADump Unit Members 161
 DALoader Unit Members 173
 DAScript Unit Members 186
 DASQLMonitor Unit Members 206
 Data Encryption 83
 Data Type Mapping 79
 Database Property 125
 Database Specific Aspects of 64-bit Development 115
 DatabaseName Property 125
 DataHeader Property 150
 DataSet Manager 91
 DataSet Property
 DADDataAdapter 362
 TCustomDAUpdateSQL 294
 TDAScript 189
 TOraScript 825
 DataSize Property
 TAttribute 370
 TOraType 797
 DataType Property
 TAttribute 370
 TDAParam 333
 TObjectType 383
 DateFormat Property
 TDPColumn 747
 TOraSessionOptions 551
 DateLanguage Property 551
 DBAccess Unit Members 215
 dbForge Studio for Oracle 102
 DBLengthMax Property
 TDAMapRule 311
 TMapRule 145
 DBLengthMin Property
 TDAMapRule 312
 TMapRule 145
 DBMonitor 96
 DBMonitorOptions Property 208
 DBScaleMax Property
 TDAMapRule 312
 TMapRule 145
 DBScaleMin Property
 TDAMapRule 312
 TMapRule 146
 DBType Property
 TDAMapRule 312
 TMapRule 146
 Debug Property
 TCustomDADataset 251
 TCustomDASQL 283
 TDADump 164
 TDAScript 190
 TOraErrorHandler 738
 TOraSession 532
 DefaultCloseAction Property 341
 DefaultSession Property 896
 DefaultSortType Property 300
 DefaultValues Property
 TDADatasetOptions 304
 TOraDataSetOptions 473
 TOraDataSetOptionsDS 480
 DeferredLobRead Property
 TOraDataSetOptions 473
 TOraDataSetOptionsDS 481
 DeferredPost Method 401
 DefSession Variable 604
 Delay Property 676
 DeleteField Method 913
 DeleteFields Method 914
 DeleteItem Method 774
 DeleteObject Property 294
 DeleteSQL Property 294
 DeleteWhere Method 266
 Delimiter Property 190
 DeliveryMode Property
 TDequeueOptions 624
 TEnqueueOptions 628
 TQueueMessageProperties 677
 Demo Projects 18
 DependEvents Property 844
 Deployment 37
 Dequeue Method 634
 DequeueCondition Property 625

DequeueMode Property 625
DequeueOptions Property 631
Describe Method 798
DescriptorType Property
 TOraInterval 704
 TOraTimeStamp 724
Destination Property 134
Detach Method 901
DetailDelay Property 304
DetailFields Property 251
Devart.Dac.DataAdapter Unit Members 360
Devart.Odac.DataAdapter Unit Members 364
DialogClass Property 223
Direct Property 551
DisableBuffering Method 714
DisablePropagationSchedule Method 649
Disconnect Method 235
Disconnected Mode 78
Disconnected Property 252
DisconnectedMode Property 300
DML Array 69
DMLRefresh Property 448
DoNotRaiseExcetionOnUaFail Variable 411
dpAbort 756
dpFail 756
dpIgnore 756
dqmBrowse 681
dqmLocked 681
dqmRemove 681
dqmRemoveNoData 681
DropQueue Method 650
DropQueueTable Method 665
dsView 888

- E -

eaAbort 205
eaAES128 153
eaAES192 153
eaAES256 153
eaBlowfish 153
eaCast128 153
eaContinue 205
eaException 205
eaFail 205
eaRC4 153
eaTripleDES 153
EDAError Class 219
EDAError.Component Property 219
EDAError.ErrorCode Property 220
EDataMappingError Class 142
EDataTypeMappingError Class 142
Editions 30
ehNone 153
ehTag 153
ehTagAndHash 153
EInvalidDBTypeMapping Class 143
EInvalidFieldTypeMapping Class 143
EmptyTable Method 878
EnableBuffering Method 714
Enabled Property
 TOraChangeNotification 433
 TOraTrace 585
EnableIntegers Property 552
EnableNumbers Property 552
EnableOraTimestamp Property 552
EnablePropagationSchedule Method 650
Encryption Property 252
EncryptionAlgorithm Property 150
Encryptor Property 310
EndLine Property
 TDAScript 190
 TDASTatement 199
EndOffset Property
 TDAScript 190
 TDASTatement 199
EndPos Property
 TDAScript 191
 TDASTatement 199
Enqueue Method 635
EnqueueArray Method 637
EnqueueMessageProperties Property 632
EnqueueOptions Property 632
EnqueueTime Property 677
EOraError Class 743
EOraError.Sender Property 744
ErrorCode Property 220
ErrorOffset Method
 TDAScript 194
 TOraDataSet 461
 TOraSQL 562
etAlert 619
etPipe 619

EUnsupportedDataMapping Class
144

Events Property 611

EventType Property 611

ExceptionQueue Property 677

ExecProc Method

TCustomDAConnection 236

TCustomOraPackage 816

TOraStoredProc 582

ExecProcEx Method

TCustomDAConnection 237

TCustomOraPackage 816

ExecSQL Method

TCustomDAConnection 238

TCustomDAUpdateSQL 298

ExecSQLEx Method 238

Execute Method

Execute 287, 288

TCRBatchMove 137

TCustomConnectDialog 225

TCustomDADataset 266

TCustomDASQL 287

TDAScript 194

ExecuteFile Method 194

ExecuteNext Method 195

ExecuteStream Method 195

Executing Method

TCustomDADataset 267

TCustomDASQL 288

Exists Method

TOraFile 700

TOraObject 785

TOraXML 807

Exists Property 423

Expand Property 844

Expiration Property 677

ExtendedFieldsInfo Property 474

Extract Method 808

- F -

Features 8

FetchAll Property

TOraDataSet 448

TOraQuery 522

TOraTable 871

Fetch Method

TCustomDADataset 267

TOraDataSet 461

Fetching Method 267

FetchingAll Method 268

FetchRows Property 252

FieldLength Property

TDAMapRule 313

TMapRule 146

FieldMappingMode Property 134

FieldName Property

TDAMapRule 313

TMapRule 146

Fields Property 310

FieldsAsString Property

TOraDataSetOptions 474

TOraDataSetOptionsDS 481

FieldScale Property

TDAMapRule 313

TMapRule 146

FieldsOrigin Property 305

FieldType Property

TDAColumn 175

TDAMapRule 313

FileDir Property

TBFileField 423

TOraFile 698

FileName Property

TBFileField 424

TOraFile 698

Fill Method 362

FilterSQL Property 252

FinalSQL Property

TCustomDADataset 253

TCustomDASQL 283

FindAttribute Method 385

FindKey Method 268

FindMacro Method

TCustomDADataset 269

TCustomDASQL 288

TDAScript 195

TMacros 348

FindNearest Method 269

FindParam Method

TCustomDADataset 269

TCustomDASQL 289

TDAParams 339

TOraDataSet 461

TOraSQL 562

FlatBuffers Property 305

FloatPrecision Variable 732
Flush Method 785
Format Property
 TOraTimeStamp 724
 TOraTimeStampField 583
FracPrecision Property
 TOraInterval 704
 TOraIntervalField 484
FreeCursor Method 694
FreeInterval Method 707
FreeLob Method 714
FreeObject Method 785
FreeTemporary Method 714
Frequently Asked Questions 40
fsAbort 597
fsBegin 597
fsEnd 597
fsError 597
fsReauth 597
ftSelect 598
ftSession 598
FullRefresh Property 474

- G -

gcInternal 905
gcMTS 905
GenerateHeader Property 171
GetArray Method 462
GetBlob Method 401
GetDatabaseNames Method 239
GetDataType Method 270
GetFieldObject Method 270
GetFieldPrecision Method 271
GetFieldScale Method 271
GetFile Method 462
GetInterval Method 463
GetKeyList Method 463
GetLob Method 464
GetLobLocator Method 464
GetMessage Method 613
GetMetaDataKinds Method 328
GetObject Method 465
GetOrderBy Method 272
GetRef Method 465
GetRestrictions Method 328
GetSequenceNames Method 542
GetServerList Method 225

GetSessionPID Method 588
GetStoredProcNames Method 240
GetSubscribers Method 651
GetTable Method 466
GetTableNames Method 240
GetTimeStamp Method 466
Getting Started 4
Getting Support 39
GetTraceFileName Method 588
GlobalCoordinator Property 896
GotoCurrent Method 272
GrantQueuePrivilege Method 651
GrantSystemPrivilege Method 666

- H -

haMD5 154
haSHA1 154
HashAlgorithm Property 150
Hierarchy Chart 26
Home Property 532
HomeName Property 532
Host Property 211

- I -

IgnoreErrors Property
 TDAMapRule 314
 TMapRule 146
ihFail 154
ihIgnoreError 154
ihSkipData 154
ilReadCommitted 129
ilReadOnly 598
ilSerializable 598
InactiveTimeOut Property 897
Increasing Performance 85
IndexFieldNames Property 396
Indicator Property 781
IndicatorSize Property 797
Init Method 715
Insert Method 672
InsertItem Method 768
InsertObject Property 295
InsertSQL Property 295
Installation 33
Instance Property 781

IntegerPrecision Variable 732
 InternalName Property 533
 Interval Property 611
 InTransaction Property 228
 InvalidHashAction Property 151
 IsDirty Method 786
 IsEqual Method 349
 IsInit Method 715
 IsLocked Method 786
 IsNull Property
 TDAParam 334
 TOraInterval 704
 TOraNumber 719
 TOraObject 781
 TOraTimeStamp 724
 IsolationLevel Property 897
 IsOpen Method 701
 IsPLSQL Property 449
 IsQuery Property
 TCustomDADataset 253
 TOraDataSet 449
 IsSchemaBased Method 809
 IsTemporary Method 715
 IsUnicode Property 374
 ItemAsDateTime Property(Indexer)
 TOraArray 763
 TOraParam 501
 ItemAsFloat Property(Indexer)
 TOraArray 763
 TOraParam 501
 ItemAsInteger Property(Indexer)
 TOraArray 763
 TOraParam 502
 ItemAsInterval Property(Indexer) 502
 ItemAsObject Property(Indexer) 764
 ItemAsOCIString Property(Indexer) 764
 ItemAsString Property(Indexer)
 TOraArray 764
 TOraParam 503
 ItemAsTimeStamp Property(Indexer) 503
 ItemClear Method 506
 ItemExists Property(Indexer) 765
 ItemIsNull Property(Indexer)
 TOraArray 765
 TOraParam 503
 Items Property(Indexer)
 TDAColumns 176
 TDAParams 338

TDASentences 202
 TMacros 347
 TOraParams 508
 TOraSentences 830
 TQueueAgents 671
 ItemText Property(Indexer) 504
 ItemType Property 765
 ItemValue Property(Indexer) 504

- K -

KeepDesignConnected Property 300
 KeepPrepared Property 481
 KeyFields Property
 TCustomDADataset 253
 TOraDataSet 449
 KeySequence Property 450
 KeyViolCount Property 135

- L -

LabelSet Property 223
 LargeIntPrecision Variable 733
 LastError Property 533
 LDA Property 533
 LeadPrecision Property
 TOraInterval 705
 TOraIntervalField 485
 Length Property
 TAttribute 370
 TOraParam 504
 LengthLob Method 715
 Licensing and Subscriptions 38
 Listen Method 638
 ImDirect 756
 ImDML 756
 ImLockDelayed 356
 ImLockImmediate 356
 ImNone 356
 Load Method 180
 LoadFromDataSet Method 180
 LoadFromFile Method
 TBlob 376
 TDAParam 336
 TVirtualTable 914
 LoadFromStream Method
 TBlob 377

- LoadFromStream Method
 - TDAParam 336
 - TOraXML 810
 - TVirtualTable 915
- LoadMode Property 750
- LocalConstraints Property 396
- LocalFailover Property 301
- LocalMasterDetail Property 305
- LocalUpdate Property 396
- Locate Method 402
- LocateEx Method 404
- Lock Method
 - TCustomDADataset 272
 - TOraObject 786
- LockMode Property
 - TOraDataSet 450
 - TOraQuery 523
 - TOraStoredProc 575
 - TOraTable 871
- LockObject Property 295
- LockSQL Property 296
- LoginPrompt Property 229
- LongStrings Property 306
- IsCustom 356
- IsEnglish 356
- IsFrench 356
- IsGerman 356
- IsItalian 356
- IsPolish 356
- IsPortuguese 356
- IsRussian 356
- IsSpanish 356
- IxCASEnsensitive 390
- IxNearest 390
- IxNext 390
- IxPartialCompare 390
- IxPartialKey 390
- IxUp 390

- M -

- MacroByName Method
 - TCustomDADataset 273
 - TCustomDASQL 289
 - TDAScript 196
 - TMacros 349
- MacroChar Variable 359
- MacroCount Property

- TCustomDADataset 254
- TCustomDASQL 283
- Macros 89
- Macros Property
 - TCustomDADataset 254
 - TCustomDASQL 284
 - TDAScript 191
- Mappings Property 135
- MarkDelete Method 786
- MarkUpdate Method 787
- Master/Detail Relationships 48
- MasterFields Property 255
- MasterSource Property 255
- MaxPoolSize Property 351
- MaxRetries Property 642
- MaxSize Property 765
- MaxTraceFileSize Property 586
- MemData Unit Members 367
- MemDS Unit Members 392
- MessageGrouping Property 660
- MessageId Property
 - TDequeueOptions 625
 - TQueueMessage 673
- MessageProperties Property 673
- MetaDataKind Property 325
- MigrateQueueTable Method 666
- Migration Wizard 97
- MinPoolSize Property 351
- mmFieldIndex 140
- mmFieldName 140
- moCustom 213
- moDBMonitor 213
- Mode Property 135
- moDialog 213
- Modified Property
 - TOraDataSetField 468
 - TOraReferenceField 524
- ModifyObject Property 296
- ModifySQL Property 296
- moHandled 213
- MonitorMessage Method 241
- moSQLMonitor 213
- MovedCount Property 136
- mtDate 730
- mtNone 730
- mtNumber 730
- mtString 730
- MultipleConsumers Property

MultipleConsumers Property
 TOraQueueAdmin 642
 TOraQueueTable 660

- N -

Name Property
 TDAColumn 175
 TMacro 346
 TQueueAgent 669
 National Property 505
 Navigation Property 625
 NeverConnect Property 552
 NextItemType Method 613
 NextMessageType Method 613
 NonBlocking Property
 TOraDataSet 451
 TOraSQL 558
 ntBCD 390
 ntFloat 390
 ntFmtBCD 390
 NumberRange Property 306

- O -

Objects 63
 ObjectType Property
 TAttribute 371
 TOraObject 782
 OCICallStyle Property
 TOraCursor 693
 TOraSession 534
 OCICallStyle Variable 688
 OCIDateTime Property 725
 OCIDateTimePtr Property 725
 OCIDLL Variable 688
 OCIInterval Property 705
 OCIIntervalPtr Property 705
 OCILobLocator Property 711
 OCILobLocatorPtr Property 711
 OCINumber Property 719
 OCINumberPtr Property 719
 OCIRef Property 792
 OCIRefPtr Property 792
 OCISmt Property 693
 OCISvcCtx Property
 TOraLob 712

TOraObject 782
 TOraSession 534
 OCIThreaded Variable 689
 OCIVersion Variable 689
 OCIVersionSt Variable 689
 OdacVcl Unit Members 412
 OdacVersion Constant 606
 Offset Property 371
 omAllRows 731
 omChoose 731
 omDefault 731
 omFirstRows 731
 omFirstRows1 731
 omFirstRows10 731
 omFirstRows100 731
 omFirstRows1000 731
 Omit Property 200
 omRule 731
 OnBackupProgress Event 169
 OnBatchMoveProgress Event 137
 OnChange Event 436
 OnConnectChange Event 546
 OnConnectionLost Event 242
 OnError Event
 TCustomDAConnection 243
 TDAAlerter 159
 TDADump 169
 TDAScript 197
 TDATransaction 343
 TOraErrorHandler 740
 TOraLoader 752
 TOraTransaction 904
 OnEvent Event 617
 OnFailover Event 546
 OnGetColumnData Event
 TDALoader 182
 TOraLoader 752
 OnMessage Event 639
 OnProgress Event 182
 OnPutData Event
 TDALoader 183
 TOraLoader 753
 OnRestoreProgress Event 170
 OnSQL Event 209
 OnTimeOut Event 617
 OnUpdateError Event 409
 OnUpdateRecord Event 409
 Open Method

- Open Method
 - TOraErrorHandler 740
 - TOraFile 701
- Operations Property 434
- OptimizerMode Property 553
- Options Property
 - TCustomDAConnection 229
 - TCustomDADataset 256
 - TCustomDASQLMonitor 208
 - TCustomSmartQuery 845
 - TDADump 164
 - TOraDataSet 451
 - TOraSession 534
 - TVirtualTable 910
- OptionsDS Property 452
- optLocal 735
- optMTS 735
- optOCI 735
- Ora Unit Members 417
- OraAlerter Unit Members 607
- OraAQ Unit Members 620
- OraCall Unit Members 685
- OraClasses Unit Members 690
- Oracle Package Wizard 100
- OracleErrorMaxLength Variable 689
- OracleVersion Property 536
- OraConnectionPool Unit Members 734
- OraDataAdapter Class 365
- OraDeveloper Tools 105
- OraErrHand Unit Members 736
- OraError Unit Members 687, 742
- OraLoader Unit Members 745
- OraObjects Unit Members 757
- OraPackage Unit Members 813
- OraProvider Unit Members 820
- OraQueryCompatibilityMode Variable 604
- OraScript Unit Members 822
- OraSmart Unit Members 831
- OraSQLMonitor Unit Members 889
- OraTools Add-In 110
- OraTransaction Unit Members 892
- OrderFields Property 871
- OriginalMessageId Property 678
- Overload Property 576
- Overview 1
- Owner Property 371

- P -

- PackageName Property 819
- PackMessage Method 614
- ParamByName Method
 - TCustomDADataset 273
 - TCustomDASQL 290
 - TDAParams 339
 - TOraDataSet 466
 - TOraSession 542
 - TOraSQL 563
- ParamCheck Property
 - TCustomDADataset 257
 - TCustomDASQL 284
- ParamCount Property
 - TCustomDADataset 257
 - TCustomDASQL 284
- Params Property
 - TCustomDADataset 258
 - TCustomDASQL 285
 - TCustomOraPackage 815
 - TDASTatement 200
 - TOraDataSet 453
 - TOraSQL 558
 - TOraStatement 828
- ParamType Property 334
- ParamValues Property(Indexer) 285
- Password Property
 - TCREncryptor 151
 - TCustomDAConnection 230
 - TOraSession 536
- PasswordLabel Property 223
- PayloadArrayType Name Property 632
- PayloadType Name Property
 - TOraQueue 633
 - TOraQueueTable 660
- Persistent Property 434
- Pin Method 794
- Ping Method 542
- PL/SQL Tables 71
- PISqlTraceComment Method 588
- PISqlTraceLimit Method 589
- PISqlTraceMode Property 586
- PISqlTracePause Method 589
- PISqlTraceResume Method 589
- PISqlTraceRunNumber Method 590
- PISqlTraceStart Method 590

PISqlTraceStop Method 590
 Pooling Property 230
 PoolingOptions Property
 TCustomDAConnection 231
 TOraSession 536
 PoolType Property 510
 Port Property
 TDBMonitorOptions 211
 TOraChangeNotification 434
 Precision Property
 TDPColumn 747
 TOraTimeStamp 725
 PrefetchRows Property 474
 Prepare Method
 TCustomDADataset 274
 TCustomDASQL 290
 TMemDataSet 405
 Prepared Property
 TCustomDASQL 286
 TMemDataSet 397
 PrepareSQL Method
 TOraStoredProc 582
 TOraTable 878
 PrepareUpdateSQL Property 475
 PrimaryInstance Property 661
 Priority Property 678
 ProblemCount Property 136
 Protocol Property 670
 ProxyPassword Property 510
 ProxySession Property 537
 ProxyUsername Property 510
 PurgePipe Method 614
 PurgeQueueTable Method 667
 PutColumnData Method 180
 PutMessage Method 614

- Q -

qdmBuffered 681
 qdmPersistent 681
 qdmPersistentOrBuffered 681
 qmgNone 682
 qmgTransactional 682
 qmsExpired 682
 qmsProcessed 682
 qmsReady 682
 qmsWaiting 682
 qnFirstMessage 682

qnFirstMessageMultiGroup 682
 qnNextMessage 682
 qnNextMessageMultiGroup 682
 qnNextTransaction 682
 qsdBefore 683
 qsdNone 683
 qsdTop 683
 qslDefault 683
 qslEnqueueTime 683
 qslEnqueueTimePriority 683
 qslPriority 683
 qslPriorityEnqueueTime 683
 qtExceptionQueue 684
 qtNormalQueue 684
 QueryRecCount Property 306
 QueueName Property
 TOraQueue 633
 TOraQueueAdmin 642
 QueueTableName Property
 TOraQueueAdmin 643
 TOraQueueTable 661
 QueueType Property 643
 QuoteNames Property
 TDADatasetOptions 306
 TDADumpOptions 171
 qvlImmediate 684
 qvOnCommit 684

- R -

RawAsString Property
 TOraDataSetOptions 475
 TOraDataSetOptionsDS 481
 RawPayload Property 674
 Read Method 377
 ReadAliases Property 415
 ReadLob Method 716
 ReadOnly Property 258
 ReadQueueProperties Method 652
 ReadQueueTableProperties Method 667
 RecipientList Property 678
 ReconnectTimeout Property 211
 RecordCount Property 136
 Ref Property 490
 RefCount Property 386
 RefIsNull Method 794
 ReflectChangeNotify Property 475
 Refresh Method

Refresh Method
 TBFileField 424
 TOraFile 701
 TOraObject 787
RefreshEvent Property 845
RefreshFields Event 858
RefreshMode Property 453
RefreshObject Property 296
RefreshOptions Property 258
RefreshRecord Method 274
RefreshSQL Property 297
RelativeMessageId Property 628
Release Method 387
RemoveFromPool Method 241
RemoveOnRefresh Property 307
RemoveRegistration Method 435
RemoveSession Method 901
RemoveSubscriber Method 652
RequiredFields Property 307
Requirements 32
Restore Method 167
RestoreFromFile Method 168
RestoreFromStream Method 168
RestoreSQL Method 275
RestoreUpdates Method 406
Restrictions Property 326
Resume Method 901
ResumeTimeout Property 897
Resync Method 275
RetentionTime Property 643
Retries Property 224
RetryDelay Property 643
ReturnParams Property
 TDADatasetOptions 307
 TOraDataSet 454
RevertRecord Method 406
RevokeQueuePrivilege Method 652
RevokeSystemPrivilege Method 668
rmAfterInsert 598
rmAfterUpdate 598
rmAlways 598
rmNone 598
rmRaise 357
rmReconnect 357
rmReconnectExecute 357
roAfterInsert 357
roAfterUpdate 357
roBeforeEdit 357

Rollback Method
 TCustomDAConnection 241
 TDATransaction 342
RollbackToSavepoint Method
 TOraSession 543
 TOraTransaction 902
RowsAffected Property
 TCustomDADataset 259
 TCustomDASQL 286
RowsProcessed Property
 TOraDataSet 454
 TOraSQL 559

- S -

SavePassword Property 224
Savepoint Method
 TOraSession 543
 TOraTransaction 902
SaveSQL Method 275
SaveToFile Method
 TBlob 377
 TVirtualTable 915
SaveToStream Method
 TBlob 378
 TOraXML 810
 TVirtualTable 915
SaveToXML Method 406
Scale Property
 TAttribute 371
 TDPColumn 748
Scan Method 349
SchedulePropagation Method 653
Schema Property 537
Script Property 200
ScrollableCursor Property
 TOraDatasetOptions 475
 TOraDatasetOptionsDS 481
SecondaryInstance Property 661
Secure Property 662
SendDataSetChangeEventAfterOpen
Variable 411
Sender Property 744
SenderId Property 679
SendEvent Method 158
SendMessage Method 615
SendPipeMessage Method 615
SendTimeout Property 211

- SequenceDeviation Property 628
- SequenceMode Property 454
- Server Property
 - TCustomDAConnection 231
 - TOraSession 537
- ServerLabel Property 224
- Session Property
 - TConnectDialog 415
 - TCustomOraPackage 815
 - TOraAlerter 611
 - TOraDataSet 455
 - TOraErrorHandler 739
 - TOraLoader 751
 - TOraQueue 633
 - TOraQueueAdmin 644
 - TOraQueueTable 662
 - TOraScript 826
 - TOraSQL 559
 - TOraTrace 586
- SessionName Property 125
- Sessions Property(Indexer) 897
- Sessions Variable 604
- SessionsCount Property 898
- SetBlobData 337
- SetBlobData Method
 - TDAParam 337
 - TOraParam 507
- SetDate Method 728
- SetDaySecond Method 707
- SetFieldsReadOnly Property 307
- SetKey Method 151
- SetOrderBy Method 276
- SetTime Method 728
- SetTimeZoneOffset Method 729
- SetYearMonth Method 708
- Size Property
 - TAttribute 372
 - TBlob 375
 - TDAParam 334
 - TDPColumn 748
 - TObjectType 384
 - TOraArray 766
- SmartRefresh Property 846
- SmartState Property 846
- smInsert 599
- smPost 599
- SortList Property 662
- Source Property 137
- SQL Property
 - TCustomDADataset 259
 - TCustomDASQL 286
 - TDADump 165
 - TDAScript 191
 - TDASentence 200
 - TOraSession 538
- SQL Property(Indexer) 297
- SQLDelete Property 259
- SQLGeneratorCompatibility Variable 359
- SQLInsert Property 260
- SQLLock Property 260
- SQLRefresh Property 261
- SQLSaved Method 276
- SqlTraceMode Property 586
- SqlTraceStart Method 591
- SqlTraceStop Method 591
- SQLType Property
 - TOraDataSet 455
 - TOraSQL 559
- SQLUpdate Property 261
- Start Method 158
- StartDequeue Method 653
- StartEnqueue Method 654
- StartLine Property
 - TDAScript 191
 - TDASentence 201
- StartOffset Property
 - TDAScript 192
 - TDASentence 201
- StartPos Property
 - TDAScript 192
 - TDASentence 201
- StartQueue Method 654
- StartTransaction Method
 - StartTransaction 544, 903
 - TCustomDAConnection 242
 - TDATransaction 342
 - TOraSession 543
 - TOraTransaction 902
- State Property
 - TOraTrace 587
 - TQueueMessageProperties 679
- StatementCache Property
 - TOraDataSetOptions 476
 - TOraSessionOptions 553
 - TOraSQL 560
- StatementCacheSize Property 553

Statements Property
 TDAScript 192
 TOraScript 826
stBinary 390
stCaseInsensitive 390
stCaseSensitive 390
Stop Method 159
StopDequeue Method 655
StopEnqueue Method 655
StopQueue Method 655
StorageClause Property 662
StoredProcName Property 576
StoreLogInfo Property 224
StrictUpdate Property
 TDADatasetOptions 308
 TOraDataSet 455
StringPayload Property 674

- T -

Table Property
 TOraNestedTable 491
 TOraParam 505
TableName Property
 TDALoader 178
 TOraErrorHandler 739
 TOraTable 872
TableNames Property 165
taCommit 129
TAfterExecuteEvent Procedure Reference 352
TAfterFetchEvent Procedure Reference 352
TAfterStatementExecuteEvent Procedure Reference 203
TAlertErrorEvent Procedure Reference 160
taRollback 129
TAttribute Class 368
TAttribute.AttributeNo Property 369
TAttribute.DataSize Property 370
TAttribute.DataType Property 370
TAttribute.Length Property 370
TAttribute.ObjectType Property 371
TAttribute.Offset Property 371
TAttribute.Owner Property 371
TAttribute.Scale Property 371
TAttribute.Size Property 372

TBDESession Class 119
TBDESession.Database Property 125
TBDESession.DatabaseName Property 125
TBDESession.SessionName Property 125
TBeforeFetchEvent Procedure Reference 353
TBeforeFetchProc Procedure Reference 128
TBeforeStatementExecuteEvent Procedure Reference 203
TBFileField Class 421
TBFileField.AsFile Property 422
TBFileField.AutoRefresh Property 422
TBFileField.BlobType Property 423
TBFileField.Exists Property 423
TBFileField.FileDir Property 423
TBFileField.FileName Property 424
TBFileField.Refresh Method 424
TBlob Class 372
TBlob.Assign Method 376
TBlob.AsString Property 374
TBlob.AsWideString Property 374
TBlob.Clear Method 376
TBlob.IsUnicode Property 374
TBlob.LoadFromFile Method 376
TBlob.LoadFromStream Method 377
TBlob.Read Method 377
TBlob.SaveToFile Method 377
TBlob.SaveToStream Method 378
TBlob.Size Property 375
TBlob.Truncate Method 378
TBlob.Write Method 379
TCheckMode Enumeration 597
TCompressedBlob Class 379
TConnectChangeEvent Procedure Reference 595
TConnectDialog Class 413
TConnectDialog.ReadAliases Property 415
TConnectDialog.Session Property 415
TConnectionLostEvent Procedure Reference 353
TConnectMode Enumeration 730
TConnLostCause Enumeration 389
TCRBatchMode Enumeration 140
TCRBatchMove Class 131
TCRBatchMove.AbortOnKeyViol Property 133

- TCRBatchMove.AbortOnProblem Property 133
- TCRBatchMove.ChangedCount Property 134
- TCRBatchMove.CommitCount Property 134
- TCRBatchMove.Destination Property 134
- TCRBatchMove.Execute Method 137
- TCRBatchMove.FieldMappingMode Property 134
- TCRBatchMove.KeyViolCount Property 135
- TCRBatchMove.Mappings Property 135
- TCRBatchMove.Mode Property 135
- TCRBatchMove.MovedCount Property 136
- TCRBatchMove.OnBatchMoveProgress Event 137
- TCRBatchMove.ProblemCount Property 136
- TCRBatchMove.RecordCount Property 136
- TCRBatchMove.Source Property 137
- TCRBatchMoveProgressEvent Procedure Reference 139
- TCRCursor Class 127
- TCRDataSource Class 220
- TCREncDataHeader Enumeration 153
- TCREncryptionAlgorithm Enumeration 153
- TCREncryptor Class 149
- TCREncryptor.DataHeader Property 150
- TCREncryptor.EncryptionAlgorithm Property 150
- TCREncryptor.HashAlgorithm Property 150
- TCREncryptor.InvalidHashAction Property 151
- TCREncryptor.Password Property 151
- TCREncryptor.SetKey Method 151
- TCRFieldMappingMode Enumeration 140
- TCRHashAlgorithm Enumeration 154
- TCRInvalidHashAction Enumeration 154
- TCRIsoLevel Enumeration 129
- TCRTransactionAction Enumeration 129
- TCursorField Class 424
- TCursorField.AsCursor Property 425
- TCustomConnectDialog Class 220
- TCustomConnectDialog.CancelButton Property 222
- TCustomConnectDialog.Caption Property 222
- TCustomConnectDialog.ConnectButton Property 223
- TCustomConnectDialog.DialogClass Property 223
- TCustomConnectDialog.Execute Method 225
- TCustomConnectDialog.GetServerList Method 225
- TCustomConnectDialog.LabelSet Property 223
- TCustomConnectDialog.PasswordLabel Property 223
- TCustomConnectDialog.Retries Property 224
- TCustomConnectDialog.SavePassword Property 224
- TCustomConnectDialog.ServerLabel Property 224
- TCustomConnectDialog.StoreLogInfo Property 224
- TCustomConnectDialog.UsernameLabel Property 225
- TCustomDAConnection Class 226
- TCustomDAConnection.ApplyUpdates Method 233
- TCustomDAConnection.Commit Method 234
- TCustomDAConnection.Connect Method 234
- TCustomDAConnection.ConnectDialog Property 228
- TCustomDAConnection.ConvertEOL Property 228
- TCustomDAConnection.CreateDataSet Method 235
- TCustomDAConnection.CreateSQL Method 235
- TCustomDAConnection.Disconnect Method 235
- TCustomDAConnection.ExecProc Method 236
- TCustomDAConnection.ExecProcEx Method 237
- TCustomDAConnection.ExecSQL Method 238
- TCustomDAConnection.ExecSQLEx Method 238
- TCustomDAConnection.GetDatabaseNames Method 239
- TCustomDAConnection.GetStoredProcNames Method 240

- TCustomDAConnection.GetTableNames Method 240
- TCustomDAConnection.InTransaction Property 228
- TCustomDAConnection.LoginPrompt Property 229
- TCustomDAConnection.MonitorMessage Method 241
- TCustomDAConnection.OnConnectionLost Event 242
- TCustomDAConnection.OnError Event 243
- TCustomDAConnection.Options Property 229
- TCustomDAConnection.Password Property 230
- TCustomDAConnection.Pooling Property 230
- TCustomDAConnection.PoolingOptions Property 231
- TCustomDAConnection.RemoveFromPool Method 241
- TCustomDAConnection.Rollback Method 241
- TCustomDAConnection.Server Property 231
- TCustomDAConnection.StartTransaction Method 242
- TCustomDAConnection.Username Property 232
- TCustomDADataset Class 243
- TCustomDADataset.AddWhere Method 265
- TCustomDADataset.AfterExecute Event 278
- TCustomDADataset.AfterFetch Event 279
- TCustomDADataset.AfterUpdateExecute Event 279
- TCustomDADataset.BaseSQL Property 250
- TCustomDADataset.BeforeFetch Event 279
- TCustomDADataset.BeforeUpdateExecute Event 280
- TCustomDADataset.BreakExec Method 265
- TCustomDADataset.Connection Property 251
- TCustomDADataset.CreateBlobStream Method 266
- TCustomDADataset.Debug Property 251
- TCustomDADataset.DeleteWhere Method 266
- TCustomDADataset.DetailFields Property 251
- TCustomDADataset.Disconnected Property 252
- TCustomDADataset.Encryption Property 252
- TCustomDADataset.Execute Method 266
- TCustomDADataset.Executing Method 267
- TCustomDADataset.Fetched Method 267
- TCustomDADataset.Fetching Method 267
- TCustomDADataset.FetchingAll Method 268
- TCustomDADataset.FetchRows Property 252
- TCustomDADataset.FilterSQL Property 252
- TCustomDADataset.FinalSQL Property 253
- TCustomDADataset.FindKey Method 268
- TCustomDADataset.FindMacro Method 269
- TCustomDADataset.FindNearest Method 269
- TCustomDADataset.FindParam Method 269
- TCustomDADataset.GetData Type Method 270
- TCustomDADataset.GetFieldObject Method 270
- TCustomDADataset.GetFieldPrecision Method 271
- TCustomDADataset.GetFieldScale Method 271
- TCustomDADataset.GetOrderBy Method 272
- TCustomDADataset.GotoCurrent Method 272
- TCustomDADataset.IsQuery Property 253
- TCustomDADataset.KeyFields Property 253
- TCustomDADataset.Lock Method 272
- TCustomDADataset.MacroByName Method 273
- TCustomDADataset.MacroCount Property 254
- TCustomDADataset.Macros Property 254
- TCustomDADataset.MasterFields Property 255
- TCustomDADataset.MasterSource Property 255
- TCustomDADataset.Options Property 256

- TCustomDADataset.ParamByName Method 273
- TCustomDADataset.ParamCheck Property 257
- TCustomDADataset.ParamCount Property 257
- TCustomDADataset.Params Property 258
- TCustomDADataset.Prepare Method 274
- TCustomDADataset.ReadOnly Property 258
- TCustomDADataset.RefreshOptions Property 258
- TCustomDADataset.RefreshRecord Method 274
- TCustomDADataset.RestoreSQL Method 275
- TCustomDADataset.Resync Method 275
- TCustomDADataset.RowsAffected Property 259
- TCustomDADataset.SaveSQL Method 275
- TCustomDADataset.SetOrderBy Method 276
- TCustomDADataset.SQL Property 259
- TCustomDADataset.SQLDelete Property 259
- TCustomDADataset.SQLInsert Property 260
- TCustomDADataset.SQLLock Property 260
- TCustomDADataset.SQLRefresh Property 261
- TCustomDADataset.SQLSaved Method 276
- TCustomDADataset.SQLUpdate Property 261
- TCustomDADataset.UniDirectional Property 262
- TCustomDADataset.Unlock Method 277
- TCustomDASQL Class 280
- TCustomDASQL.AfterExecute Event 292
- TCustomDASQL.ChangeCursor Property 282
- TCustomDASQL.Connection Property 282
- TCustomDASQL.Debug Property 283
- TCustomDASQL.Execute Method 287
- TCustomDASQL.Executing Method 288
- TCustomDASQL.FinalSQL Property 283
- TCustomDASQL.FindMacro Method 288
- TCustomDASQL.FindParam Method 289
- TCustomDASQL.MacroByName Method 289
- TCustomDASQL.MacroCount Property 283
- TCustomDASQL.Macros Property 284
- TCustomDASQL.ParamByName Method 290
- TCustomDASQL.ParamCheck Property 284
- TCustomDASQL.ParamCount Property 284
- TCustomDASQL.Params Property 285
- TCustomDASQL.ParamValues Property(Indexer) 285
- TCustomDASQL.Prepare Method 290
- TCustomDASQL.Prepared Property 286
- TCustomDASQL.RowsAffected Property 286
- TCustomDASQL.SQL Property 286
- TCustomDASQL.UnPrepare Method 291
- TCustomDASQL.WaitExecuting Method 291
- TCustomDASQLMonitor Class 207
- TCustomDASQLMonitor.Active Property 208
- TCustomDASQLMonitor.DBMonitorOptions Property 208
- TCustomDASQLMonitor.OnSQL Event 209
- TCustomDASQLMonitor.Options Property 208
- TCustomDASQLMonitor.TraceFlags Property 209
- TCustomDAUpdateSQL Class 292
- TCustomDAUpdateSQL.Apply Method 298
- TCustomDAUpdateSQL.DataSet Property 294
- TCustomDAUpdateSQL.DeleteObject Property 294
- TCustomDAUpdateSQL.DeleteSQL Property 294
- TCustomDAUpdateSQL.ExecSQL Method 298
- TCustomDAUpdateSQL.InsertObject Property 295
- TCustomDAUpdateSQL.InsertSQL Property 295
- TCustomDAUpdateSQL.LockObject Property 295
- TCustomDAUpdateSQL.LockSQL Property 296

- TCustomDAUpdateSQL.ModifyObject Property 296
- TCustomDAUpdateSQL.ModifySQL Property 296
- TCustomDAUpdateSQL.RefreshObject Property 296
- TCustomDAUpdateSQL.RefreshSQL Property 297
- TCustomDAUpdateSQL.SQL Property(Indexer) 297
- TCustomOraPackage Class 814
- TCustomOraPackage.ExecProc Method 816
- TCustomOraPackage.ExecProcEx Method 816
- TCustomOraPackage.Params Property 815
- TCustomOraPackage.Session Property 815
- TCustomOraPackage.VariableByName Method 817
- TCustomOraQuery Class 426
- TCustomSmartQuery Class 832
- TCustomSmartQuery.AfterSmartRefresh Event 857
- TCustomSmartQuery.DependEvents Property 844
- TCustomSmartQuery.Expand Property 844
- TCustomSmartQuery.Options Property 845
- TCustomSmartQuery.RefreshEvent Property 845
- TCustomSmartQuery.RefreshFields Event 858
- TCustomSmartQuery.SmartRefresh Property 846
- TCustomSmartQuery.SmartState Property 846
- TCustomSmartQuery.View Method 852
- TDAAlerter Class 156
- TDAAlerter.Active Property 157
- TDAAlerter.AutoRegister Property 157
- TDAAlerter.Connection Property 157
- TDAAlerter.OnError Event 159
- TDAAlerter.SendEvent Method 158
- TDAAlerter.Start Method 158
- TDAAlerter.Stop Method 159
- TDABackupProgressEvent Procedure Reference 172
- TDAColumn Class 174
- TDAColumn.FieldType Property 175
- TDAColumn.Name Property 175
- TDAColumns Class 175
- TDAColumns.Items Property(Indexer) 176
- TDAConnectionErrorEvent Procedure Reference 353
- TDAConnectionOptions Class 299
- TDAConnectionOptions.DefaultSortType Property 300
- TDAConnectionOptions.DisconnectedMode Property 300
- TDAConnectionOptions.KeepDesignConnected Property 300
- TDAConnectionOptions.LocalFailover Property 301
- TDADatasetOptions Class 301
- TDADatasetOptions.AutoPrepare Property 304
- TDADatasetOptions.CacheCalcFields Property 304
- TDADatasetOptions.DefaultValues Property 304
- TDADatasetOptions.DetailDelay Property 304
- TDADatasetOptions.FieldsOrigin Property 305
- TDADatasetOptions.FlatBuffers Property 305
- TDADatasetOptions.LocalMasterDetail Property 305
- TDADatasetOptions.LongStrings Property 306
- TDADatasetOptions.NumberRange Property 306
- TDADatasetOptions.QueryRecCount Property 306
- TDADatasetOptions.QuoteNames Property 306
- TDADatasetOptions.RemoveOnRefresh Property 307
- TDADatasetOptions.RequiredFields Property 307
- TDADatasetOptions.ReturnParams Property 307
- TDADatasetOptions.SetFieldsReadOnly Property 307
- TDADatasetOptions.StrictUpdate Property 308
- TDADatasetOptions.TrimFixedChar Property 308
- TDADatasetOptions.UpdateAllFields Property 308

- TDADatasetOptions.UpdateBatchSize Property 308
- TDADump Class 162
- TDADump.Backup Method 166
- TDADump.BackupQuery Method 166
- TDADump.BackupToFile Method 166
- TDADump.BackupToStream Method 167
- TDADump.Connection Property 163
- TDADump.Debug Property 164
- TDADump.OnBackupProgress Event 169
- TDADump.OnError Event 169
- TDADump.OnRestoreProgress Event 170
- TDADump.Options Property 164
- TDADump.Restore Method 167
- TDADump.RestoreFromFile Method 168
- TDADump.RestoreFromStream Method 168
- TDADump.SQL Property 165
- TDADump.TableNames Property 165
- TDADumpOptions Class 170
- TDADumpOptions.AddDrop Property 171
- TDADumpOptions.GenerateHeader Property 171
- TDADumpOptions.QuoteNames Property 171
- TDAEncryptionOptions Class 309
- TDAEncryptionOptions.Encryptor Property 310
- TDAEncryptionOptions.Fields Property 310
- TDALoader Class 176
- TDALoader.Columns Property 178
- TDALoader.Connection Property 178
- TDALoader.CreateColumns Method 179
- TDALoader.Load Method 180
- TDALoader.LoadFromDataSet Method 180
- TDALoader.OnGetColumnData Event 182
- TDALoader.OnProgress Event 182
- TDALoader.OnPutData Event 183
- TDALoader.PutColumnData Method 180
- TDALoader.TableName Property 178
- TDAMapRule Class 310
- TDAMapRule.DBLengthMax Property 311
- TDAMapRule.DBLengthMin Property 312
- TDAMapRule.DBScaleMax Property 312
- TDAMapRule.DBScaleMin Property 312
- TDAMapRule.DBType Property 312
- TDAMapRule.FieldLength Property 313
- TDAMapRule.FieldName Property 313
- TDAMapRule.FieldScale Property 313
- TDAMapRule.FieldType Property 313
- TDAMapRule.IgnoreErrors Property 314
- TDAMapRules Class 314
- TDAMapRules.AddDBTypeRule Method 315
- TDAMapRules.AddFieldNameRule Method 319
- TDAMapRules.AddRule Method 321
- TDAMetaData Class 321
- TDAMetaData.Connection Property 325
- TDAMetaData.GetMetaDataKinds Method 328
- TDAMetaData.GetRestrictions Method 328
- TDAMetaData.MetaDataKind Property 325
- TDAMetaData.Restrictions Property 326
- TDANumericType Enumeration 390
- TDAParam Class 328
- TDAParam.AsBlob Property 331
- TDAParam.AsBlobRef Property 331
- TDAParam.AsFloat Property 331
- TDAParam.AsInteger Property 331
- TDAParam.AsLargeInt Property 332
- TDAParam.AsMemo Property 332
- TDAParam.AsMemoRef Property 332
- TDAParam.AssignField Method 335
- TDAParam.AssignFieldValue Method 335
- TDAParam.AsSQLTimeStamp Property 333
- TDAParam.AsString Property 333
- TDAParam.AsWideString Property 333
- TDAParam.DataType Property 333
- TDAParam.IsNull Property 334
- TDAParam.LoadFromFile Method 336
- TDAParam.LoadFromStream Method 336
- TDAParam.ParamType Property 334
- TDAParam.SetBlobData 337
- TDAParam.SetBlobData Method 337
- TDAParam.Size Property 334
- TDAParam.Value Property 334
- TDAParams Class 337
- TDAParams.FindParam Method 339
- TDAParams.Items Property (Indexer) 338
- TDAParams.ParamByName Method 339
- TDAPutDataEvent Procedure Reference 184

- TDARestoreProgressEvent Procedure Reference 172
- TDAScript Class 187
- TDAScript.AfterExecute Event 197
- TDAScript.BeforeExecute Event 197
- TDAScript.BreakExec Method 193
- TDAScript.Connection Property 189
- TDAScript.DataSet Property 189
- TDAScript.Debug Property 190
- TDAScript.Delimiter Property 190
- TDAScript.EndLine Property 190
- TDAScript.EndOffset Property 190
- TDAScript.EndPos Property 191
- TDAScript.ErrorOffset Method 194
- TDAScript.Execute Method 194
- TDAScript.ExecuteFile Method 194
- TDAScript.ExecuteNext Method 195
- TDAScript.ExecuteStream Method 195
- TDAScript.FindMacro Method 195
- TDAScript.MacroByName Method 196
- TDAScript.Macros Property 191
- TDAScript.OnError Event 197
- TDAScript.SQL Property 191
- TDAScript.StartLine Property 191
- TDAScript.StartOffset Property 192
- TDAScript.StartPos Property 192
- TDAScript.Statements Property 192
- TDASTatement Class 197
- TDASTatement.EndLine Property 199
- TDASTatement.EndOffset Property 199
- TDASTatement.EndPos Property 199
- TDASTatement.Omit Property 200
- TDASTatement.Params Property 200
- TDASTatement.Script Property 200
- TDASTatement.SQL Property 200
- TDASTatement.StartLine Property 201
- TDASTatement.StartOffset Property 201
- TDASTatement.StartPos Property 201
- TDASTatements Class 201
- TDASTatements.Items Property(Indexer) 202
- TDATraceFlag Enumeration 213
- TDATraceFlags Set 212
- TDATransaction Class 339
- TDATransaction.Active Property 341
- TDATransaction.Commit Method 341
- TDATransaction.DefaultCloseAction Property 341
- TDATransaction.OnError Event 343
- TDATransaction.Rollback Method 342
- TDATransaction.StartTransaction Method 342
- TDATransactionErrorEvent Procedure Reference 354
- TDBMonitorOptions Class 210
- TDBMonitorOptions.Host Property 211
- TDBMonitorOptions.Port Property 211
- TDBMonitorOptions.ReconnectTimeout Property 211
- TDBMonitorOptions.SendTimeout Property 211
- TDBObject Class 381
- TDequeueMode Enumeration 681
- TDequeueOptions Class 622
- TDequeueOptions.ConsumerName Property 624
- TDequeueOptions.Correlation Property 624
- TDequeueOptions.DeliveryMode Property 624
- TDequeueOptions.DequeueCondition Property 625
- TDequeueOptions.DequeueMode Property 625
- TDequeueOptions.MessageId Property 625
- TDequeueOptions.Navigation Property 625
- TDequeueOptions.Transformation Property 625
- TDequeueOptions.Visibility Property 626
- TDequeueOptions.WaitTimeout Property 626
- TDO Property 797
- TDPColumn Class 746
- TDPColumn.DateFormat Property 747
- TDPColumn.Precision Property 747
- TDPColumn.Scale Property 748
- TDPColumn.Size Property 748
- TDPErroAction Enumeration 756
- TDPErroEvent Procedure Reference 754
- TDPGetColumnDataEvent Procedure Reference 754
- TDPPutDataEvent Procedure Reference 755
- TemporaryLobUpdate Property
 - TOraDataSetOptions 476
 - TOraSQL 560
- TEnqueueOptions Class 626

- TEnqueueOptions.DeliveryMode Property 628
- TEnqueueOptions.RelativeMessageId Property 628
- TEnqueueOptions.SequenceDeviation Property 628
- TEnqueueOptions.Transformation Property 629
- TEnqueueOptions.Visibility Property 629
- TErrorAction Enumeration 205
- TEventType Enumeration 619
- TFailoverEvent Procedure Reference 595
- TFailoverState Enumeration 597
- TFailoverType Enumeration 598
- tfBlob 213
- tfConnect 213
- tfError 213
- tfMisc 213
- tfObjDestroy 213
- tfParams 213
- tfPool 213
- tfQExecute 213
- tfQFetch 213
- tfQPrepare 213
- tfService 213
- tfStmt 213
- tfTransact 213
- TGetColumnDataEvent Procedure Reference 184
- TGlobalCoordinator Enumeration 905
- ThreadSafety Property 538
- TimeOut Property
 - TOraAlerter 612
 - TOraChangeNotification 434
- TimeZone Property 725
- TLabelSet Enumeration 356
- TLoaderProgressEvent Procedure Reference 185
- TLoadMode Enumeration 756
- TLocateExOption Enumeration 390
- TLocateExOptions Set 388
- TLockMode Enumeration 356
- TMacro Class 343
- TMacro.Active Property 344
- TMacro.AsDateTime Property 345
- TMacro.AsFloat Property 345
- TMacro.AsInteger Property 345
- TMacro.AsString Property 345
- TMacro.Name Property 346
- TMacro.Value Property 346
- TMacros Class 346
- TMacros.AssignValues Method 348
- TMacros.FindMacro Method 348
- TMacros.IsEqual Method 349
- TMacros.Items Property(Indexer) 347
- TMacros.MacroByName Method 349
- TMacros.Scan Method 349
- TMapRule Class 144
- TMapRule.DBLengthMax Property 145
- TMapRule.DBLengthMin Property 145
- TMapRule.DBScaleMax Property 145
- TMapRule.DBScaleMin Property 146
- TMapRule.DBType Property 146
- TMapRule.FieldLength Property 146
- TMapRule.FieldName Property 146
- TMapRule.FieldScale Property 146
- TMapRule.IgnoreErrors Property 146
- TMemDataSet Class 393
- TMemDataSet.ApplyUpdates Method 399
- TMemDataSet.CachedUpdates Property 395
- TMemDataSet.CancelUpdates Method 400
- TMemDataSet.CommitUpdates Method 401
- TMemDataSet.DeferredPost Method 401
- TMemDataSet.GetBlob Method 402
- TMemDataSet.IndexFieldNames Property 396
- TMemDataSet.LocalConstraints Property 396
- TMemDataSet.LocalUpdate Property 396
- TMemDataSet.Locate Method 403
- TMemDataSet.LocateEx Method 404
- TMemDataSet.OnUpdateError Event 409
- TMemDataSet.OnUpdateRecord Event 409
- TMemDataSet.Prepare Method 405
- TMemDataSet.Prepared Property 397
- TMemDataSet.RestoreUpdates Method 406
- TMemDataSet.RevertRecord Method 406
- TMemDataSet.SaveToXML Method 407
- TMemDataSet.UnPrepare Method 407
- TMemDataSet.UpdateRecordTypes Property 397
- TMemDataSet.UpdateResult Method 408
- TMemDataSet.UpdatesPending Property 397

- TMemDataSet.UpdateStatus Method 408
- TMessageType Enumeration 730
- TMonitorOption Enumeration 213
- TMonitorOptions Set 212
- TObjectType Class 381
- TObjectType.AttributeByName Method 384
- TObjectType.AttributeCount Property 383
- TObjectType.Attributes Property(Indexer) 383
- TObjectType.DataType Property 383
- TObjectType.FindAttribute Method 385
- TObjectType.Size Property 384
- TOnErrorEvent Procedure Reference 203
- TOnEventEvent Procedure Reference 618
- TOnOraErrorEvent Procedure Reference 741
- TOnSQLEvent Procedure Reference 212
- TOnTimeoutEvent Procedure Reference 618
- TOptimizerMode Enumeration 731
- TOraAlerter Class 608
- TOraAlerter.AutoCommit Property 610
- TOraAlerter.Events Property 611
- TOraAlerter.EventType Property 611
- TOraAlerter.GetMessage Method 613
- TOraAlerter.Interval Property 611
- TOraAlerter.NextItemType Method 613
- TOraAlerter.NextMessageType Method 613
- TOraAlerter.OnEvent Event 617
- TOraAlerter.OnTimeout Event 617
- TOraAlerter.PackMessage Method 614
- TOraAlerter.PurgePipe Method 614
- TOraAlerter.PutMessage Method 614
- TOraAlerter.SendMessage Method 615
- TOraAlerter.SendPipeMessage Method 615
- TOraAlerter.Session Property 611
- TOraAlerter.Timeout Property 612
- TOraAlerter.UnpackMessage Method 616
- TOraArray Class 758
- TOraArray.AppendItem Method 768
- TOraArray.Clear Method 768
- TOraArray.InsertItem Method 768
- TOraArray.ItemAsDateTime Property(Indexer) 763
- TOraArray.ItemAsFloat Property(Indexer) 763
- TOraArray.ItemAsInteger Property(Indexer) 763
- TOraArray.ItemAsObject Property(Indexer) 764
- TOraArray.ItemAsOCIString Property(Indexer) 764
- TOraArray.ItemAsString Property(Indexer) 764
- TOraArray.ItemExists Property(Indexer) 765
- TOraArray.ItemIsNull Property(Indexer) 765
- TOraArray.ItemType Property 765
- TOraArray.MaxSize Property 765
- TOraArray.Size Property 766
- TOraChangeNotification Class 432
- TOraChangeNotification Component 56
- TOraChangeNotification.Active Property 433
- TOraChangeNotification.Enabled Property 433
- TOraChangeNotification.OnChange Event 436
- TOraChangeNotification.Operations Property 434
- TOraChangeNotification.Persistent Property 434
- TOraChangeNotification.Port Property 434
- TOraChangeNotification.RemoveRegistration Method 435
- TOraChangeNotification.Timeout Property 434
- TOraChangeNotificationEvent Procedure Reference 596
- TOraCursor Class 691
- TOraCursor.AllocCursor Method 693
- TOraCursor.CanFetch Method 694
- TOraCursor.CDA Property 692
- TOraCursor.FreeCursor Method 694
- TOraCursor.OCICallStyle Property 693
- TOraCursor.OCIStmt Property 693
- TOraDataSet Class 436
- TOraDataSet.ChangeNotification Property 447
- TOraDataSet.CheckMode Property 447
- TOraDataSet.CreateProcCall Method 460
- TOraDataSet.Cursor Property 448
- TOraDataSet.DMLRefresh Property 448
- TOraDataSet.ErrorOffset Method 461
- TOraDataSet.FetchAll Property 448

- ToraDataSet.Fetched Method 461
- ToraDataSet.FindParam Method 461
- ToraDataSet.GetArray Method 462
- ToraDataSet.GetFile Method 462
- ToraDataSet.GetInterval Method 463
- ToraDataSet.GetKeyList Method 463
- ToraDataSet.GetLob Method 464
- ToraDataSet.GetLobLocator Method 464
- ToraDataSet.GetObject Method 465
- ToraDataSet.GetRef Method 465
- ToraDataSet.GetTable Method 466
- ToraDataSet.GetTimeStamp Method 466
- ToraDataSet.IsPLSQL Property 449
- ToraDataSet.IsQuery Property 449
- ToraDataSet.KeyFields Property 449
- ToraDataSet.KeySequence Property 450
- ToraDataSet.LockMode Property 450
- ToraDataSet.NonBlocking Property 451
- ToraDataSet.Options Property 451
- ToraDataSet.OptionsDS Property 452
- ToraDataSet.ParamByName Method 466
- ToraDataSet.Params Property 453
- ToraDataSet.RefreshMode Property 453
- ToraDataSet.ReturnParams Property 454
- ToraDataSet.RowsProcessed Property 454
- ToraDataSet.SequenceMode Property 454
- ToraDataSet.Session Property 455
- ToraDataSet.SQLType Property 455
- ToraDataSet.StrictUpdate Property 455
- ToraDataSet.UpdateObject Property 455
- ToraDataSetField Class 467
- ToraDataSetField.Modified Property 468
- ToraDataSetOptions Class 468
- ToraDataSetOptions.AutoClose Property 473
- ToraDataSetOptions.CacheLobs Property 473
- ToraDataSetOptions.DefaultValues Property 473
- ToraDataSetOptions.DeferredLobRead Property 473
- ToraDataSetOptions.ExtendedFieldsInfo Property 474
- ToraDataSetOptions.FieldsAsString Property 474
- ToraDataSetOptions.FullRefresh Property 474
- ToraDataSetOptions.PrefetchRows Property 474
- ToraDataSetOptions.PrepareUpdateSQL Property 475
- ToraDataSetOptions.RawAsString Property 475
- ToraDataSetOptions.ReflectChangeNotify Property 475
- ToraDataSetOptions.ScrollableCursor Property 475
- ToraDataSetOptions.StatementCache Property 476
- ToraDataSetOptions.TemporaryLobUpdate Property 476
- ToraDataSetOptionsDS Class 476
- ToraDataSetOptionsDS.AutoClose Property 480
- ToraDataSetOptionsDS.CacheLobs Property 480
- ToraDataSetOptionsDS.DefaultValues Property 480
- ToraDataSetOptionsDS.DeferredLobRead Property 481
- ToraDataSetOptionsDS.FieldsAsString Property 481
- ToraDataSetOptionsDS.KeepPrepared Property 481
- ToraDataSetOptionsDS.RawAsString Property 481
- ToraDataSetOptionsDS.ScrollableCursor Property 481
- ToraDataSource Class 482
- ToraEncryptor Class 482
- ToraErrorHandler Class 737
- ToraErrorHandler.Active Property 738
- ToraErrorHandler.Close Method 739
- ToraErrorHandler.Debug Property 738
- ToraErrorHandler.OnError Event 740
- ToraErrorHandler.Open Method 740
- ToraErrorHandler.Session Property 739
- ToraErrorHandler.TableName Property 739
- ToraFile Class 694
- ToraFile.Close Method 700
- ToraFile.Exists Method 700
- ToraFile.FileDir Property 698
- ToraFile.FileName Property 698
- ToraFile.IsOpen Method 701
- ToraFile.Open Method 701
- ToraFile.Refresh Method 701

- ToraInterval Class 702
- ToraInterval.AllocInterval Method 706
- ToraInterval.AsString Property 703
- ToraInterval.Compare Method 706
- ToraInterval.DescriptorType Property 704
- ToraInterval.FracPrecision Property 704
- ToraInterval.FreeInterval Method 707
- ToraInterval.IsNull Property 704
- ToraInterval.LeadPrecision Property 705
- ToraInterval.OCIInterval Property 705
- ToraInterval.OCIIntervalPtr Property 705
- ToraInterval.SetDaySecond Method 707
- ToraInterval.SetYearMonth Method 708
- ToraIntervalField Class 483
- ToraIntervalField.AsInterval Property 484
- ToraIntervalField.FracPrecision Property 484
- ToraIntervalField.LeadPrecision Property 485
- ToraIsolationLevel Enumeration 598
- ToraLoader Class 748
- ToraLoader Component 51
- ToraLoader.LoadMode Property 750
- ToraLoader.OnError Event 752
- ToraLoader.OnGetColumnData Event 752
- ToraLoader.OnPutData Event 753
- ToraLoader.Session Property 751
- ToraLob Class 708
- ToraLob.AllocLob Method 713
- ToraLob.Cached Property 711
- ToraLob.CreateTemporary Method 713
- ToraLob.DisableBuffering Method 714
- ToraLob.EnableBuffering Method 714
- ToraLob.FreeLob Method 714
- ToraLob.FreeTemporary Method 714
- ToraLob.Init Method 715
- ToraLob.IsInit Method 715
- ToraLob.IsTemporary Method 715
- ToraLob.LengthLob Method 715
- ToraLob.OCIlobLocator Property 711
- ToraLob.OCIlobLocatorPtr Property 711
- ToraLob.OCISvcCtx Property 712
- ToraLob.ReadLob Method 716
- ToraLob.WriteLob Method 716
- ToraMetaData Class 485
- ToraNestedTable Class 487
- ToraNestedTable.Ref Property 490
- ToraNestedTable.Table Property 491
- ToraNestTable Class 769
- ToraNestTable.DeleteItem Method 774
- ToraNumber Class 716
- ToraNumber.AsFloat Property 718
- ToraNumber.AsInteger Property 718
- ToraNumber.AsLargeInt Property 718
- ToraNumber.AssignTo Method 720
- ToraNumber.AsString Property 719
- ToraNumber.Compare Method 720
- ToraNumber.IsNull Property 719
- ToraNumber.OCINumber Property 719
- ToraNumber.OCINumberPtr Property 719
- ToraNumberField Class 491
- ToraNumberField.AsNumber Property 492
- ToraObject Class 774
- ToraObject.AllocObject Method 783
- ToraObject.Assign Method 784
- ToraObject.AttrAsArray Property(Indexer) 777
- ToraObject.AttrAsDateTime Property(Indexer) 778
- ToraObject.AttrAsFloat Property(Indexer) 778
- ToraObject.AttrAsInteger Property(Indexer) 778
- ToraObject.AttrAsLob Property(Indexer) 778
- ToraObject.AttrAsObject Property(Indexer) 779
- ToraObject.AttrAsOCIDate Property(Indexer) 779
- ToraObject.AttrAsOCINumber Property(Indexer) 780
- ToraObject.AttrAsOCIString Property(Indexer) 780
- ToraObject.AttrAsString Property(Indexer) 780
- ToraObject.AttrIsNull Property(Indexer) 781
- ToraObject.CreateObject Method 784
- ToraObject.Exists Method 785
- ToraObject.Flush Method 785
- ToraObject.FreeObject Method 785
- ToraObject.Indicator Property 781
- ToraObject.Instance Property 781
- ToraObject.IsDirty Method 786
- ToraObject.IsLocked Method 786
- ToraObject.IsNull Property 781
- ToraObject.Lock Method 786

- ToraObject.MarkDelete Method 786
- ToraObject.MarkUpdate Method 787
- ToraObject.ObjectType Property 782
- ToraObject.OCISvcCtx Property 782
- ToraObject.Refresh Method 787
- ToraObject.Unmark Method 787
- ToraPackage Class 817
- ToraPackage.PackageName Property 819
- ToraParam Class 492
- ToraParam.AsArray Property 497
- ToraParam.AsBFile Property 497
- ToraParam.AsBLOBLocator Property 498
- ToraParam.AsCLOBLocator Property 498
- ToraParam.AsCursor Property 498
- ToraParam.AsInterval Property 499
- ToraParam.AsNumber Property 499
- ToraParam.AsObject Property 499
- ToraParam.AsOraBlob Property 499
- ToraParam.AsOraClob Property 500
- ToraParam.AsRef Property 500
- ToraParam.AsTable Property 500
- ToraParam.AsTimeStamp Property 500
- ToraParam.AsXML Property 501
- ToraParam.ItemAsDateTime Property(Indexer) 501
- ToraParam.ItemAsFloat Property(Indexer) 501
- ToraParam.ItemAsInteger Property(Indexer) 502
- ToraParam.ItemAsInterval Property(Indexer) 502
- ToraParam.ItemAsString Property(Indexer) 503
- ToraParam.ItemAsTimeStamp Property(Indexer) 503
- ToraParam.ItemClear Method 506
- ToraParam.ItemIsNull Property(Indexer) 503
- ToraParam.ItemText Property(Indexer) 504
- ToraParam.ItemValue Property(Indexer) 504
- ToraParam.Length Property 504
- ToraParam.National Property 505
- ToraParam.SetBlobData Method 507
- ToraParam.Table Property 505
- ToraParams Class 507
- ToraParams.Items Property(Indexer) 508
- ToraPoolingOptions Class 508
- ToraPoolingOptions.PoolType Property 510
- ToraPoolingOptions.ProxyPassword Property 510
- ToraPoolingOptions.ProxyUsername Property 510
- ToraPoolingType Enumeration 735
- ToraProvider Class 821
- ToraQuery Class 510
- ToraQuery.FetchAll Property 522
- ToraQuery.LockMode Property 523
- ToraQuery.UpdatingTable Property 523
- ToraQueue
 - ToraQueueAdmin and ToraQueueTable Components 54
- ToraQueue Class 629
- ToraQueue.AsyncNotification Property 631
- ToraQueue.Dequeue Method 634
- ToraQueue.DequeueOptions Property 631
- ToraQueue.Enqueue Method 635
- ToraQueue.EnqueueArray Method 637
- ToraQueue.EnqueueMessageProperties Property 632
- ToraQueue.EnqueueOptions Property 632
- ToraQueue.Listen Method 638
- ToraQueue.OnMessage Event 639
- ToraQueue.PayloadArrayTypeName Property 632
- ToraQueue.PayloadTypeName Property 633
- ToraQueue.QueueName Property 633
- ToraQueue.Session Property 633
- ToraQueueAdmin Class 639
- ToraQueueAdmin.AddSubscriber Method 645
- ToraQueueAdmin.AlterComment Method 646
- ToraQueueAdmin.AlterMaxRetries Method 646
- ToraQueueAdmin.AlterPropagationSchedule Method 646
- ToraQueueAdmin.AlterQueue Method 647
- ToraQueueAdmin.AlterRetentionTime Method 648
- ToraQueueAdmin.AlterRetryDelay Method 648

- TOracleQueueAdmin.AlterSubscriber Method 648
- TOracleQueueAdmin.Comment Property 642
- TOracleQueueAdmin.CreateQueue Method 649
- TOracleQueueAdmin.DisablePropagationSchedule Method 649
- TOracleQueueAdmin.DropQueue Method 650
- TOracleQueueAdmin.EnablePropagationSchedule Method 650
- TOracleQueueAdmin.GetSubscribers Method 651
- TOracleQueueAdmin.GrantQueuePrivilege Method 651
- TOracleQueueAdmin.MaxRetries Property 642
- TOracleQueueAdmin.MultipleConsumers Property 642
- TOracleQueueAdmin.QueueName Property 642
- TOracleQueueAdmin.QueueTableName Property 643
- TOracleQueueAdmin.QueueType Property 643
- TOracleQueueAdmin.ReadQueueProperties Method 652
- TOracleQueueAdmin.RemoveSubscriber Method 652
- TOracleQueueAdmin.RetentionTime Property 643
- TOracleQueueAdmin.RetryDelay Property 643
- TOracleQueueAdmin.RevokeQueuePrivilege Method 652
- TOracleQueueAdmin.SchedulePropagation Method 653
- TOracleQueueAdmin.Session Property 644
- TOracleQueueAdmin.StartDequeue Method 653
- TOracleQueueAdmin.StartEnqueue Method 654
- TOracleQueueAdmin.StartQueue Method 654
- TOracleQueueAdmin.StopDequeue Method 655
- TOracleQueueAdmin.StopEnqueue Method 655
- TOracleQueueAdmin.StopQueue Method 655
- TOracleQueueAdmin.UnschedulePropagation Method 656
- TOracleQueueAdmin.VerifyQueueTypes Method 656
- TOracleQueueTable Class 657
- TOracleQueueTable.AlterComment Method 663
- TOracleQueueTable.AlterPrimaryInstance Method 664
- TOracleQueueTable.AlterQueueTable Method 664
- TOracleQueueTable.AlterSecondaryInstance Method 665
- TOracleQueueTable.Comment Property 659
- TOracleQueueTable.Compatible Property 659
- TOracleQueueTable.CreateQueueTable Method 665
- TOracleQueueTable.DropQueueTable Method 665
- TOracleQueueTable.GrantSystemPrivilege Method 666
- TOracleQueueTable.MessageGrouping Property 660
- TOracleQueueTable.MigrateQueueTable Method 666
- TOracleQueueTable.MultipleConsumers Property 660
- TOracleQueueTable.PayloadTypeName Property 660
- TOracleQueueTable.PrimaryInstance Property 661
- TOracleQueueTable.PurgeQueueTable Method 667
- TOracleQueueTable.QueueTableName Property 661
- TOracleQueueTable.ReadQueueTableProperties Method 667
- TOracleQueueTable.RevokeSystemPrivilege Method 668
- TOracleQueueTable.SecondaryInstance Property 661
- TOracleQueueTable.Secure Property 662
- TOracleQueueTable.Session Property 662
- TOracleQueueTable.SortList Property 662
- TOracleQueueTable.StorageClause Property 662
- TOracleRef Class 787
- TOracleRef.AsHex Property 791
- TOracleRef.Clear Method 794
- TOracleRef.OCIRef Property 792

- ToraRef.OCIRefPtr Property 792
- ToraRef.Pin Method 794
- ToraRef.ReflsNull Method 794
- ToraRef.Unpin Method 795
- ToraReferenceField Class 523
- ToraReferenceField.Modified Property 524
- ToraScript Class 823
- ToraScript.DataSet Property 825
- ToraScript.Session Property 826
- ToraScript.Statements Property 826
- ToraSession Class 524
- ToraSession.AssignConnect Method 540
- ToraSession.AutoCommit Property 529
- ToraSession.ChangePassword Method 541
- ToraSession.Connected Property 530
- ToraSession.ConnectMode Property 530
- ToraSession.ConnectPrompt Property 531
- ToraSession.ConnectionString Property 531
- ToraSession.Debug Property 532
- ToraSession.GetSequenceNames Method 542
- ToraSession.Home Property 532
- ToraSession.HomeName Property 532
- ToraSession.InternalName Property 533
- ToraSession.LastError Property 533
- ToraSession.LDA Property 533
- ToraSession.OCICallStyle Property 534
- ToraSession.OCISvcCtx Property 534
- ToraSession.OnConnectChange Event 546
- ToraSession.OnFailover Event 546
- ToraSession.Options Property 534
- ToraSession.OracleVersion Property 536
- ToraSession.ParamByName Method 542
- ToraSession.Password Property 536
- ToraSession.Ping Method 542
- ToraSession.PoolingOptions Property 536
- ToraSession.ProxySession Property 537
- ToraSession.RollbackToSavepoint Method 543
- ToraSession.Savepoint Method 543
- ToraSession.Schema Property 537
- ToraSession.Server Property 537
- ToraSession.SQL Property 538
- ToraSession.StartTransaction Method 544
- ToraSession.ThreadSafety Property 538
- ToraSession.Username Property 539
- ToraSessionOptions Class 547
- ToraSessionOptions.CharLength Property 550
- ToraSessionOptions.Charset Property 550
- ToraSessionOptions.ClientIdentifier Property 550
- ToraSessionOptions.ConnectionTimeout Property 550
- ToraSessionOptions.ConvertEOL Property 551
- ToraSessionOptions.DateFormat Property 551
- ToraSessionOptions.DateLanguage Property 551
- ToraSessionOptions.Direct Property 551
- ToraSessionOptions.EnableIntegers Property 552
- ToraSessionOptions.EnableNumbers Property 552
- ToraSessionOptions.EnableOraTimestamp Property 552
- ToraSessionOptions.NeverConnect Property 552
- ToraSessionOptions.OptimizerMode Property 553
- ToraSessionOptions.StatementCache Property 553
- ToraSessionOptions.StatementCacheSize Property 553
- ToraSessionOptions.UseOCI7 Property 553
- ToraSessionOptions.UseUnicode Property 554
- ToraSQL Class 554
- ToraSQL.ArrayLength Property 557
- ToraSQL.BreakExec Method 561
- ToraSQL.CreateProcCall Method 562
- ToraSQL.ErrorOffset Method 562
- ToraSQL.FindParam Method 562
- ToraSQL.NonBlocking Property 558
- ToraSQL.ParamByName Method 563
- ToraSQL.Params Property 558
- ToraSQL.RowsProcessed Property 559
- ToraSQL.Session Property 559
- ToraSQL.SQLType Property 559
- ToraSQL.StatementCache Property 560

- TOraSQL.TemporaryLobUpdate Property 560
- TOraSQLMonitor Class 890
- TOraStatement Class 827
- TOraStatement.Params Property 828
- TOraStatements Class 829
- TOraStatements.Items Property(Indexer) 830
- TOraStoredProc Class 563
- TOraStoredProc.ExecProc Method 582
- TOraStoredProc.LockMode Property 575
- TOraStoredProc.Overload Property 576
- TOraStoredProc.PrepareSQL Method 582
- TOraStoredProc.StoredProcName Property 576
- TOraTable Class 858
- TOraTable.EmptyTable Method 878
- TOraTable.FetchAll Property 871
- TOraTable.LockMode Property 871
- TOraTable.OrderFields Property 871
- TOraTable.PrepareSQL Method 878
- TOraTable.TableName Property 872
- TOraTimeStamp Class 721
- TOraTimeStamp.AllocDateTime Method 726
- TOraTimeStamp.AsDateTime Property 723
- TOraTimeStamp.AsString Property 723
- TOraTimeStamp.Compare Method 727
- TOraTimeStamp.Construct Method 727
- TOraTimeStamp.DescriptorType Property 724
- TOraTimeStamp.Format Property 724
- TOraTimeStamp.IsNull Property 724
- TOraTimeStamp.OCIDateTime Property 725
- TOraTimeStamp.OCIDateTimePtr Property 725
- TOraTimeStamp.Precision Property 725
- TOraTimeStamp.SetDate Method 728
- TOraTimeStamp.SetTime Method 728
- TOraTimeStamp.SetTimeZoneOffset Method 729
- TOraTimeStamp.TimeZone Property 725
- TOraTimeStampField Class 582
- TOraTimeStampField.AsTimeStamp Property 583
- TOraTimeStampField.Format Property 583
- TOraTrace Class 584
- TOraTrace.Enabled Property 585
- TOraTrace.GetSessionPID Method 588
- TOraTrace.GetTraceFileName Method 588
- TOraTrace.MaxTraceFileSize Property 586
- TOraTrace.PISqlTraceComment Method 588
- TOraTrace.PISqlTraceLimit Method 589
- TOraTrace.PISqlTraceMode Property 586
- TOraTrace.PISqlTracePause Method 589
- TOraTrace.PISqlTraceResume Method 589
- TOraTrace.PISqlTraceRunNumber Method 590
- TOraTrace.PISqlTraceStart Method 590
- TOraTrace.PISqlTraceStop Method 590
- TOraTrace.Session Property 586
- TOraTrace.SqlTraceMode Property 586
- TOraTrace.SqlTraceStart Method 591
- TOraTrace.SqlTraceStop Method 591
- TOraTrace.State Property 587
- TOraTrace.TraceFileIdentifier Property 587
- TOraTransaction Class 893
- TOraTransaction Component 52
- TOraTransaction.Active Property 895
- TOraTransaction.AddSession Method 900
- TOraTransaction.BranchQualifiers Property(Indexer) 896
- TOraTransaction.ClearSessions Method 900
- TOraTransaction.DefaultSession Property 896
- TOraTransaction.Detach Method 901
- TOraTransaction.GlobalCoordinator Property 896
- TOraTransaction.InactiveTimeOut Property 897
- TOraTransaction.IsolationLevel Property 897
- TOraTransaction.OnError Event 904
- TOraTransaction.RemoveSession Method 901
- TOraTransaction.Resume Method 901
- TOraTransaction.ResumeTimeOut Property 897
- TOraTransaction.RollbackToSavepoint Method 902
- TOraTransaction.Savepoint Method 902

- TOrTransaction.SessionsProperty(Indexer) 897
- TOrTransaction.SessionsCount Property 898
- TOrTransaction.StartTransaction Method 903
- TOrTransaction.TransactionId Property 898
- TOrTransaction.TransactionNameProperty 898
- TOrType Class 795
- TOrType.DataSize Property 797
- TOrType.Describe Method 799
- TOrType.IndicatorSize Property 797
- TOrType.TDO Property 797
- TOrUpdateSQL Class 591
- TOrXML Class 799
- TOrXML.AllocObject Method 806
- TOrXML.AsString Property 803
- TOrXML.Exists Method 807
- TOrXML.Extract Method 808
- TOrXML.IsSchemaBased Method 809
- TOrXML.LoadFromStream Method 810
- TOrXML.SaveToStream Method 810
- TOrXML.Transform Method 810
- TOrXML.Validate Method 811
- TOrXMLField Class 593
- TOrXMLField.AsXML Property 593
- TPISqlTraceMode Set 596
- TPoolingOptions Class 350
- TPoolingOptions.ConnectionLifetimeProperty 351
- TPoolingOptions.MaxPoolSize Property 351
- TPoolingOptions.MinPoolSize Property 351
- TPoolingOptions.Validate Property 351
- TQueueAgent Class 668
- TQueueAgent.Address Property 669
- TQueueAgent.Name Property 669
- TQueueAgent.Protocol Property 670
- TQueueAgents Class 670
- TQueueAgents.Add Method 671
- TQueueAgents.Insert Method 672
- TQueueAgents.Items Property(Indexer) 671
- TQueueDeliveryMode Enumeration 681
- TQueueMessage Class 672
- TQueueMessage.MessageId Property 673
- TQueueMessage.MessagePropertiesProperty 673
- TQueueMessage.RawPayload Property 674
- TQueueMessage.StringPayload Property 674
- TQueueMessageEvent Procedure Reference 680
- TQueueMessageGrouping Enumeration 682
- TQueueMessageProperties Class 674
- TQueueMessageProperties.AttemptsProperty 676
- TQueueMessageProperties.CorrelationProperty 676
- TQueueMessageProperties.Delay Property 676
- TQueueMessageProperties.DeliveryModeProperty 677
- TQueueMessageProperties.EnqueueTimeProperty 677
- TQueueMessageProperties.ExceptionQueueProperty 677
- TQueueMessageProperties.ExpirationProperty 677
- TQueueMessageProperties.OriginalMessageId Property 678
- TQueueMessageProperties.PriorityProperty 678
- TQueueMessageProperties.RecipientListProperty 678
- TQueueMessageProperties.SenderIdProperty 679
- TQueueMessageProperties.State Property 679
- TQueueMessageProperties.TransactionGroupProperty 679
- TQueueMessageState Enumeration 682
- TQueueNavigation Enumeration 682
- TQueueSequenceDeviation Enumeration 683
- TQueueSortList Enumeration 683
- TQueueType Enumeration 684
- TQueueVisibility Enumeration 684
- TraceFileIdentifier Property 587
- TraceFlags Property 209
- TransactionGroup Property 679
- TransactionId Property 898
- TransactionName Property 898

Transform Method 810
Transformation Property
 TDequeueOptions 625
 TEnqueueOptions 629
Transparent Application Failover Support 75
TRefreshMode Enumeration 598
TRefreshOption Enumeration 357
TRefreshOptions Set 354
TRetryMode Enumeration 357
TrimFixedChar Property 308
Truncate Method 378
TSequenceMode Enumeration 599
TSharedObject Class 385
TSharedObject.AddRef Method 387
TSharedObject.RefCount Property 386
TSharedObject.Release Method 387
TSmartQuery Class 878
TSmartQueryOptions Class 885
TSmartState Enumeration 888
TSortType Enumeration 390
TSqlTraceMode Set 596
TUpdateExecuteEvent Procedure Reference 354
TUpdateReckKind Enumeration 391
TUpdateReckinds Set 388
TVirtualTable Class 907
TVirtualTable.AddField Method 912
TVirtualTable.Assign Method 913
TVirtualTable.Clear Method 913
TVirtualTable.DeleteField Method 913
TVirtualTable.DeleteFields Method 914
TVirtualTable.LoadFromFile Method 914
TVirtualTable.LoadFromStream Method 915
TVirtualTable.Options Property 910
TVirtualTable.SaveToFile Method 915
TVirtualTable.SaveToStream Method 915
TVirtualTableOption Enumeration 918
TVirtualTableOptions Set 917

- U -

ukDelete 391
ukInsert 391
ukUpdate 391
Unicode Character Data 61
UniDirectional Property 262

UnLock Method 277
Unmark Method 787
UnpackMessage Method 615
Unpin Method 795
UnPrepare Method
 TCustomDASQL 291
 TMemDataSet 407
UnschedulePropagation Method 656
Update Method 363
UpdateAllFields Property 308
UpdateBatchSize Property 308
UpdateObject Property 455
UpdateRecordTypes Property 397
UpdateResult Method 408
UpdatesPending Property 397
UpdateStatus Method 408
Updating Data with ODAC Dataset Components 47
UpdatingTable Property 523
UseDefSession Variable 605
UseOCI7 Property 553
Username Property
 TCustomDACConnection 232
 TOraSession 539
UsernameLabel Property 225
UseUnicode Property 554
Using Connection Pooling 87
Using Several DAC Products in One IDE 90

- V -

Validate Method 811
Validate Property 351
Value Property
 TDAParam 334
 TMacro 346
VariableByName Method 817
VARRAY Data Type 68
VerifyQueueTypes Method 656
View Method 852
VirtualTable Unit Members 906
Visibility Property
 TDequeueOptions 626
 TEnqueueOptions 629
voPersistentData 918
voStored 918

- W -

WaitExecuting Method 291

WaitTimeout Property 626

What's New 12

Working in an Unstable Network 77

Write Method 379

WriteLob Method 716

Writing GUI Applications with ODAC 98

Writing Oracle External Procedures with
ODAC 73

- X -

XMLTYPE Data Type 65