Table of Contents

MSAccess 359

Classes 361

1 Overview

SQL Server Data Access Components (SDAC) is a library of components that provides access to Microsoft SQL Server databases. SDAC connects to SQL Server directly through OLE DB. The SDAC library is designed to help programmers develop faster and cleaner SQL Server database applications. SDAC is a complete replacement for standard SQL Server connectivity solutions and presents an efficient alternative to the Borland Database Engine for access to SQL Server.
The SDAC library is actively developed and supported by the Devart Team. If you have questions about SDAC, email the developers at sdac@devart.com or visit SDAC online at http://www.devart.com/sdac/.

Advantages of SDAC Technology

SDAC is a direct database connectivity wrapper built specifically for the SQL Server. SDAC offers wide coverage of the SQL Server feature set, and emphasizes optimized data access strategies.

Wide Coverage of SQL Server Features

By providing access to the most advanced database functionality, SDAC allows developers to harness the full capabilities of the SQL Server and optimize their database applications. SDAC provides complete support for working with SQL Server Compact Edition, SQL Server queuing and reliable messaging, IRowsetFastLoad interface, working with metadata information, MARS. Get a full list of supported SQL Server features in the Features topic.

Optimized Code

The goal of SDAC is to enable developers to write efficient and flexible database applications. The SDAC library is implemented using optimized code and advanced data access algorithms. Component interfaces undergo comprehensive performance tests and are designed to help you write efficient product data access layers. Find out more about using SDAC to optimize your database applications in Increasing Performance.

Compatibility with other Connectivity Methods

The SDAC interface retains compatibility with standard VCL data access components like BDE. Existing BDE-based applications can be easily migrated to SDAC and enhanced to take advantage of SQL Server-specific features. Project migration can be automated with the BDE/ADO Migration Wizard. Find out more about Migration Wizard in Using Migration Wizard.

Development and Support

SDAC is an SQL Server connectivity solution that is actively developed and supported. SDAC comes with full documentation, demo projects, and fast (usually within one business day) technical support by the SDAC development team. Find out more about getting help or submitting feedback and suggestions to the SDAC Development Team in the Getting Support topic.
A description of the SDAC 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)
- VCL, VCL.NET, and CLX versions of library available
- Full support of the latest Microsoft SQL Server versions, including Express and Compact editions
- Support for all SQL Server data types
- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- All types of local sorting and filtering, including by calculated and lookup fields
- Automatic data updating with TMSQuery, TMSTable, and TMSStoredProc components
- Unicode support
- Support for many SQL Server-specific features, such as messaging and bulk copy operations
- Advanced script execution with TMSScript component
- Support for using macros in SQL
- Easy migration from BDE and ADO with Migration Wizard
- Lets you use Professional Edition of Delphi and C++Builder to develop client/server applications
- Included annual SDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

The full list of SDAC features are available in the Features topic.

2 Editions

SQL Server Data Access Components comes in three editions: SDAC Standard Edition, SDAC Professional Edition, and SDAC Trial Edition.
SDAC Standard Edition includes the SDAC basic connectivity components and the SDAC Migration Wizard. SDAC Standard Edition is a good choice for beginning SQL Server developers and a cost-effective solution for database application developers who only need basic connectivity functionality for SQL Server.
SDAC *Professional* Edition shows off the full power of SDAC, enhancing SDAC Standard Edition with support for SQL Server-specific functionality. SDAC *Professional* Edition is intended for serious application developers who want to take advantage of all the SQL Server-specific functionality support provided by SDAC.
SDAC Trial Edition is the evaluation version of SDAC. It includes all the functionality of SDAC Professional Edition with a trial limitation of 60 days. C++Builder and supported .NET IDEs have additional trial limitations*.
You can get source code of all the component classes in SDAC by purchasing the special SDAC *Professional* Edition with Source Code**.
For more information about getting the SDAC edition you want, visit the How to Order section.

SDAC Edition Matrix

Feature	**Professional ****	**Standard**	**Trial**
Base Components			
TMSConnection TMSQuery TMSSQL TMSTable TMSStoredProc TMSUpdateSQL TMSSQLMonitor TMSConnectDialog TMSScript TMSDataSource TCRDBGrid TVirtualTable MSDataAdapter	+	+	+
Additional Components TMSEncryptor TMSCompactConnection TMSTransaction TMSChangeNotification TMSLoader TMSDump TMSServiceBroker TMSMetadata TCRBatchMove	+	-	+
Design-time features, including component editors and property editors	+	+	+
DataSet Manager ***	+	-	+
Migration Wizard ***	+	+	+
Trial limitations*	-	-	+

** Trial Edition is a fully working version of SDAC Professional Edition for a trial period of 60 days on most supported IDEs. After the trial period expires you should either register or uninstall SDAC. SDAC Trial Edition requires the IDE to be launched on the target workstation when testing .NET applications and applications written in C++Builder. For more information about trial limitations see the Ordering topic.*
*** Professional edition with source code is available. Migration Wizard, DataSet Manager source code is not distributed.*
**** Not available for C++Builder, Delphi 8, and FreePascal.*

3 Getting Started

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

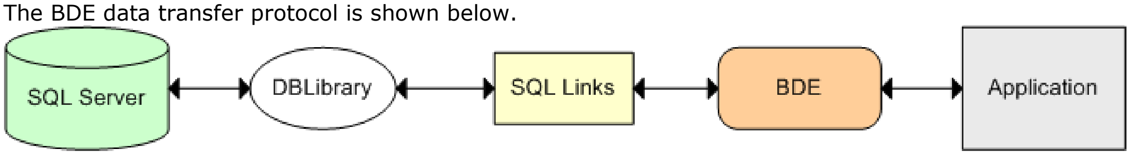
- What is SDAC?
- How does SDAC work?
- Installing SDAC.
- Working with the SDAC demo projects.
- Compiling and deploying your SDAC project.
- Using the SDAC documentation.
- How to get help with SDAC.

What is SDAC?

SQL Server Data Access Components (SDAC) is a component library which provides direct connectivity to SQL Server for Delphi, Delphi for .NET and C++Builder, and helps you develop fast SQL Server-based database applications with these environments.
Many SDAC classes are based on VCL, VCL for .NET, and CLX classes and interfaces. SDAC is a replacement for the Borland Database Engine, provides native database connectivity, and is specifically designed as an interface to the SQL Server database.
An introduction to SDAC is provided in the Overview section.
A list of the SDAC features you may find useful is listed in the Features section.
An overview of the SDAC component classes is provided in the Components List section.

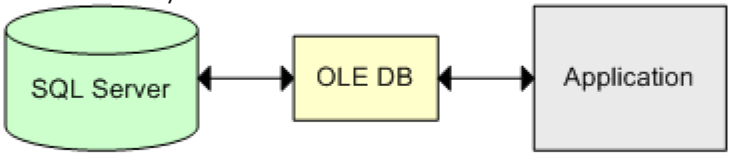
How does SDAC work?

SDAC uses OLE DB directly through a set of COM-based interfaces to connect to server. SDAC is designed to be lightweight and consists of a minimal layer between your code and SQL Server databases.
In comparison, the Borland Database Engine (BDE) uses several layers to access SQL Server, and requires additional data access software to be installed on client machines.
The BDE data transfer protocol is shown below.



BDE Connection Protocol

SDAC works directly through OLE DB, which is a native SQL Server interface. Applications with SDAC components access server directly:



SDAC Connection Flow

Installing SDAC

To install SDAC, complete the following steps.

1. Choose and download the version of the SDAC installation program that is compatible with your IDE. For instance, if you are installing SDAC 4.00, you should use the following files:

   For BDS 2006 and Turbo - sdac400d10.exe
   For Delphi 7 - sdac400d7.exe

   For more information, visit the the SDAC download page.
2. Close all running IDEs.
3. Launch the SDAC installation program you downloaded in the first step and follow the instructions

to install SDAC.

By default, the SDAC installation program should install compiled SDAC libraries automatically on all IDEs.
To check SDAC has been installed properly, launch your IDE and make sure that an SQL Server Access page has been added to the Component palette and that an SQL Server menu was added to the Menu bar.
If you have bought SDAC Professional Edition with Source Code, you will be able to download both the compiled version of SDAC and the SDAC source code. The installation process for the compiled version is standard, as described above.The SDAC source code must be compiled and installed manually. Consult the supplied *ReadmeSrc.txt* file for more details.
To find out what gets installed with SDAC or to troubleshoot your SDAC installation, visit the Installation topic.

Working with the SDAC demo projects

The SDAC installation package includes a number of demo projects that demonstrate SDAC capabilities and use patterns. The SDAC demo projects are automatically installed in the SDAC installation folder.
To get started working with SDAC quickly, launch and explore the introductory SDAC demo project, *SdacDemo*, from your IDE. This demo project is a collection of demos that show how SDAC 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.

SdacDemo Walkthrough

1. Launch your IDE.
2. Choose File   Open Project from the menu bar
3. Find the SDAC directory and open the *SdacDemo* project. This project should be located in the Demos\SdacDemo folder.
   For example, if you are using Borland Developer Studio 2006, the demo project may be found at \Program Files\Devart\Sdac for Delphi 2006\Demos\Win32\SdacDemo\SdacDemo.bdsproj
4. Select Run   Run or press F9 to compile and launch the demo project. *SdacDemo* should start, and a full-screen SDAC 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 *SdacDemo*, 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 SQL Server. However, you will not be able to actually retrieve data from SQL Server or execute commands until you connect to the database.
5. Click on the "Connect" button in the *SdacDemo* toolbar. A Connect dialog box will open. Enter the connection parameters you use to connect to your SQL Server and click "Connect" in the dialog box.
   Now you have a fully functional interface to your SQL Server. You will be able to go through the different demos, browse tables, create and drop objects, and execute 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 you specify a test database in the connection step.
6. Click on the "Create" button to create all objects that will be used by *SdacDemo.* If some of these objects already exist in the database you have connected to, the following error message will appear.

   An error has occurred: There is already an object named 'SDAC Long char' in the database.
   You can manually create objects required for demo by using the following file: %SDAC%\Demos\InstallDemoObjects.sql
   %SDAC% is the SDAC installation path on your computer.
   Ignore this exception?

   This is a standard warning from the object execution script. Click "Yes to All" to ignore this message. *SdacDemo* will create the *SdacDemo* objects on the server you have connected to.
7. Choose a demo that demonstrates an aspect of working with SQL Server 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 SQL Server tables, select the Table demo from the "Working with Components" folder. A simple SQL Server table browser will open in the view panel which will let you open a table in your database by specifying its name and clicking on the Open button.
8. Click the "Demo source" button in the *SdacDemo* toolbar to find out how the demo you have selected was implemented. The source code behind the demo project will appear in the view panel. Try to find the places where SDAC components are used to connect to the database.
9. Click on the "Form as text" button in the *SdacDemo* toolbar to view the code behind the interface to the demo. Try to find the places where SDAC components are created on the demo form.
10. Repeat these steps for other demos listed in the explorer window. The available demos are organized in three folders.

Working with components
A collection of projects that show how to work with the basic SDAC components.
General demos
A collection of projects that show off the SDAC technology and demonstrate some ways to work with data.
SQL Server-specific demos
A collection of projects that demonstrate how to incorporate SQL Server features in database applications.

11. When you are finished working with the project, click on the "Drop" button in the *SdacDemo* toolbar to remove all schema objects added in Step 6.

Other SDAC demo projects

SDAC is accompanied by a number of other demo projects. A description of all SDAC demos is located in the Demo Projects topic.

Compiling and deploying your SDAC project

Compiling SDAC-based projects

By default, to compile a project that uses SDAC classes, your IDE compiler needs to have access to the SDAC dcu (obj) files. If you are compiling with runtime packages, the compiler will also need to have access to the SDAC bpl files. All the appropriate settings for both these scenarios should take place automatically during installation of SDAC. You should only need to modify your environment manually if you are using SDAC Professional Edition with Source Code.
You can check that your environment is properly configured by trying to compile one of the SDAC demo projects. If you have no problem compiling and launching the SDAC demos, your environment has been properly configured.
For more information about which library files and environment changes are needed for compiling SDAC-based projects, consult the Installation topic.

Deploying SDAC-based projects

To deploy an application that uses SDAC, you will need to make sure the target workstation has access to the following files.
- The SDAC bpl files, if compiling with runtime packages.
- The SDAC assembly files, if using VCL for .NET components.

If you are evaluating deploying projects with SDAC 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 in SDAC Trial Edition for C++Builder will only work if the C++Builder IDE is launched. More information about SDAC Trial Edition limitations is provided here.
The list of the files which may need to be deployed with SDAC-based applications is included in the Deployment topic.

Using the SDAC documentation

The SDAC documentation describes how to install and configure SDAC, how to use SDAC Demo Projects, and how to use the SDAC libraries.
The SDAC documentation includes a detailed reference of all the SDAC components and classes. Many of the SDAC components and classes inherit or implement members from other VCL, VCL for .NET, CLX classes and interfaces. The product documentation also includes a 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 a specific standard VCL/CLX class an SDAC component is inherited from, see the corresponding topic in your IDE documentation.
At install time, the SDAC documentation is integrated into your IDE. It can be invoked from the SQL Server menu added to the Menu Bar, or by pressing F1 in an object inspector or on a selected code segment.

How to get help with SDAC

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

-

If you have any questions about SDAC installation or licensing, consult the Licensing and FAQ sections.

- You can get community assistance and SDAC technical support on the SDAC Forum.
- To get help through the SDAC Priority Support program, send an email to the SDAC development team at sdac@devart.com.
- If you have any questions about ordering SDAC or any other Devart product, contact sales@devart.com.

For more information, consult the Getting Support topic.

4 Features

In this topic you will find the complete SDAC feature list sorted by categories.

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 and VCL for .NET versions of library available
- Separated run-time and GUI specific parts allow you to create pure console applications such as CGI
- Unicode support

Network and connectivity:

- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- Connection string support
- Ability to search for installed SQL Server databases in a local network
- Support for connection encryption without certificate validation

Compatibility:

- Full support of the latest Microsoft SQL Server versions, including Express and Compact editions
- Support for all SQL Server data types
- Compatible with all IDE versions starting with Delphi 5, C++Builder 5, and FreePascal (except Delphi 8)
- Includes provider for UniDAC Standard Edition
- Wide reporting component support, including support for InfoPower, ReportBuilder, and FastReport
- Wide support of all standard Borland and third-party visual data-aware controls
- Allows you to use Professional Edition of Delphi and C++Builder to develop client/server applications

SQL Server technology support:

- TMSLoader component for improving record insertion performance
- TMSServiceBroker component for SQL Server queuing and reliable messaging
- Enhanced support for User-defined Types of SQL Server
- Enhanced support for SQL Server Compact Edition with the TMSCompactConnection component
- Possibility to change application name for a connection
- Possibility to change workstation identifier for a connection
- Configuration of OEM/ANSI character translation
- Ability to lock records and tables

Performance:

- High overall performance
- Fast controlled fetch of large data blocks
- Optimized string data storing
- Advanced connection pooling
- Incredible performance of applying updates in CachedUpdates mode
- Caching of calculated and lookup fields
- Fast Locate in a sorted DataSet
- Preparing of user-defined update statements

Local data storage operations:

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

Data access and data management automation:

- Automatic data updating with TMSQuery, TMSTable and TMSStoredProc components
- Support for Query Notifications with the TMSChangeNotification component
- Automatic record refreshing
- Automatic query preparing

- Support for getting results from queries with the FOR XML clause in readable view
- Support for ftWideMemo field type in Delphi 2006 and higher

Extended data access functionality:
- Separate component for executing SQL statements
- Simplified access to table data with TMSTable component
- Ability to retrieve metadata information with TMSMetaData component
- BLOB compression support
- Support for using macros in SQL
- FmtBCD fields support
- NonBlocking mode allows background executing and fetching data in separate threads
- Ability to customize update commands by attaching external components to TMSUpdateSQL objects.
- Deferred detail DataSet refresh in master/detail relationships
- MIDAS technology support
- MSDataAdapter component for WinForms and ASP.NET applications.
- Distributed transactions are supported with the TMSTransaction component

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 TMSScript component
- Support for executing individual statements in scripts
- Support for executing huge scripts stored in files with dynamic loading
- Optimized multi-statement script execution

SQL execution monitoring:
- Extended SQL tracing capabilities provided by TMSSQLMonitor 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
- Advanced design-time component and property editors
- Automatic design-time component linking
- Easy migration from BDE and ADO with Migration Wizard
- More convenient data source setup with the TMSDataSource component
- Syntax highlighting in design-time editors

Product clarity:
- Complete documentation sets
- Printable documentation in PDF format
- A large amount of helpful demo projects

Error handling:
- Multiple error processing support
- Unicode error messages support

Licensing and support:
- Included annual SDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

5 What's New

05-Sep-12 New Features in SDAC 6.5:
- Rad Studio XE3 is supported
- Windows 8 is supported

21-Jun-12 New Features in SDAC 6.2:
- 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 TMSEncryptor component for data encryption is added
- Calling of the TCustomDASQL.BeforeExecute event is added

23-Nov-11 New Features in SDAC 6.1:
- Update 4 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- FireMonkey support is improved
- Lazarus 0.9.30.4 and FPC 2.6.0 are supported

15-Sep-11 New Features in SQL Server Data Access Components 6.00:
- 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 SQL Server Data Access Components 5.10:
- Lazarus 0.9.30 and FPC 2.4.2 is supported
- Support for Table-Valued Parameters is added
- TMSTableData component for storing data of Table-Valued Parameter type is added
- Support for SQL Server Compact Edition 4.0 is added
- Support of API interface for managing FILESTREAM data is added

13-Sep-10 New Features in SQL Server Data Access Components 5.00:
- Embarcadero RAD Studio XE suppored

10-Sep-09 New Features in SQL Server Data Access Components 4.80:
- Embarcadero RAD Studio 2010 supported

23-Oct-08 New Features in SQL Server Data Access Components 4.70:
- 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
- Completed with more comprehensive structured Help

23-May-08 New Features in SQL Server Data Access Components 4.50:
- Added compatibility with UniDAC
- Improved support of default field values
- Added ability to specify key fields for a dataset
- Added support of automatic records locking
- Added an option for setting lock wait timeout

09-Jan-08 New Features in SQL Server Data Access Components 4.35:
- SQL Server Compact Edition 3.5 supported
- Tested with SQL Server 2008 CTP 4

27-Sep-07 New Features in SQL Server Data Access Components 4.30:
- CodeGear RAD Studio 2007 supported
- Added enhanced support for User-defined Types of SQL Server

- Added support for distributed transactions with the new TMSTransaction component
- Added support for Query Notifications with the new TMSChangeNotification component
- Improved support with SQL Server Compact Edition with the new TMSCompactConnection component
- Added support for getting results from queries with the FOR XML clause in readable view
- Added ability to lock records and tables
- TMSMetaData is enhanced with more schema row sets
- Added support for connection encryption without certificate validation
- Added ability to force record fetch for datasets open in FetchAll=False mode
- Added support for detailed error messages output to DBMonitor
- Added ability to use the default login database if no database is assigned on connect
- Added the OnProgress event in TMSLoader

12-Jun-07 New Features in SQL Server Data Access Components 4.10:

- C++Builder 2007 supported

22-Mar-07 New Features in SQL Server Data Access Components 4.00:

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
- Added DataSet Manager to control project datasets
- New TMSScript component for easy execution of multistatement scripts with the following features added:
- Support for executing individual statements in scripts
- Support for executing huge scripts stored in files with dynamic loading
- Support for using standard SQL Server client tool syntax
- New TMSServiceBroker component for SQL Server 2005 queuing and reliable messaging added
- New TCRBatchMove component for transferring data between all types of TDataSet descendants added
- New TMSDump component for loading data to and from the server added
- Support for data export and import to/from XML
- WideMemo field type in Delphi 2006 supported
- Support for sending messages to DBMonitor from any point in your program added
- Added asynchronous execute and fetch modes
- Compressed BLOB support

Support for more SQL Server functionality:

- SQL Server Compact Edition supported
- Multiple Active Result Sets (MARS) supported
- Support for new data types, including XML, varchar(MAX), nvarchar(MAX), varbinary(MAX) added
- Improved record insertion performance with new TMSLoader component
- Added support for a new level of transaction isolation added
- Support for more server objects in TMSMetaData added
- Stored procedure parameters with default values supported

Extensions and improvements to existing functionality:

- General performance improved
- Master/detail functionality extensions:
- Local master/detail relationships support added
- Master/detail relationships in CachedUpdates mode support added
- Working with calculated and lookup fields improvements:
- Local sorting and filtering added
- Record location speed increased
- Improved working with lookup fields
- Greatly increased performance of applying updates in CachedUpdates mode
- Connection pool functionality improvements:
- Efficiency significantly improved
- API for draining the connection pool added
- Ability to customize update commands by attaching external components to TMSUpdateSQL objects added
- Support for DefaultValues on record insertion added
- Some performance improvements achieved:
- NUMERIC fields fetching

- Improved performance of executing Update commands while editing a dataset
- DataSet refreshing
- Record refreshing after updates and inserts
- Support for selecting database name in TMSConnectDialog component

Usability improvements:
- Syntax highlighting in design-time editors added
- Completely restructured and clearer demo projects
- Added FAQ section

28-Aug-06 New Features in SQL Server Data Access Components 3.80:
- Professional editions of Turbo Delphi, Turbo Delphi for .NET, Turbo C++ supported

26-Jan-06 New Features in SQL Server Data Access Components 3.70:
- Support for Delphi 2006 added
- Support for SQL Server 2005 added

30-May-05 New Features in SQL Server Data Access Components 3.55:
- Ability of automatic preparing query with TCustomDADataSet.Options.AutoPrepare property added
- Ability to synchronize position at different DataSets with TCustomDADataSet.GotoCurrent method added
- Optimized MSSQLMonitor BLOB parameters processing
- Improved behavior on editing master key on Master/Detail relation

24-Jan-05 New Features in SQL Server Data Access Components 3.50:
- Support for Delphi 2005 added
- Support for SQL Server 2005 beta 2 added
- Guid fields support for VirtualTable added

21-Oct-04 New Features in SQL Server Data Access Components 3.00:
- Support for Delphi 8 added
- Local sorting ability with TMemDataSet.IndexFieldNames added
- Connection pooling support
- TCRDBGrid sources in Standard edition
- MSDataAdapter component added
- .NET Windows Forms demo project added
- ASP.NET demo project added
- TMSConnection.GetStoredProcNames, GetTableNames, GetDatabaseNames added
- TMSConnection.ClientVersion, ServerVersion added
- Milliseconds support added

27-Jul-04 New Features in SQL Server Data Access Components 2.45.2:
- Methods TMSSQL.BreakExec and TCustomMSDataSet.BreakExec added
- Property TMSConnection.Options.AutoTranslate added
- Method ExecSQL in TMSConnection added
- Methods GetTableNames and GetDatabaseNames in TMSConnection added
- Unicode support for Locate on Win9x added

02-Oct-03 New Features in SQL Server Data Access Components 2.45:
- Property MSConnection.Options.WorkstationID added
- Performance to insert large BLOBs improved
- Performance significantly improved
- Event TMSConnection.OnInfoMessage added
- Multiple Errors support added
- Property MSConnection.Options.ApplicationName added
- Property TBlob.AsWideString added
- Parameters parsing improved. Symbol ':' in string literals is ignored
- Network error processing improved
- Performance demo added

04-Apr-03 New Features in SQL Server Data Access Components 2.40:
- WideString support added
- Property MSDataSet.Options.QuoteNames added
- Property MSConnection.Options.KeepDesignConnected added
- Property MSConnectDialog.StoreLogInfo published

24-Feb-03 New Features in SQL Server Data Access Components 2.35:

- Speed optimization for opening small queries
- MSConnection.Options added
- Limited MSConnection.ConnectString support added
- Output string and (var)bytes parameters are now obtained from the server with the maximum length not depending on set Param.Size
- DBMonitor client implementation moved to COM server

26-Dec-02 New Features in SQL Server Data Access Components 2.30:

- Delphi 7 supported
- New memory management model for ftString and ftVarBytes types added
- Support for blob fields in CachedUpdates mode added

09-Aug-02 New Features in SQL Server Data Access Components 2.05:

- DBMonitor support

18-Jul-02 New Features in SQL Server Data Access Components 2.00:

- Server cursors supported
- Queries with Multiple Result Sets supported
- Performance improved
- Opening queries without fetching all rows to client (FetchAll = False) supported
- UniDirectional support added
- Quick getting Identity value
- Refresh supported for StoredProc
- FullRefresh supported
- Check for old row values while executing Update and Delete added
- Changed behavior on close connection with open transaction from Commit to Rollback

21-Mar-02 New Features in SQL Server Data Access Components 1.30:

- C++Builder 6 supported

08-Nov-01 New Features in SQL Server Data Access Components 1.20:

- Added TMSParam class to represent parameters
- Query Analyzer and Enterprise Manager integration added
- Accelerated getting identity value on refresh

6 Demo Projects

SDAC includes a number of demo projects that show off the main SDAC functionality and development patterns.
The SDAC demo projects consist of one large project called *SdacDemo* with demos for all main SDAC components, use cases, and data access technologies, and a number of smaller projects on how to use SDAC in different IDEs and how to integrate SDAC with third-party components.
Most demo projects are built for Delphi and Borland Developer Studio. There are only two SDAC demos for C++Builder. However, the C++Builder distribution includes source code for all other demo projects as well.

Where are the SDAC demo projects located?

In most cases all SDAC demo projects are located in "%Sdac%\Demos\".
In Delphi 2007 for Win32 under Windows Vista all SDAC demo projects are located in "My Documents \Devart\Sdac for Delphi 2007\Demos", for example "C:\Documents and Settings\All Users\Documents \Devart\Sdac 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 follows.

```
Demos
   |—dotNet
   |   |—SdacDemo [.NET version of the main SDAC demo project]
   |   |—TechnologySpecific
   |   |   |— MSSQLCompact [.NET version of the demo, using SQL Server Compact Edition]
   |   |—Miscellaneous
   |       |— [Some other .NET demo projects]
   |-Win32
       |—SdacDemo [Win32 version of the main SDAC demo project]
       |—ThirdParty
       |   |— [A collection of demo projects on integration with third-party components]
       |—TechnologySpecific
       |   |— MSSQLCompact [Win32 version of the demo, using SQL Server Compact Edition]
       |—Performance [Demo project, that compares performance of SDAC with another
       components (BDE, ADO, dbExpress)]
       |—Miscellaneous
          |— [Some other Win32 demo projects on design technologies]
```

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

```
Demos
   |—SdacDemo [The main SDAC demo project]
   |—TechnologySpecific
   |   |— MSSQLCompact [Win32 version of the demo, using SQL Server Compact Edition]
   |—ThirdParty
   |   |— [A collection of demo projects on integration with third-party components]
   |—Performance [Demo project, that compares performance of SDAC with another components
   (BDE, ADO, dbExpress)]
   |—Miscellaneous
      |— [Some other demo projects on design technologies]
```

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

> **Note:** This documentation describes ALL the SDAC demo projects. The actual demo projects you will have installed on your computer depends on your SDAC version, SDAC 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 SDAC demo projects

To explore an SDAC demo project,

1. Launch your IDE.
2. In your IDE, choose File Open Project from the menu bar.
3. Find the directory you installed SDAC 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* file for more details.

The included sample applications are fully functional. To use the demos, you have to first set up a connection to SQL Server. You can do so 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 the necessary database objects. When you are done with a demo, click "Drop" to remove all the objects used for the demo from your database.
Note: The SDAC demo directory includes two sample SQL scripts for creating and dropping all the test schema objects used in the SDAC demos. You can modify and execute this script manually, if you would like. This will not change the behavior of the demos.
You can find a complete walkthrough for the main SDAC demo project in the Getting Started topic. The other SDAC demo projects include a *ReadMe* file with individual building and launching instructions.

Demo project descriptions

SdacDemo

*SdacDemo*is one large project that includes three collections of demos.

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

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

SQL Server-specific demos
A collection of samples that demonstrate how to incorporate SQL Server features in database applications.

SdacDemo can be opened from %Sdac%\Demos\SdacDemo\SdacDemo.dpr (.bdsproj). The following table describes all the demos contained in this project.

Working with Components

Name	Description
ChangeNotification	This demo uses the TMSChangeNotification component to automate synchronization of local data with the actual data on the server. Synchronization happens immediately after changes are applied to the server from a different connection.
ConnectDialog	Demonstrates how to customize the SDAC connect dialog. Changes the standard SDAC 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 SDAC connect dialog class.
CRDBGrid	Demonstrates how to work with the TCRDBGrid component. Shows off the main TCRDBGrid features, like filtering, searching, stretching, using compound headers, and more.
Dump	Demonstrates how to backup data from tables with the TMSDump component. Shows how to use scripts created during back up to restore table data. This demo lets you back up a table either by specifying the table name or by writing a SELECT query.
Loader	Uses the TMSLoader component to quickly load data into a server table. This demo also compares the two TMSLoader data loading handlers: GetColumnData and PutData.
Query	Demonstrates working with TMSQuery, which is one of the most useful SDAC components. Includes many TMSQuery 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 here when first becoming familiar with SDAC.
ServiceBroker	Demonstrates working with the TMSServiceBroker component. This sample shows how to organize simple messaging.
Sql	Uses the TMSSQL component to execute SQL statements. Demonstrates how to work in a separate thread, in standard mode, in NonBlocking mode, and how to break long-duration query execution.
StoredProc	Uses the TMSStoredProc component to access an editable recordset from an SQL Server stored procedure in the client application.

Table	Demonstrates how to use TMSTable 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.
Transaction	Demonstrates usage of the TMSTransaction component to control distributed transactions. The demo shows how to ensure consistent data changes across two connections.
UpdateSQL	Demonstrates using the TMSUpdateSQL component to customize update commands. Lets you optionally use TMSSQL and TMSQuery 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 SDAC's local storage functionality. This sample shows how to perform local filtering, sorting and TVirtualTable by multiple fields, including by calculated and lookup fields.
MasterDetail	Uses SDAC functionality to work with master/detail relationships. This sample shows how to use TMSMetaDatalocal master/detail functionality. Demonstrates different kinds of master/detail linking, including linking by SQL, by simple fields, and by calculated fields.
Pictures	Uses SDAC functionality to work with BLOB fields and graphics.. The sample demonstrates how to retrieve binary data from SQL Server database and display it on visual components. Sample also shows how to load and save pictures to files and to the database.
StoredProcUpdates	Demonstrates updating a recordset using stored procedures. Update events are tied to stored procedure calls in design time, and every recordset change causes the corresponding stored procedure call to be performed. The commands to call stored procedures are stored in the SQLInsert, SQLDelete, SQLUpdate properties of TMSQuery.
Threads	Demonstrates how SDAC can be used in multithreaded applications. This sample allows you to set up several threads and test SDAC's performance with multithreading.

SQL Server-specific Demos

Name	Description
Lock	This demo shows how to ensure database consistency with locking mechanism of SQL Server through the SDAC functionality. Basing on your choice, a record can be locked exclusively, or just protected from writing.
LongStrings	Demonstrates SDAC functionality for working with long string fields (fields that have more than 256 characters). Shows the different ways they can be displayed as memo fields and string fields.
ServerCursors	Compares performance of opening a large recordset with different cursor types : client cursor in FetchAll=True mode, client cursor in FetchAll=False mode, and server cursor.
Text	Uses SDAC functionality to work with text. The sample demonstrates how to retrieve text data from SQL Server database and display it on visual components. Sample also shows how to load and save text to files and to the database.
UDT	This demo demonstrates SDAC abilities for working with CLR User-defined Types (UDT) of SQL Server. The demo folder includes the demo itself, and the sources of a sample type used in this demo. For more information on how to perform all necessary settings, see the *Readme.html* file in the demo folder.

Supplementary Demo Projects

SDAC also includes a number of additional demo projects that describe some special use cases, show

how to use SDAC in different IDEs and give examples of how to integrate it with third-party components. These supplementary SDAC demo projects are sorted into subfolders in the %Sdac%\Demos\ directory.

Location	Name	Description
dotNet/		*[folder appears only for IDEs with support for .NET*
Miscellaneous	**AspNet**	Uses MSDataAdapter 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 a DataGrid and sends user changes back to the database.
	WinForms	Shows how to use SDAC to create a WinForms application. This demo project creates a simple WinForms application and fills a data grid from an MSDataAdapter data source.
SdacDemo	***SdacD mo***	*[.NET version of the main SDAC demo project - see above*
Technology Specific	**MSSQLCompact**	Demonstrates how to create applications that work with Microsoft SQL Server Compact Edition. Demo connects to a database and opens a table. If the specified database does not exist, it will be created automatically. User must have SQL Server Compact Edition installed to test this demo. This is the .NET version of the project.
Win32/		*[folder appears only for IDEs with support for .NET. For all other IDEs contents appear in root*
ThirdParty	**FastReport**	Demonstrates how SDAC can be used with FastReport components. This project consists of two parts. The first part is several packages that integrate SDAC components into the FastReport editor. The second part is a demo application that lets you design and preview reports with SDAC technology in the FastReport editor.
	InfoPower	Uses InfoPower components to display recordsets retrieved with SDAC. This demo project displays an InfoPower grid component and fills it with the result of an SDAC query. Shows how to link SDAC data sources to InfoPower components.
	IntraWeb	A collection of sample projects that show how to use SDAC 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.
	QuickReport	Lets you launch and view a QuickReport application based on SDAC. This demo project lets you modify the application in design-time.
	ReportBuilder	Uses SDAC data sources to create a ReportBuilder report that takes data from SQL Server database. Shows how to set up a ReportBuilder document in design-time and how to integrate SDAC components into the Report Builder editor to perform document design in run-time.
Technology Specific	**MSSQLCompact**	Demonstrates how to create applications that work with Microsoft SQL Server Compact Edition. Demo connects to a database and opens a table. If the specified database does not exists, it will be created automatically. User must have SQL Server Compact Edition installed to test this demo. This is the Win32 version of the project.

Miscellaneous	**CBuilder**	A general demo project about creating SDAC-based applications with C++Builder. Lets you execute SQL scripts and work with result sets in a grid. This is one of the two SDAC demos for C++Builder.
	Dll	Demonstrates creating and loading DLLs for SDAC-based projects. This demo project consists of two parts - an MSDll project that creates a DLL of a form that sends a query to the server and displays its results, and an MSExe project that can be executed to display a form for loading and running this DLL. Allows you to build a dll for one SDAC-based project and load and test it from a separate application.
	FailOver	Demonstrates the recommended approach to working with unstable networks. This sample lets you perform transactions and updates in several different modes, simulate a 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.
	Midas	Demonstrates using MIDAS technology with SDAC. 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 interaction to a server application for processing.
	Performance	Measures SDAC performance on several types of queries. This project lets you compare SDAC performance to BDE, ADO, and dbExpress. Tests the following functionality: Fetch, Master/Detail, Stored Procedure Call, 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
SdacDemo	***SdacDemo***	*[Win32 version of the main SDAC demo project - see above*

7 Component List

This topic presents a brief description of the components included in the SQL Server Data Access Components library. Click on the name of each component for more information. These components are added to the SQL Server 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 SDAC components are included in all SDAC editions. SDAC Professional Edition components are not included in SDAC Standard Edition.

Basic SDAC components

TMSConnection	Lets you set up and control connections to SQL Server.
TMSQuery	Uses SQL statements to retrieve data from SQL Server table or tables and supply it to one or more data-aware components through a TDataSource component. Provides flexible update of data.
TMSTable	Lets you retrieve and update data in a single table without writing SQL statements.
TMSStoredProc	Executes stored procedures and functions.
TMSSQL	Executes SQL statements and stored procedures that do not return rowsets.
TMSScript	Executes sequences of SQL statements.
TMSUpdateSQL	Lets you tune update operations for DataSet component.
TMSDataSource	Provides an interface between a SDAC dataset components and data-aware controls on a form.
TMSSQLMonitor	Interface for monitoring dynamic SQL execution in SDAC-based applications.
TMSConnectDialog	Is used on client side to supply username, password, database and server name.
TMSTableData	Is used for working with user-defined table types in SQL Server 2008.
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 SQL Server Access page.
MSDataAdapter	.NET component, uses TDataSet as data source for retrieving and saving data to System.Data.DataSet.

SDAC Professional Edition components

TMSEncryptor	Represents data encryption and decryption in client application.
TMSLoader	Provides quick data loading to SQL Server database.
TMSDump	Serves to store a database or its parts as a script and also to restore database from received script.
TMSServiceBroker	Lets you send and receive messages using the SQL Server Service Broker system.
TMSMetaData	Retrieves embracing metadata on specified SQL object.

TMSChangeNotification	Lets you react on different server side changes on-the-fly. Based on the Query Notifications mechanism of SQL Server.
TMSTransaction	Lets you control distributed transactions via Microsoft Distributed Transaction Coordinator.
TMSCompactConnection	Lets you set up and control connections to SQL Server Compact Edition.
TCRBatchMove	Transfers data between all types of TDataSets descendants. This component is placed on the Data Access page of the Component palette, not on the SQL Server Access page.

8 Hierarchy Chart

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

```
TObject
   |—TPersistent
        |—TComponent
             |—TCustomConnection
             |   |—TCustomDAConnection
             |       |—TCustomMSConnection
             |          |—TMSConnection
             |          |—TMSCompactConnection
             |—TDataSet
             |   |—TMemDataSet
             |       |—TCustomDADataSet
             |       |   |—TCustomMSDataSet
             |       |       |—TMSQuery
             |       |       |—TCustomMSTable
             |       |       |   |—TMSTable
             |       |       |—TCustomMSStoredProc
             |       |       |   |—TMSStoredProc
             |       |       |—TMSMetaData
             |       |—TVirtualTable
             |       |—TMSTableData
             |—TDataSource
             |   |—TCRDataSource
             |       |—TMSDataSource
             |—TCustomDASQL
             |   |—TMSSQL
             |—TCustomDASQLMonitor
             |   |—TMSSQLMonitor
             |—TCustomConnectDialog
             |   |—TMSConnectDialog
             |—TDALoader
             |   |—TMSLoader
             |—TDADump
             |   |—TMSDump
             |—TDAScript
             |   |—TMSScript
             |—DADataAdapter
             |   |—MSDataAdapter
             |—TDATransaction
             |   |TMSTransaction
             |—TCRBatchMove
             |—TMSChangeNotification
             |—TMSServiceBroker
             |—TField
               |—TMSUDTField
               |—TMSXMLField
             |—TCREncryptor
             |   |TMSEncryptor
```

9 Requirements

SDAC requires OLE DB installed on the workstation.
Note: In current versions of Microsoft Windows, as Windows 2000, OLE DB is already included as standard package. But it's highly recommended to download latest version (newer than 2.5) of Microsoft Data Access Components (MDAC).
If you are working with SQL Server Compact Edition, you should have it installed. You can download SQL Server Compact Edition from http://www.microsoft.com/sql/editions/compact/default.mspx. For more information visit Working with SQL Server Compact Edition.

10 Compatibility

SQL Server Compatibility

SDAC supports the following versions of SQL Server:

- SQL Server 2012 (including Express edition)
- SQL Server 2008 R2 (including Express edition)
- SQL Server 2008 (including Express edition)
- SQL Server 2005 (including Express edition)
- SQL Server 2000 (including MSDE)
- SQL Server 7
- SQL Server Compact 4.0, 3.5, 3.1
- SQL Azure

IDE Compatibility

SDAC 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 C++Builder XE2 for Win32
- 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 Win32 and Win64 platforms

Only Architect, Enterprise, and Professional IDE editions are supported. For Delphi XE/XE2/XE3, C++Builder XE/XE2/XE3 SDAC additionally supports Starter Edition.
Lazarus and Free Pascal are supported only in Trial Edition and Professional editions with source code.

Supported Target Platforms

- Windows, 32-bit and 64-bit

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

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.

11 Installation

This topic contains the environment changes made by the SDAC installer. If you are having problems with using SDAC or compiling SDAC-based products, check this list to make sure your system is properly configured.
Compiled versions of SDAC are installed automatically by the SDAC Installer for all supported IDEs except for Lazarus. Versions of SDAC with Source Code must be installed manually. Installation of SDAC from sources is described in the supplied *ReadMeSrc.txt* file.

Before installing SDAC ...

Two versions of SDAC 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 SDAC may be incompatible with older versions of MyDAC, IBDAC, and ODAC.
So before installing a new version of SDAC, uninstall any previous version of SDAC 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 sdac@devart.com
Note: You can avoid performing SDAC uninstallation manually when upgrading to a new version by directing the SDAC 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 `sdacXX.exe /force`, specifying the full path to the appropriate version of the installation program).

Installed packages

The SDAC package libraries are divided into Win32 project files and .NET project files.
Note: `%SDAC%` denotes the path to your SDAC installation directory.

Delphi/C++Builder Win32 project packages

Name	*Description*	*Location*
dacXX.bpl	DAC run-time package	Windows\System32
dcldacXX.bpl	DAC design-time package	Delphi\Bin
dacvclXX.bpl*	DAC VCL support package	Delphi\Bin
sdacXX.bpl	SDAC run-time package	Windows\System32
dclsdacXX.bpl	SDAC design-time package	Delphi\Bin
sdacvclXX.bpl*	VCL support package	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

Name	*Description*	*Location*
Devart.Dac.dll	DAC run-time package	Global Assembly Cache
Devart.Dac.Design.dll	DAC design-time package	%SDAC%\Bin
Devart.Dac.AdoNet.dll	Data provider core package	Delphi\Bin
Devart.Sdac.dll	SDAC Delphi for .NET run-time package	Global Assembly Cache
Devart.Sdac.Design.dll	SDAC design-time package	%SDAC%\Bin
Devart.Vcl.dll	TCRDBGrid component	Global Assembly Cache
Devart.Sdac.AdoNet.dll	Data provider for SQL Server package	Global Assembly Cache

Additional packages for using SDAC managers and wizards

Name	*Description*	*Location*
datasetmanagerXX.bpl	DataSet Manager package	Delphi\Bin

oramigwizardXX.dll	SDAC BDE Migration wizard	%SDAC%\Bin

Additional .NET packages for using SDAC managers and wizards

Name	*Description*	*Location*
Devart.Dac.DsManager.dll	DataSet Manager Assembly	Global Assembly Cache
Devart.Sdac.MigWizard.dll*	SDAC BDE Migration wizard Assembly	Global Assembly Cache

** Included in Borland Delphi 8 only*

Environment Changes

To compile SDAC-based applications, your environment must be configured to have access to the SDAC libraries. Environment changes are IDE-dependent.
For all instructions, replace `%SDAC%` with the path to your SDAC installation directory

Delphi

- `%SDAC%\Lib` should be included in the Library Path accessible from Tools Environment options Library.

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

Delphi for .NET

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

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

C++Builder

C++Builder 5, 6:

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

C++Builder 2006, 2007:

- `$(BCB)\SDAC\Lib` should be included in the Library search path of the Default Project Options accessible from Project Default Options C++Builder Linker Paths and Defines.
- `$(BCB)\SDAC\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 SDAC Installer performs C++Builder environment changes automatically for compiled versions of SDAC.

Lazarus

The SDAC installation program only copies SDAC files. You need to install SDAC packages to Lazarus IDE manually. Open %SDAC%\Source\Lazarus1\dclsdac.lpk file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilded with SDAC packages.
Do not press the Compile button for the package. Compiling will fail because there are no SDAC sources.
To check that your environment has been properly configured, try to compile one of the demo projects included with SDAC. The SDAC demo projects are located in %SDAC%/Demos.

Installation of Additional Components and Add-ins

DBMonitor

DBMonitor is 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 SDAC. DBMonitor is intended to hamper application being monitored as little as possible. For more information, visit the DBMonitor page online.

12 Deployment

SDAC requires OLE DB to be installed on the workstation. In current versions of Microsoft Windows, as Windows 2000, OLE DB is already included as standard package. But it is highly recommended to download the latest version (newer than 2.5) of Microsoft Data Access Components (MDAC).
Many features of SQL Server like Query Notifications, MARS require Microsoft SQL Server Native Client. If you need to use these features, you should download and install Microsoft SQL Server Native Client.
For applications that use SQL Server Compact Edition, the server itself is required to be installed on the client computer.
In order to use extended abilities of UDT fields, you will need to deploy the Devart.Sdac.UDTProxy.dll file with your application. This file should be present in the directory with your application, or registered in GAC.
SDAC 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 can check that 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 SDAC 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
sdacXX.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
sdacXX.bpl	always
dacvclXX.bpl	if your application uses the SdacVcl unit
sdacvclXX.bpl	if your application uses the SdacVcl unit
crcontrolsXX.bpl	if your application uses the CRDBGrid component

Deploying .NET applications

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

Devart.Dac.dll	always
Devart.Sdac.dll	always
Devart.Dac.AdoNet.dll	If your application uses MSDataAdapter component
Devart.Sdac.AdoNet.dll	If your application uses MSDataAdapter 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

SQL Server 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 SDAC installation folder.

Licensing

There are three types of full licenses for SDAC: Single Licenses, Team Licenses, and Site Licenses.
Single Licenses must be purchased for each developer working on a project that uses SDAC.
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 SDAC Trial Edition under a temporary **Evaluation License**, which allows you to test SDAC Trial Edition for a period of 60 days, after which you must either remove all files associated with SDAC or purchase a full license.
Licenses can be purchased for the following editions of SDAC: SDAC Standard Edition, SDAC Professional Edition, and SDAC Professional Edition with Source Code. An edition comparison chart can be found here
.
To purchase a license for SDAC, please visit www.devart.com/sdac/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 SDAC: SDAC Standard Edition, SDAC Professional Edition, and SDAC Professional Edition with Source Code.
Users can evaluate SDAC with SDAC Trial Edition under Evaluation License.
A comparison chart can be found here.

Subscriptions

The SDAC Subscription program is an annual maintenance and support service for SDAC users.
Users with a valid SDAC Subscription get the following benefits:

- Product support through the SDAC Priority Support program
- Access to new versions of SDAC when they are released
- Access to all SDAC 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

SDAC Evaluation License lets you try SDAC Trial Edition for a period of 60 days.
There are no functionality limitations in SDAC 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 SDAC Trial Edition
- If you are deploying a project built with SDAC Trial Edition, you will need to include the SDAC 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 SDAC and describes the SDAC Priority Support program.

Support Options

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

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

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

SDAC Priority Support

SDAC Priority Support is an advanced product support service for getting expedited individual assistance with SDAC-related questions from the SDAC 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 SDAC Subscription.
To get help through the SDAC Priority Support program, please send an email to sdac@devart.com describing the problem you are having. Make sure to include the following information in your message:

- The version of Delphi or C++Builder you are using.
- Your SDAC Registration number.
- Full SDAC edition name and version number. You can find both of these in the About sheet of TMSConnection Editor or from the SQL Server About menu.
- Versions of the SQL Server 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 Northwind 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 SQL Server Data Access Components.
If you have encounter a question with using SDAC, please browse through this list first. If this page does not answer your question, refer to the Getting Support topic in SDAC help.

Installation and Deployment

1. **I am having a problem installing SDAC or compiling SDAC-based projects...**
   You may be having a compatibility issue that shows up in one or more of the following forms:
   - Get a "Setup has detected already installed DAC packages which are incompatible with current version" message during SDAC installation.
   - Get a "Procedure entry point ... not found in ... " message when starting IDE.
   - Get a "Unit ... was compiled with a different version of ..." message on compilation.

   You can have such problems if you installed incompatible SDAC, ODAC, 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 the client computer so that my applications that use SDAC can run?**
   SDAC requires OLE DB installed on the workstation. In current versions of Microsoft Windows, such as Windows 2000, OLE DB is already included as part of the standard installation. But it is highly recommended to download the latest version (newer than 2.5) of Microsoft Data Access Components (MDAC).
   Many features of SQL Server like Query Notifications, MARS require Microsoft SQL Server Native Client. If you need to use these features, you should download and install Microsoft SQL Server Native Client.
   For applications that use SQL Server Compact Edition, the server itself is required to be installed on the client computer.
   For more information, please refer to the Deployment topic of the SDAC help.

Licensing and Subscriptions

1. **Am I entitled to distribute applications written with SDAC?**
   If you have purchased a full version of SDAC, you are entitled to distribute pre-compiled programs created with its use. You are not entitled to propagate any components inherited from SDAC or using SDAC source code. For more information see the *License.rtf* file in your SDAC installation directory.
2. **Can I create components using SDAC?**
   You can create your own components that are inherited from SDAC or that use the SDAC 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 SDAC 4.00?**
   The basic SDAC license agreement will remain the same. With SDAC 4.00, the SDAC Edition Matrix will be reorganized and a new SDAC Subscription Program will be introduced.
4. **What do the SDAC 4.00 Edition Levels correspond to?**
   SDAC 4.00 will come in four editions: Trial, Standard, Professional, and Professional 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 SDAC 4.00

Old Edition Level	New Edition Level
- *No Correspondence* -	SDAC Standard Edition
SDAC Standard Edition	SDAC Professional Edition
SDAC Professional Edition	SDAC Professional Edition with Sources
SDAC Trial Edition	SDAC Trial Edition

The feature list for each edition can be found in the SDAC documentation and on the SDAC website

.

5. **I have a registered version of SDAC. Will I need to pay to upgrade to future versions?**
   After SDAC 4.00, all upgrades to future versions are free to users with an active SDAC Subscription.
   Users that have a registration for versions of SDAC prior to SDAC 4.00 will have to first upgrade to SDAC 4.00 to jump in on the Subscription program.
6. **What are the benefits of the SDAC Subscription program?**
   The **SDAC Subscription Program** is an annual maintenance and support service for SDAC users. Users with a valid SDAC Subscription get the following benefits:
   - Access to new versions of SDAC when they are released
   - Access to all SDAC updates and bug fixes
   - Product support through the SDAC Priority Support program
   - Notification of new product versions

   Priority Support is an advanced product support program which offers you expedited individual assistance with SDAC-related questions from the SDAC developers themselves. Priority Support is carried out over email and has a two business day response policy.
   The SDAC Subscription Program is available for registered users of SDAC 4.00 and higher.
7. **Can I use my version of SDAC after my Subscription expires?**
   Yes, you can. SDAC version licenses are perpetual.
8. **I want a SDAC Subscription! How can I get one?**
   An annual SDAC Subscription is included when ordering or upgrading to any registered (non-Trial) edition of SDAC 4.00 or higher.
   You can renew your SDAC Subscription on the SDAC Ordering Page. For more information, please contact sales@crlab.com.
9. **Does this mean that if I upgrade to SDAC 4 from SDAC 3, I'll get an annual SDAC Subscription for free?**
   Yes.
10. **How do I upgrade to SDAC 4.00?**
    To upgrade to SDAC 4.00, you can get a Version Update from the SDAC Ordering Page. For more information, please contact sales@crlab.com.

Performance

1. **How productive is SDAC?**
   SDAC uses the lowest documented protocol level (OLE DB) to access the database server. This allows SDAC to achieve high performance. From time to time we compare SDAC with other products, and SDAC always takes first place.
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, SDAC 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 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 SDAC component editors at design time?**
   To enable syntax highlighting for SDAC, you should download and install the freeware SynEdit component set.
2. **How can I determine which version of SDAC am I using?**
   You can determine your SDAC version number in several ways:
   - During installation of SDAC, consult the SDAC Installer screen.
   - After installation, see the *history.html* file in your SDAC installation directiory.
   - At design-time, select SQL Server About SDAC from the main menu of your IDE.
   - At run-time, check the value of the SdacVersion 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 SDAC 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 SDAC during the execution of the Post, Delete, or RefreshRecord methods, and are used to construct the query to the server. Their values can contain parameters with names of fields in the underlying data source, which will be later replaced by appropriate data values.
For example, you can use the SQLInsert template to insert a row into a query instance as follows.

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

The value of the SQLInsert property will then be used by SDAC 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 SDAC dataset object will generate the query itself using the appropriate insert, update, delete, or refresh record syntax.

5. **How can I get a list of the databases on the server?**
   Use the TCustomDAConnection.GetDatabaseNames method.
6. **How can I get a list of the tables list in a database?**
   Use the TCustomDAConnection.GetTableNames method.
7. **Some questions about the visual part of SDAC**
   The following questions usually arise from the same problem:
   - I set the Debug property to True but nothing happens!
   - While executing a query, the screen cursor does not change to an hour-glass.
   - Even if I have LoginPromp set to True, the connect dialog does not appear.

   To fix this problem, you should add the SdacVcl unit to the uses clause of your project.

General Questions

1. **I would like to develop an application that works with SQL Server. Should I use SDAC or DbxSda?**
   DbxSda is our dbExpress driver for SQL Server. dbExpress technology serves for providing a more or less uniform way to access different servers (SQL Server, MySQL, Oracle and so on). It is based on drivers that include server-specific features. Like any universal tool, in many specialized cases dbExpress providers lose some functionality. For example, the dbExpress design-time is quite poor and cannot be expanded.
   SDAC is a specialized set of components for SQL Server, which has advanced server-specific design-time and a component interface similar to that of BDE.
   We tried to include maximal support of SQL Server-specific features in both DbxSda and SDAC. However, the nature of dbExpress technology has some insurmountable restrictions. For example, Unicode fields cannot be passed from a driver to dbExpress.
   SDAC and DbxSda use the same kernel and thus have similar performance. In some cases dbExpress is slower because data undergoes additional conversion to correspond to dbExpress standards.
   To summarise, if it is important for you to be able to quickly adapt your application to a database server other than SQL Server, it is probably better to use DbxSda. In other cases, especially when migrating from BDE or ADO, you should use SDAC.
2. **What are the advantages of SDAC over Delphi ADO components for accessing SQL Server?**
   ADO is a universal components while SDAC is specialized in SQL Server, so SDAC takes into account lots of server specific features. SDAC has great benefit in performance (5-10 times in different tests) because it works directly through OLE DB, which is a native SQL Server interface. Moreover, SDAC provides advanced design-time editors.
3. **Behaviour of my application has changed when I upgraded SDAC. How can I restore the old behaviour with the new version?**
   We always try to keep SDAC compatible with previous versions, but sometimes we have to change behaviour of SDAC 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 SDAC help. This topic describes such changes, and how to revert to the old SDAC behaviour.
4. **On connect, I get an "OLE DB error occured. Code 800401F0h. CoInitialize has not been called" error. What can I do?**
   As SDAC uses OLE DB, it is necessary to initialize OLE by calling CoInitialize before a new

connection is established. Usually VCL does this automatically. SDAC does not call to the CoInitialize and CoUninitialize functions itself, as this may cause unexpected problems if OLE is used in the program by someone else.

5. **Are the SDAC connection components thread-safe?**
   Yes, SDAC is thread-safe but there is a restriction. But the same TCustomMSConnection object descendant cannot be used in several threads. So if you have a multithreaded application, you should have a TCustomMSConnection object descendant for each thread that uses SDAC.
6. **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 SDAC help for more information.

7.

Some questions with using TCustomDADataSet.FetchAll=False mode

Common issues when using FetchAll=False:

- I have problems working with temporary tables.
- I have problems working with transactions.
- Sometimes my application hangs on applying changes to database.

Usage of FetchAll=False mode has many advantages; however, it also has some restrictions since it requires an additional connection to server to be created for data fetching. The additional connection is created to prevent the main connection from blocking.
These problems can be avoided by setting the FetchAll property to True. Please see description of the FetchAll propery and the CreateConnection option in SDAC help for more information.
One more way to solve these problems is to use the Multiple Active Result Sets (MARS) feature. This feature lets you keep more than one unfetched record set within a single connection. To enable MARS, set the MultipleActiveResultSets option of TMSConnection to True. Note: To use MARS, you will need to have SQL Server and SQL Native Client installed.

16 Using SDAC

16.1 Updating Data with SDAC Dataset Components

SDAC components that are descendants from TCustomDADataSet provide different means for reflecting local changes to the server.
The first approach is to use automatic generation of update SQL statements. Using this approach you should provide a SELECT statement, everything else will be made by SDAC automatically. In case when a SELECT statement uses multiple tables, you can use UpdatingTable property to specify which table will be updated. If UpdatingTable is blank, the table that corresponds to the first field in the dataset is used. This approach is the most preferable and is used in most cases.
Another approach is to set update SQL statements using SQLInsert, SQLUpdate and SQLDelete properties. Set them with SQL statements that will perform corresponding data modifications on behalf of the original statement whenever insert, update or delete operation is called. This is useful when there is no possibility to generate correct statement or you need to execute some specific statements. For example update operations should be made with stored procedure calls.
You may also assign UpdateObject property with the TMSUpdateSQL class instance which holds all updating SQL statements in one place. You can generate all these SQL statements using SDAC design time editors. For more careful customization of data update operations you can use InsertObject, ModifyObject and DeleteObject properties of TMSUpdateSQL component.

See Also
- TMSQuery
- TMSStoredProc
- TMSTable
- TMSUpdateSQL

16.2 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 SDAC 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.
SDAC provides two ways to bind tables. First code example shows how to bind two TCustomMSDataSet components (TMSQuery, TMSTable or even TMSStoredProc) into MD relationship via parameters.

```
procedure TForm1.Form1Create(Sender: TObject);
var
  Master, Detail: TMSQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TMSQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TMSQuery.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: TMSQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TMSQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TMSQuery.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

16.3 Migration Wizard

NOTE:

Migration Wizard is available only for Delphi IDE and is not available for C++Builder.
BDE/ADO Migration Wizard allows you to convert your BDE or ADO projects to SDAC. This wizard replaces BDE/ADO components at the specified project (dfm-and pas-files) to SDAC.
To convert a project, perform the following steps.

- Select **BDE/ADO Migration Wizard** from **SQL Server** menu
- Select **Replace BDE components** or **Replace ADO components** to replace corresponding components with SDAC 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/ADO 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/ADO Migration Wizard** from **SQL Server** menu
- Select Restore original files from backup and press the Next button.
- Select the backup file. By default it is RExpert.reu file in RBackup folder of your converted project. Press the Next button.
- Check your settings and press the Finish button to start the conversion operation.
- Press **Yes** in the dialog that appeared.

Your project will be restored to its previous state.

See Also

- Migration from BDE
- Migration from ADO

16.4 Migration from BDE

In SDAC the interests of BDE application developers were taken into consideration. So starting to use SDAC after working with BDE would be easy even for developing complex projects. Moreover, SDAC does not have problems like ones with LiveQuery and compatibility of applications developed using different versions in BDE. On SDAC developing BDE users interests were taken in consideration so conversion from using BDE into SDAC can be passed without difficulties even for complex projects. Moreover, SDAC does not have problems appropriated BDE with LiveQuery and compatibility of different programs that were developed using different BDE version and so on.
Abandoning BDE gives one more important advantage - positive effect on performance. Instead of complex BDE-ODBC drivers system it uses the fastest access - directly to Microsoft SQL Server.
SDAC provides special Wizard to simplify the conversion of already existing projects. This Wizard replaces BDE-components in the specified project (dfm-and pas-files) to SDAC. BDE-components that will be replaced:

- TDatabase -> TMSConnection
- TQuery -> TMSQuery
- TTable -> TMSTable
- TStoredProc -> TMSStoredProc
- TUpdateSQL -> TMSUpdateSQL

To run the Wizard select BDE/ADO Migration Wizard item in SDAC menu and follow the instructions. This Wizard is available only for Delphi IDE.
Note: Wizard serves only to simplify routine operations and after the conversion project might be uncompiled.
Below is a list of properties and methods that cannot be converted automatically. Here you can find hints for users to simplify manual replacement.

TDatabase

- AliasName - specific BDE property.Not supported by SDAC.
- DatabaseName - has a different meaning in BDE and SDAC. In SDAC it means SQL Server database name. See TCustomMSConnection for details.
- Locale - not supported by SDAC.
- KeepConnection - not supported by SDAC.
- Params - see TCustomMSConnection properties.
- Session, SessionAlias, SessionName - SDAC does not need global management of a group of database connections in an application. So these properties are not supported.
- Temporary - has no meaning in SDAC. Additional connections are created but they are not available for the user. See FetchAll = False for details.
- TraceFlags - see TCustomDASQLMonitor.TraceFlags.
- TransIsolation - see IsolationLevel.
- Execute - use ExecSQL instead of this method.
- FlushSchemaCache - not supported by SDAC.
- GetFieldNames - not supported by SDAC.
- IsSQLBased - not supported by SDAC. For SQL Server must be always True.
- ApplyUpdates - parameters are not supported. To update only specified DataSets, use ApplyUpdates . Update is performed within a transaction.

TBDEDataSet

- BlockReadSize - see FetchRows.
- CacheBlobs - SQL Server does not provide service of suspended BLOB loading.
- KeySize - specific BDE property. Not supported by SDAC.

TDBDataSet

- AutoRefresh - supported through TCustomDADataSet.RefreshOptions.
- DBFlags, DBHandle, DBLocate, DBSession, Handle - specific BDE property. Not supported by SDAC.
- SessionName - not supported by SDAC.
- UpdateMode - not supported by SDAC. By default, the behaviour corresponds upWhereKeyOnly. To change this behaviour see TCustomDADataSet.SQLUpdate, TCustomDADataSet.SQLDelete, TCustomDADataSet.SQLRefresh, and TCustomMSDataSet.Options.CheckRowVersion.

TQuery

- Constrained - specific BDE property. Not supported by SDAC.
- DataSource - see TCustomDADataSet.MasterSource.

- Local - specific BDE property. Not supported by SDAC.
- RequestLive - almost all query result sets can be updated. See TMSQuery.UpdatingTable, TCustomDADataSet.ReadOnly, CanModify, TCustomDADataSet.SQLInsert, TCustomDADataSet.SQLUpdate, TCustomDADataSet.SQLDelete.
- Text - specific BDE property. Not supported by SDAC.

TTable

- DefaultIndex - not used in SDAC. If you need to sort a table by any field see TCustomMSTable.OrderFields, TMemDataSet.IndexFieldNames.
- Exists, CreateTable, AddIndex, DeleteIndex, StoreDefs, Deletetable, TableType - SDAC does not allow to create tables using TTable. If you need to create a table execute 'CREATE TABLE ...' query or use any special third-party tools.
- IndexFieldNames - a list of fields for local sorting. See TMemDataSet.IndexFieldNames.
- IndexDefs, IndexFieldCount, IndexFields, IndexFiles, IndexName, GetIndexNames, GetIndexInfo - Not supported by SDAC.
- KeyExclusive - not supported by SDAC. Use SELECT ... FROM .. WHERE ... to get requested result .
- KeyFieldCount - not supported by SDAC as key fields are not used for searching on client side.
- TableLevel - BDE-specific property. Not supported by SDAC.
- ApplyRange, CancelRange, EditRangeStart, EditRangeEnd, SetRange - SDAC does not support Range.
- BatchMove - has no meaning in SQL Server. Use SELECT ... INTO ... syntax to copy records to server side.
- FindKey, FindNearest, GotoCurrent, GotoKey, GotoNearest, EditKey, SetKey - use TMemDataSet.Locate and TMemDataSet.LocateEx.
- GetDetailLinkFields - use TCustomDADataSet.DetailFields, TCustomDADataSet.MasterFields.
- RenameTable - use 'RENAME TABLE ...' script.
- ConstraintCallBack, ConstraintsDisabled, DisableConstraints, EnableConstraints - has no meaning in SQL Server.
- FlushBuffers - not supported by SDAC.
- Translate - use AnsiToNative and similar functions.

TSession

SDAC does not need global management of a group of database connections in an application.

TUpdateSQL

A complete analogue to TMSUpdateSQL.

16.5 SQL Server Compact Edition

SDAC supports SQL Server Compact Edition. SQL Server Compact Edition is an easy to install server for using by applications that do not require multi-user work with server. For example, SQL Server Compact Edition can be used on the local computers if there is no permanent connection to the main database, for money access machines, automatic cash desks, different electronic facilities and so on. Please refer to SQL Server Compact Edition Books Online for more details about the features and usage of this server edition.
To work with SQL Server Compact Edition you should change the Provider property of the connection options to prCompact, or use the TMSCompactConnection component.
Database filename should be assigned to the Database property. If the file does not exists, it will be automatically created on the connection opening. The Password property is used to connect to or create a database.
Use the Encrypt option to specify if a database will be created or encrypted. If this option is set to True, the Password property must be assigned.
The TMSStoredProc component can not work with the SQL Server Compact Edition because this server edition does not support stored procedures.
Not all values of ObjectType property are supported by TMSMetaData component with the SQL Server Compact Edition.
Please refer to MSSQLCompact Demo to get a sample.

See Also

- MSSQLCompact Demo
- TMSCompactConnection
- TMSConnection.Options

16.6 Working with User Defined Types (UDT)

What is UDT

Microsoft SQL Server 2005 introduced a new possibility to extend the standard type set with data types created in CLR. These types can be used to define columns in tables and variables, in triggers, stored procedures, and functions. UDT is an assembly containing a class written on any programming language. The language must support .NET Framework version 2.0 or higher.

SQL Server requirements

User Defined Types are supported by SQL Server 2005 and higher versions. It is necessary to make sure that CLR integration of the server is enabled. By default in SQL Server 2005 it is disabled. You can check whether the CLR integration is currently enabled running the following system routine:

```
sp_configure 'clr enabled'
```

This routine returns a dataset. If the value of the `run_value` field equals to 0, you need to enable CLR running the same routine with one additional parameter:

```
sp_configure 'clr enabled', 1
GO
RECONFIGURE
GO
```

Calling `RECONFIGURE` is required when applying a new value.

Client requirements

Here are the client requirements:

- .NET Framework version 2.00 or higher;
- Microsoft SQL Native Client;
- the assembly implementing UDT in an accessible place;
- the *Devart.Sdac.UDTProxy.dll* in an accessible place. This file can be found in the Bin subfolder of the SDAC installation directory.

An accessible place means one of the following locations:

1. The application directory (the preferable way);
2. Global Assembly Cache (GAC);
3. Any directory registered in the PATH system variable.

Creating UDT

You can create an assembly containing UDT using any programming language that supports .NET Framework version 2.00 and higher.
There is an example of a UDT implementation within SDAC demos. You can find it in the UDT demo directory of the SDAC General demo. The sample UDT is called Square. It is implemented with Microsoft Visual Studio 2005. There are both sources and the binary assembly in the Square directory.
UDTs can also be written in Pascal using Delphi 2007 for .NET.

Using UDT

UDT can be used only in Win32 application.
In order to use a UDT in SQL Server, first of all you should register the UDT. This can be done by executing CREATE ASSEMBLY command. After the assembly was registered, it is necessary to create a new type from the registered assembly using the CREATE TYPE command. Now you can use the name of the registered type in SQL commands like CREATE TABLE.
In order to provide native access to the UDT fields in your application, the client requirements should be fulfilled. Also make sure that TMSConnection is setup to use SQL Native Client as a provider (TMSConnection.Options.Provider should be equal to prNativeClient). If all settings are applied correctly, UDT fields are described as mapped to theTMSUDTField class. Otherwise, UDT fields are mapped to TVarBytesField. You can access specific properties of UDT using the AsUDT property of TMSUDTField. It may look like the code below (this code is taken from the UDT demo project included in the SDAC General demo):

```
var
  Square: variant;
begin
```

```
MSQuery.Edit;
Square := (MSQuery.FieldByName('c_square') as TMSUDTField).AsUDT;
Square.BaseX := StrToInt(edBaseX.Text);
Square.BaseY := StrToInt(edBaseY.Text);
Square.Side := StrToInt(edSide.Text);
MSQuery.Post;
```

BaseX, BaseY, and Length are properties of the Square class.
For an example see the UDT demo project included in the SDAC General demo.

See Also

- MSSQLCompact Demo
- TMSCompactConnection
- TMSConnection.Options

16.7 TMSTransaction Component

The TMSTransaction component is designed to manage distributed transactions. Distributed transactions can be performed to one or more connections connected to the same or to different databases or servers. Within each connection a separate branch of the transaction is performed. TMSTransaction is based on the Microsoft Distributed Transaction Coordinator (MSDTC) functionality. Transactions can be managed by StartTransaction, Commit, and Rollback methods of TMSTransaction. For more information on distributed transactions and MSDTC please refer to MSDN.
TMSTransaction does not support local transactions.To control local transactions you should use methods of the TMSConnection component.
The example below demonstrates using distributed transaction coordinated by Microsoft Distributed Transaction Coordinator:

```
begin
  MSConnection1.Connect;
  MSConnection2.Connect;
  MSTransaction.AddConnection(MSConnection1);
  MSTransaction.AddConnection(MSConnection2);
  MSTransaction.StartTransaction;
  MSSQL1.Connection := MSConnection1;
  MSSQL2.Connection := MSConnection2;
  try
    MSSQL1.Execute;
    MSSQL2.Execute;
    MSTransaction.Commit;
  except
    MSTransaction.Rollback;
  end;
end;
```

After both connections are established, they are added to the list of connections managed by MSTransaction. Call to StartTransaction makes both TMSConnections components work in the same distributed transaction. After MSSQL1 and MSSQL2 are executed, MSTransaction.Commit ensures that all changes to both databases are committed. If an exception occurs during execution, MSTransaction.Rollback restores both databases to their initial state.

See Also

- TMSTransaction

16.8 Performance of Obtaining Data

If you need to obtain an updatable recordset in your application and show it in a grid, the size of the data to be transferred from the server is very important. As a rule such recordsets are not that big, as it is hard for a user to handle tables containing thousands of records. In this case the most appropriate is the default SDAC behavior, when the CursorType property of the dataset is set to ctDefaultResultSet, and the FetchAll property is set to True.
Just the the same settings must be used, irrespectively of the data size, for the datasets serving as lookup sources for lookup fields.
If you want to see the result of a query execution returning a large amount of data immediately, you should set the FetchAll property to False, or use server cursors. In both cases only few records are fetched to the client immediately after opening. Other records are fetched on demand.
There are brief descriptions of advantages and disadvantages for different settings below.

- `CursorType = ctDefaultResultSet, FetchAll = True` - This is the default SDAC settings. Opening is pretty slow, but navigation is fast. All records are fetched on opening, and cached on the client.
- `CursorType = ctDefaultResultSet, FetchAll = False` - Opening is fast irrespectively of the total records count. Only several records are fetched on opening. You can specify the number of records in the FetchRows property. Other records are retrieved from the server on demand, and cached. Additional records may be demanded when scrolling through the linked grid, calling to Locate, Last, etc.
  However, these settings may cause certain problems related transaction conflict and deadlocks. For more details please refer to the description of the TCustomMSDataSet.FetchAll property.
- `CursorType in [ctStatic, ctKeyset, ctDynamic]` - All these cursors are server cursors. They are characterized by quick opening, low client memory utilization, and slow navigation. Only data required at the moment is cached. For more details about these cursor types please refer to the description of the CursorType property.

If you need to get only certain values from the server, for example only record count, it is more effective to execute a query with parameters:

```
SET :Cnt = (SELECT COUNT(*) FROM ...)
```

instead of queries like this one:

```
SELECT COUNT(*) FROM ...
```

Note: Only the ctDefaultResultSet cursor allows executing batches of queries.

See Also

- FetchAll
- CursorType
- The ServerCursors demo in the SDAC General demo
- The FetchAll demo in the SDAC General demo

16.9 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
```

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), SDAC 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

16.10 Disconnected Mode

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

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

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

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

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

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.

See Also

- TCustomDAConnection.Options
- FetchAll
- Devart.Sdac.TMSQuery.LockMode
- TCustomDAConnection.Pooling
- TCustomDAConnection.Connect
- TCustomDAConnection.Disonnect
- Working in unstable network

16.11 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, SDAC 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 SQL Server database:

```
CREATE TABLE DECIMAL_TYPES
(
  ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
  VALUE1 decimal(4, 0),
  VALUE2 decimal(10, 0),
  VALUE3 decimal(15, 0),
  VALUE4 decimal(5, 2),
  VALUE5 decimal(10, 4),
  VALUE6 decimal(15, 6)
)
```

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:

SQL Server data type	Default Delphi field type	Destination Delphi field type
decimal(4,0)	ftFloat	ftSmallint
decimal(10,0)	ftFloat	ftInteger
decimal(15,0)	ftFloat	ftLargeint
decimal(5,2)	ftFloat	ftFloat
decimal(10,4)	ftFloat	ftBCD
decimal(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       := msDecimal;
  MinPrecision := 0;
  MaxPrecision := 4;
  MinScale     := 0;
  MaxScale     := 0;
  FieldType    := ftSmallint;
  MSConnection.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecision, MinScale, MaxS
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:

```
// clear existing rules
MSConnection.DataTypeMap.Clear;
// rule for decimal(4,0)
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0,      4, 0,     0, ftSmallint);
// rule for decimal(10,0)
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 5,     10, 0,     0, ftInteger);
// rule for decimal(15,0)
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 11, rlAny, 0,     0, ftLargeint);
// rule for decimal(5,2)
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0,      9, 1, rlAny, ftFloat);
// rule for decimal(10,4)
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 10, rlAny, 1,     4, ftBCD);
// rule for decimal(15,6)
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 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 SQL Server database:

```
CREATE TABLE DECIMAL_TYPES
(
  ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
  VALUE1 decimal(5, 2),
  VALUE2 decimal(10, 4),
  VALUE3 decimal(15, 6)
)
```

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

SQL Server data type	Default Delphi field type	Destination field type
decimal(5,2)	ftFloat	ftFloat
decimal(10,4)	ftFloat	ftBCD
decimal(15,6)	ftFloat	ftFMTBCD

If rules are set in the following way:

```
MSConnection.DataTypeMap.Clear;
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0,     9, rlAny, rlAny, ftFloat);
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0, rlAny,     0,     4, ftBCD);
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0, rlAny,     0, rlAny, ftFMTBCD);
```

it will lead to the following result:

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

But if rules are set in the following way:

```
MSConnection.DataTypeMap.Clear;
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0, rlAny,     0, rlAny, ftFMTBCD);
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0, rlAny,     0,     4, ftBCD);
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0,     9, rlAny, rlAny, ftFloat);
  SQL Server data type
```

Delphi field type decimal(5,2) ftFMTBCD decimal(10,4) ftFMTBCD decimal(15,6) ftFMTBCD

This happens because the rule

```
MSConnection.DataTypeMap.AddDBTypeRule(msDecimal, 0, rlAny,     0, rlAny, ftFMTBCD);
```

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

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

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

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

Defining rules for Connection and Dataset

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

For example, such table is created in SQL Server:

```
CREATE TABLE PERSON
(
  ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
  FIRSTNAME varchar(20),
  LASTNAME varchar(30),
  GENDER_CODE varchar(1),
  BIRTH_DTTM datetime
)
```

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

```
MSConnection.DataTypeMap.Clear;
MSConnection.DataTypeMap.AddDBTypeRule(msDateTime, 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:

```
MSQuery.DataTypeMap.Clear;
MSQuery.DataTypeMap.AddDBTypeRule(msDateTime, 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:

```
MSConnection.DataTypeMap.Clear;
MSConnection.DataTypeMap.AddDBTypeRule(msDateTime, ftDate);
MSQuery.DataTypeMap.Clear;
MSQuery.DataTypeMap.AddDBTypeRule(msDateTime, ftDateTime);
```

In this case, in all DataSets for the DATETIME type fields with the ftDate type will be created, and for MSQuery - 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 MySQL database:

```
CREATE TABLE ITEM
(
  ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
  NAME CHAR(50),
  GUID CHAR(38)
)
```

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:

```
MSQuery.DataTypeMap.Clear;
MSQuery.DataTypeMap.AddDBTypeRule(msChar, 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:

```
MSQuery.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:

```
MSConnection.DataTypeMap.AddDBTypeRule(msVarchar, 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.

16.12 Data Encryption

SDAC 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 int IDENTITY (1,1) NOT NULL PRIMARY KEY,
  ENAME varbinary(2000),
  HIREDATE varbinary(200),
  SAL varbinary(200),
  FOTO image
)
```

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.

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

16.13 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 TMSSQL component is more preferable than TMSQuery. It can give several additional percents performance gain.
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.

Open

If you don't need to edit the dataset, you can set its ReadOnly property to increase its opening speed. In that case, an additional information, required for INSERT, UPDATE, and DELETE statement generation, will not be requested.

Fetch

In some situations you can increase performance a bit by using P:Devart.Dac.TDADataSetOptions. CompressBlobMode.
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.
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.Sdac.TMSLoader component instead of Insert/Post methods, or execution of the INSERT commands multiple times in a cycle. Sometimes usage of TDevart.Sdac.TMSLoader improves performance several times.

See Also

- Performance of obtaining data

16.14 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 SDAC 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;
```

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

See Also

- TMacro
- TCustomDADataSet.MacroByName
- TCustomDADataSet.Macros

16.15 Using Table-Valued Parameters

Table-valued parameters are a new parameter type introduced in SQL Server 2008. They can be used to send multiple rows of data to a Transact-SQL statement or a stored routine without creating a temporary table or many parameters. To learn more on table-valued parameters, see http://msdn.microsoft.com/en-us/library/bb510489.aspx

This topic demonstrates how to use table-valued parameters in your application by the help of SDAC

1. In order to pass a table as a parameter to a stored procedure or function, create a TABLE TYPE as follows:

```
CREATE TYPE DeptTableType AS TABLE(
  DNAME VARCHAR(20),
  LOC VARCHAR(20)
)
```

2. In a stored procedure we will transfer data from a parameter to a table on a server. Here is a script example for creating a table:

```
CREATE TABLE DEPT(
  DEPTNO INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  DNAME VARCHAR(20) NULL,
  LOC VARCHAR(20) NULL
)
```

3. Create a stored procedure that uses the table type:

```
CREATE PROCEDURE SP_InsertDept
  @TVP DeptTableType READONLY
AS
BEGIN
  INSERT INTO DEPT ([DNAME], [LOC])
    SELECT * FROM @TVP
END
```

4. To work with Table-Valued Parameters, you should use the TMSTableData component. Fill it with data:

```
MSTableData.TableTypeName := 'DeptTableType';
MSTableData.Open;
MSTableData.Append;
MSTableData.Fields[0].AsString := 'ACCOUNTING';
MSTableData.Fields[1].AsString := 'NEW YORK';
MSTableData.Post;
MSTableData.Append;
MSTableData.Fields[0].AsString := 'RESEARCH';
MSTableData.Fields[1].AsString := 'DALLAS';
MSTableData.Post;
MSTableData.Append;
MSTableData.Fields[0].AsString := 'SALES';
MSTableData.Fields[1].AsString := 'CHICAGO';
MSTableData.Post;
MSTableData.Append;
MSTableData.Fields[0].AsString := 'OPERATIONS';
MSTableData.Fields[1].AsString := 'BOSTON';
MSTableData.Post;
```

5. Use the TMSStoredProc component to transfer data from the TMSTableData component to a table on a server:

```
MSStoredProc.StoredProcName := 'SP_InsertDept';
MSStoredProc.PrepareSQL;
MSStoredProc.ParamByName('TVP').AsTable := MSTableData.Table;
MSStoredProc.ExecProc;
MSTableData.Close;
```

16.16 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

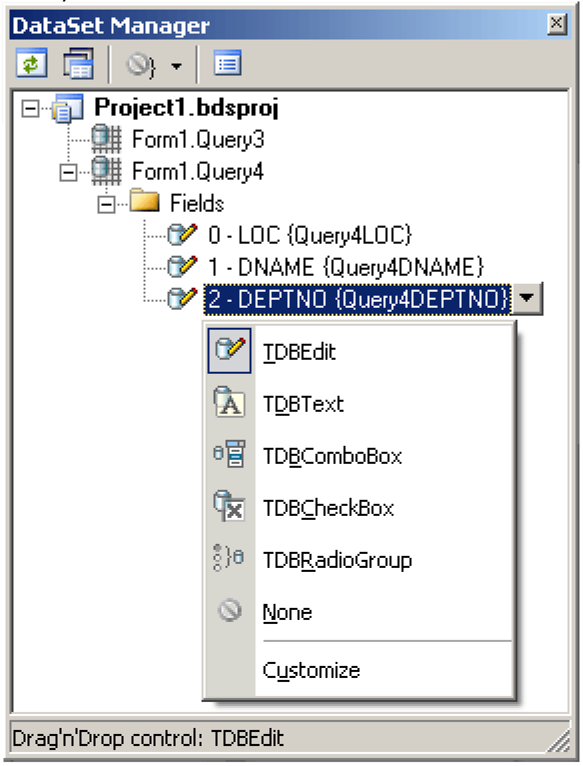
Assemblies:
- Devart.Dac.dll
- Devart.Vcl.dll
- Devart.Dac.Design.dll
- Devart.Dac.AdoNet.dll

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.

16.17 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.

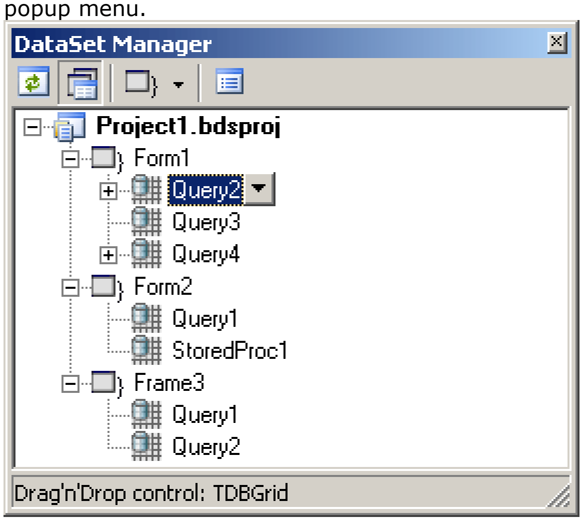
Opening the DataSet Manager window

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

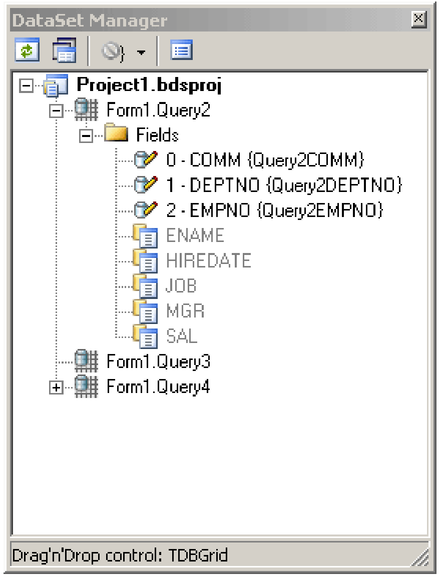
By default DataSet Manager shows DataSets of currently open forms. It can also extract DataSets from all forms in the project. To use this, click *Extract DataSets from all forms in project* button. This settings is remembered. Note, that using this mode can slow down opening of the large projects with 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

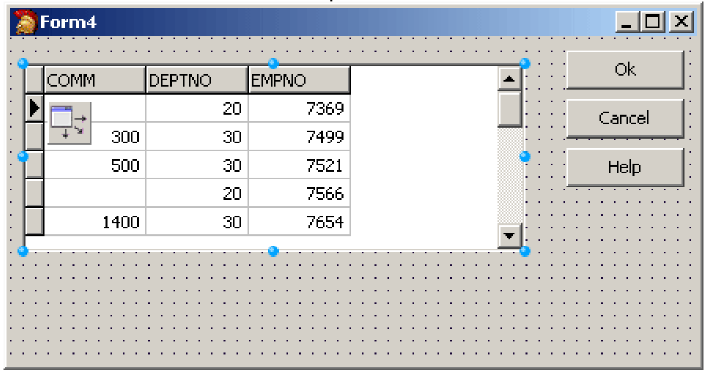
DataSet Manager window popup menu.



Creating Data-bound Controls

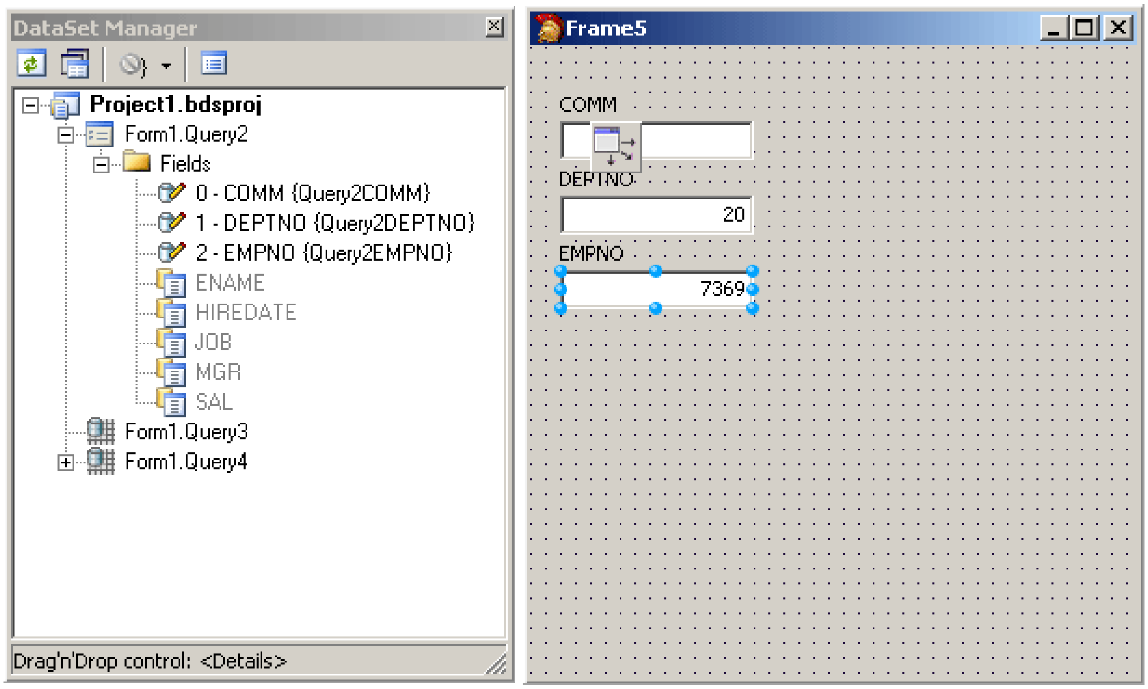
You can drag an item from the DataSet Manager window onto a form to create a new data-bound control. Each node in the DataSet Manager window allows you to choose the type of control that will be created when you drag it onto a form. You must choose between a Grid layout, where all columns or properties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





COMM	DEPTNO	EMPNO
	20	7369
300	30	7499
500	30	7521
	20	7566
1400	30	7654

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 wi ard. DataSet Manager will create TDataSource component, and a component and a label for each field.



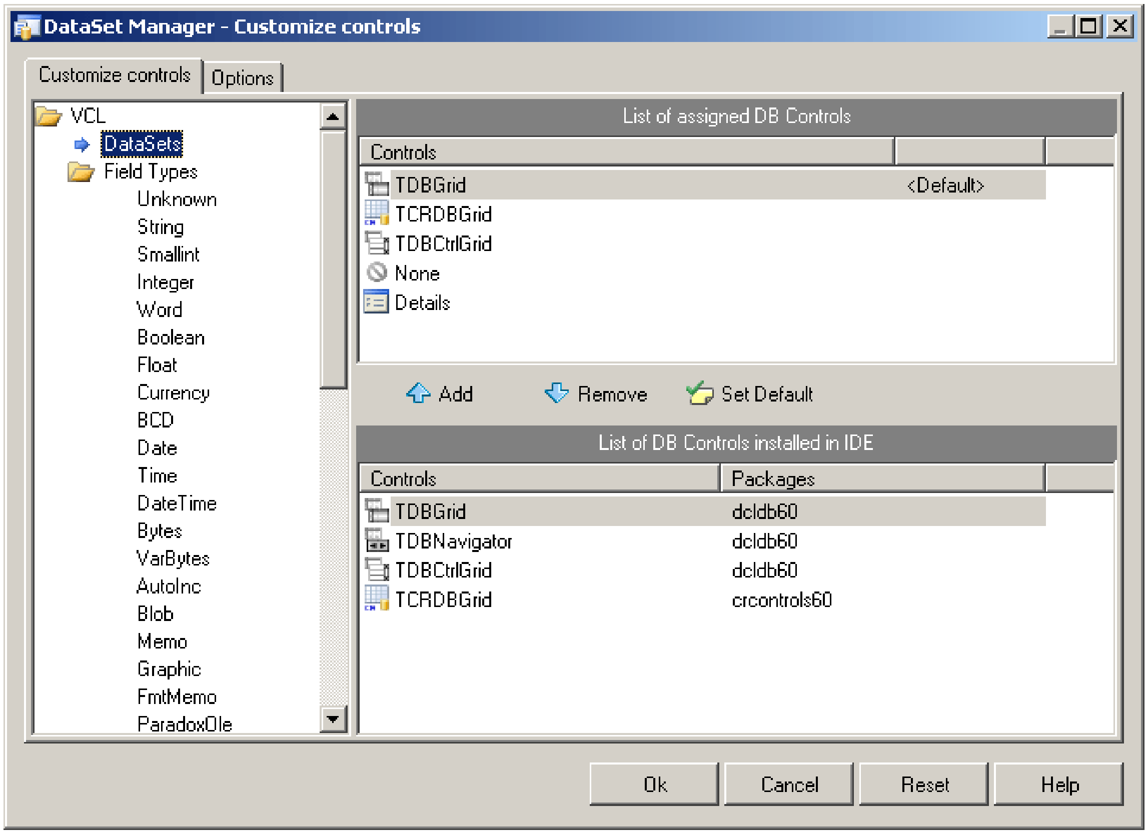
COMM

DEPTNO 20

EMPNO 7369

Adding custom controls to the DataSet Manager window

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



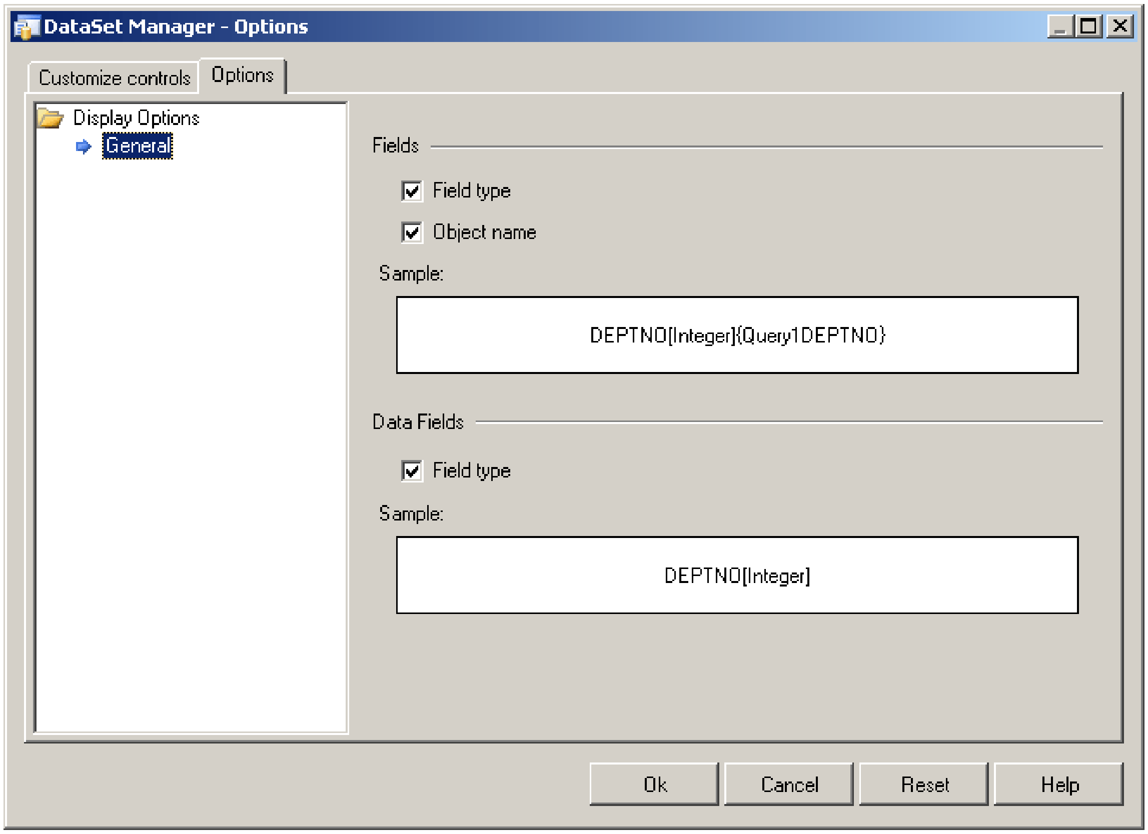
Controls	
TDBGrid	<Default>
TCRDBGrid	
TDBCtrlGrid	
None	
Details	

Controls	Packages
TDBGrid	dcldb60
TDBNavigator	dcldb60
TDBCtrlGrid	dcldb60
TCRDBGrid	crcontrols60

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.



Fields

☑ Field type

☑ Object name

Sample:

DEPTNO[Integer]{Query1DEPTNO}

Data Fields

☑ Field type

Sample:

DEPTNO[Integer]

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

16.18 DBMonitor

To extend monitoring capabilities of SDAC applications there is an additional tool called DBMonitor. It is provided as an alternative to Borland SQL Monitor which is also supported by SDAC.
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 TMSSQLMonitor 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.

16.19 Writing GUI Applications with SDAC

SDAC 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 SdacVcl unit in your application. This feature is needed for writing console applications.

Delphi and C++Builder

By default SDAC does not require Forms, Controls and other GUI related units. Only TMSConnectDialog and TMSAlerter components require the Forms unit.

16.20 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 , Database, IsolationLevel, Authentication, QuotedIdentifier, Provider, Language, Encrypt, PersistSecurityInfo, AutoTranslate, NetworkLibrary, ApplicationName, WorkstationID, PacketSize . 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.

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 TMSConnectionPoolManager. 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.

See Also

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

16.21 Compatibility with Previous Versions

We always try to keep SDAC compatible with previous versions, but sometimes we have to change the behaviour of SDAC in order to enhance its functionality, or avoid bugs. This topic describes such changes, and how to revert the old SDAC behaviour. We strongly recommend not to turn on the old behaviour of SDAC. Use options described below only if changes applied to SDAC 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 SDAC 3.55.2.22. To restore old behavior, set the BaseSQLOldBehavior variable to True.

DBAccess.SQLGeneratorCompatibility:

If the manually assigned RefreshSQL property contains only "WHERE" clause, SDAC 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 SDAC 4.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

MemDS.SendDataSetChangeEventAfterOpen:

Starting with SDAC 4.20.0.12, the DataSetChange event is sent after the dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.
To disable sending this event, change the value of this variable to False.

MemDS.DoNotRaiseExcetionOnUaFail:

Starting with SDAC 4.20.0.13, 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.

MSAccess. UseUpdateOptimization

In SDAC 4.00.0.4 update statements execution was optimized. This optimization changed behaviour of affected rows count retrieval for tables with triggers. If a trigger performs modifications of other records reacting on a modification in the underlying table, SQL Server sends several values of affected rows count (including for modifications made by the trigger). Prior to SDAC 4.00.0.4 the first value was considered as affected rows count, when in SDAC 4.00.0.4 and higher - the last value. However neither of these two approaches can be considered correct, as there can be triggers that snap into action both before modification and after modification. There is no way to determine which of the values returned by SQL Server is the correct value of affected rows count. Therefore we do not recommend using the RowsAffected property when updating tables with triggers.
StrictUpdate mode is based on RowsAffected, therefore we also do not recommend using StrictUpdate when updating tables with triggers.
If you want to disable this optimization, set the UseUpdateOptimization variable to False.

TCustomMSConnectionOptions.UseWideMemos:

Set TCustomMSConnectionOptions.UseWideMemos to False to disable mapping of NText database data type to ftWideMemo data type.

OLEDBAccess.ParamsInfoOldBehavior:

Starting with SDAC 3.70.1.26 preparing and the first call of a stored procedure were combined for performance optimization. This requires the necessity of setting the parameter type and data type of all parameters before preparing. In order to revert the old behaviour with preparation and parameters, the OLEDBAccess unit should be added to the uses clause of a unit in an application, and the following line should be added to the initialization section of the unit:

```
ParamsInfoOldBehavior := True.
```

DBAccess.ParamStringAsAnsiString:

This variable has sense for Delphi 2009 and higher.
Set its value to True to use the AsAnsiString property when setting the parameter value through TDAParam.AsString. Otherwise the AsWideString property is used. The default value is False.

16.22 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:

TYPE	32-bit	64-bit
Extended	10 bytes	8 bytes

The following two additional types are introduced to ensure compatibility in the process of developing 32- and 64-bit applications:

Extended80 – whose size in 32-bit application is 10 bytes; however, this type provides the same precision as its 8-byte equivalent in 64-bit applications.

Extended80Rec – can be used to perform low-level operations on an extended precision floating-point value. For example, the sign, the exponent, and the mantissa can be changed separately. It enables you to perform memory-related operations with 10-bit floating-point variables, but not extended-precision arithmetic operations.

Pointer and Integers

The major difference between 32- and 64-bit platforms is the volume of the used memory and, correspondingly, the size of the pointer that is used to address large memory volumes.

TYPE	32-bit	64-bit
Pointer	4 bytes	8 bytes

At the same time, the size of the Integer type remains the same for both platforms:

TYPE	32-bit	64-bit

Integer	4 bytes	4 bytes

That is why, the following code will work incorrectly on the 64-bit platform:

```
Ptr := Pointer(Integer(Ptr) + Offset);
```

While this code will correctly on the 64-bit platform and incorrectly on the 32-bit platform:

```
Ptr := Pointer(Int64(Ptr) + Offset);
```

For this purpose, the following platform-dependent integer type is introduced:

TYPE	32-bit	64-bit
NativeInt	4 bytes	8 bytes
NativeUInt	4 bytes	8 bytes

This type helps ensure that pointers work correctly both for the 32- and 64-bit platforms:

```
Ptr := Pointer(NativeInt(Ptr) + Offset);
```

However, you need to be extra-careful when developing applications for several versions of Delphi, in which case you should remember that in the previous versions of Delphi the NativeInt type had different sizes:

TYPE	Delphi Version	Size
NativeInt	D5	N/A
NativeInt	D6	N/A
NativeInt	D7	8 bytes
NativeInt	D2005	8 bytes
NativeInt	D2006	8 bytes
NativeInt	D2007	8 bytes
NativeInt	D2009	4 bytes
NativeInt	D2010	4 bytes
NativeInt	Delphi XE	4 bytes
NativeInt	Delphi XE2	4 or 8 bytes

Out parameters

Some WinAPIs have OUT parameters of the SIZE_T type, which is equivalent to NativeInt in Delphi XE2. The problem is that if you are developing only a 32-bit application, you won't be able to pass Integer to OUT, while in a 64-bit application, you will not be able to pass Int64; in both cases you will have to pass NativeInt.
For example:

```
procedure MyProc(out Value: NativeInt);
begin
  Value := 12345;
end;
var
    Value1: NativeInt;
{$IFDEF WIN32}
  Value2: Integer;
{$ENDIF}
{$IFDEF WIN64}
  Value2: Int64;
{$ENDIF}
begin
   MyProc(Value1); // will be compiled;
 MyProc(Value2); // will not be compiled !!!
end;
```

Win API

If you pass pointers to SendMessage/PostMessage/TControl.Perform, the wParam and lParam parameters should be type-casted to the WPARAM/LPARAM type and not to Integer/Longint.
Correct:

```
SendMessage(hWnd, WM_SETTEXT, 0, LPARAM(@MyCharArray));
```

Wrong:

```
SendMessage(hWnd, WM_SETTEXT, 0, Integer(@MyCharArray));
```

Replace SetWindowLong/GetWindowLog with SetWindowLongPtr/GetWindowLongPtr for GWLP HINSTANCE, GWLP ID, GWLP USERDATA, GWLP HWNDPARENT and GWLP WNDPROC as they return pointers and handles. Pointers that are passed to SetWindowLongPtr should be type-casted to LONG PTR and not to Integer/Longint.
Correct:

```
SetWindowLongPtr(hWnd, GWLP_WNDPROC, LONG_PTR(@MyWindowProc));
```

Wrong:

```
SetWindowLong(hWnd, GWL_WNDPROC, Longint(@MyWindowProc));
```

Pointers that are assigned to the TMessage.Result field should use a type-cast to LRESULT instead of Integer/Longint.
Correct:

```
Message.Result := LRESULT(Self);
```

Wrong:

```
Message.Result := Integer(Self);
```

All TWM...-records for the windows message handlers must use the correct Windows types for the fields:

```
Msg: UINT; wParam: WPARAM; lParam: LPARAM; Result: LRESULT)
```

Assembler
In order to make your application (that uses assembly code) work, you will have to make several changes to it:

- rewrite your code that mixes Pascal code and assembly code. Mixing them is not supported in 64-bit applications;
- rewrite assembly code that doesn't consider architecture and processor specifics.

You can use conditional defines to make your application work with different architectures.
You can learn more about Assembly code here: http://docwiki.embarcadero.com/RADStudio/en/Using_Inline_Assembly_Code You can also look at the following article that will help you to make your application support the 64-bit platform: http://docwiki.embarcadero.com/RADStudio/en/Converting_32-bit_Delphi_Applications_to_64-bit_Windows

Exception handling
The biggest difference in exception handling between Delphi 32 and 64-bit is that in Delphi XE2 64-bit you will gain more performance because of different internal exception mechanism. For 32-bit applications, the Delphi compiler (dcc32.exe) generates additional code that is executed any way and that causes performance loss. The 64-bit compiler (dcc64.exe) doesn't generate such code, it generates metadata and stores it in the PDATA section of an executable file instead.
But in Delphi XE2 64-bit it's impossible to have more than 16 levels of nested exceptions. Having more than 16 levels of nested exceptions will cause a Run Time error.

Debugging
Debugging of 64-bit applications in RAD Studio XE2 is remote. It is caused by the same reason: RAD Studio XE2 IDE is a 32 application, but your application is 64-bit. If you are trying to debug your application and you cannot do it, you should check that the **Include remote debug symbols** project option is enabled.
To enable it, perform the following steps:

1. Open Project Options (in the main menu **Project->Options**).
2. In the Target combobox, select **Debug configuration - 64-bit Windows platform**. If there is no such option in the combobox, right click "Target Platforms" in Project Manager and select **Add platform**. After adding the 64-bit Windows platform, the **Debug configuration - 64-bit Windows platform** option will be available in the Target combobox.
3. Select **Linking** in the left part of the Project Options form.
4. enable the **Include remote debug symbols** option.

After that, you can run and debug your 64-bit application.
To enable remote debugging, perform the following steps:

1. Install Platform Assistant Server (PAServer) on a remote computer. You can find PAServer in the %RAD Studio XE2 Install Directory%\PAServer directory. The setup paserver.exe file is an installation file for Windows, and the setup paserver. ip file is an istallation file for MacOS.
2. Run the PAServer.exe file on a remote computer and set the password that will be used to connect to this computer.
3. On a local computer with RAD Studio XE2 installed, right-click the target platform that you want to debug in Project Manager and select **Assign Remote Profile**. Click the **Add** button in the displayed window, input your profile name, click the **Next** button, input the name of a remote

computer and the password to it (that you assigned when you started PAServer on a remote computer).

After that, you can test the connection by clicking the **Test Connection** button. If your connection failed, check that your firewalls on both remote and local computers do not block your connection, and try to establish a connection once more. If your connection succeeded, click the Next button and then the Finish button. Select your newly created profile and click **OK**.
After performing these steps you will be able to debug your application on a remote computer. You application will be executed on a remote computer, but you will be able to debug it on your local computer with RAD Studio XE2.
For more information about working with Platform Assistant Server, please refer to http://docwiki.embarcadero.com/RADStudio/en/
Installing_and_Running_the_Platform_Assistant_on_the_Target_Platform

16.23 Database Specific Aspects of 64-bit Development

SQL Server Connectivity Aspects

If you are working in the Direct mode or developing a 32-bit application only, then the development process will not be different for you, except some peculiarities of each particular platform. But if you are developing a 64-bit application, you have to be aware of specifics of working with client libraries at design-time and run-time. To connect to a SQL Server database at design-time, you must have its 32-bit client library. You have to place it to the C:\Windows\SysWOW64 directory. This requirement flows out from the fact that RAD Studio XE2 is a 32-bit application and it cannot load 64-bit libraries at design-time. To work with a SQL Server database at run-time (64-bit application), you must have the 64-bit client library placed to the C:\Windows\System32 directory.

16.24 FILESTREAM Data

FILESTREAM is a feature of SQL Server 2008, which allows storage of and efficient access to BLOB data using a combination of SQL Server 2008 and the NTFS file system.
This topic demonstrates how to work with FILESTREAM data with the help of SDAC.
To work with FILESTREAM data, you should have an appropriate table on a server. SQL Server requires a table to have a column of the UNIQUEIDENTIFIER data type that has the ROWGUIDCOL attribute to be an appropriate one for working with FILESTREAM data. This column must not allow NULL values and must have either a UNIQUE or PRIMARY KEY single-column constraint. A FILESTREAM column must be defined as a VARBINARY(MAX) column that has the FILESTREAM attribute.
Here is an example of a script to create a correct table:

```
CREATE TABLE TESTFS(
  ID INT PRIMARY KEY NOT NULL,
  FS VARBINARY(MAX) FILESTREAM NULL,
  GD UNIQUEIDENTIFIER UNIQUE ROWGUIDCOL NOT NULL DEFAULT NEWID()
)
```

The FILESTREAM data is represented by a file on a computer where SQL Server is installed. In order to start working with it, you should insert any value into your FILESTREAM column. This will create a new file on a server and it will be possible to work with it. Here is an example that demonstrates it:
Delphi:

```
MSQuery.SQL.Text := 'SELECT * FROM TESTFS';
MSQuery.Open;
MSQuery.Append;
MSQuery.FieldByName('ID').AsInteger := 1;
MSQuery.FieldByName('FS').AsString := 'TEST';
MSQuery.Post;
```

C++Builder:

```
MSQuery->SQL->Text = "SELECT * FROM TESTFS";
MSQuery->Open();
MSQuery->Append();
MSQuery->FieldByName("ID")->AsInteger = 1;
MSQuery->FieldByName("FS")->AsString = "TEST";
MSQuery->Post();
```

After the steps above have been performed, it is possible to work with FILESTREAM data. Here is an example that demonstrates it:
Delphi:

```
procedure TMainForm.BitBtnRunClick(Sender: TObject);
var
  con: TMSConnection;
  qr: TMSQuery;
  fs: TMSFileStream;
  ts: AnsiString;
begin
  con := TMSConnection.Create(nil);
  qr := TMSQuery.Create(nil);
  try
    con.Authentication := auWindows; // FILESTREAM requirement
    con.Server := 'server';
    con.Database := 'database';
    qr.Connection := con;
    qr.SQL.Text := 'SELECT * FROM TESTFS';
    qr.Open;
    //writing data
    con.StartTransaction; // FILESTREAM requirement
    fs := qr.GetFileStreamForField('FS', daWrite);
    ts := 'TEST FILESTREAM';
    fs.WriteBuffer(ts[1], Length(ts));
    fs.Flush;
    fs.Close; // it's necessary to call this method before the transaction commits or rolls
    con.Commit;
    //reading data
    con.StartTransaction; // FILESTREAM requirement
    fs := qr.GetFileStreamForField('FS', daRead);
    SetLength(ts, fs.Size);
    fs.ReadBuffer(ts[1], fs.Size);
    ShowMessage(ts);
    fs.Close; // it's necessary to call this method before the transaction commits or rolls
    con.Commit;
  finally
    qr.Free;
    con.Free;
  end;
end;
```

C++Builder:

```
void __fastcall TMainForm::BitBtnRunClick(TObject *Sender)
{
  TMSConnection* con = new TMSConnection(NULL);
  TMSQuery* qr = new TMSQuery(NULL);
  try
  {
    con->Authentication = auWindows; // FILESTREAM requirement
    con->Server = "server";
    con->Database = "database";
    qr->Connection = con;
    qr->SQL->Text = "SELECT * FROM TESTFS";
    qr->Open();
    //writing data
    con->StartTransaction(); // FILESTREAM requirement
    TMSFileStream* fs = qr->GetFileStreamForField("FS", daWrite);
    char* ts = "TEST FILESTREAM";
    fs->WriteBuffer(ts, strlen(ts));
    fs->Flush();
    fs->Close(); // it's necessary to call this method before the transaction commits or ro
    con->Commit();
    //reading data
    con->StartTransaction(); // FILESTREAM requirement
    fs = qr->GetFileStreamForField("FS", daRead);
    ts = new char[fs->Size];
    fs->ReadBuffer(ts, fs->Size);
    ShowMessage(ts);
    fs->Close(); // it's necessary to call this method before the transaction commits or ro
    con->Commit();
  }
  __finally
  {
    qr->Free();
    con->Free();
  }
}
```

As you can see from these examples, you don't need to free TMSFileStream manually. SDAC takes care of freeing all assigned TMSFileStream objects.
Note: You can find more information about working with FILESTREAM data in MSDN at http://msdn.microsoft.com/en-us/library/cc949109(v=sql.100).aspx

See also

- GetFileStreamForField
- Close
- Flush

17 Reference

This page shortly describes units that exist in SDAC.

Units

Unit Name	Description
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.
DADump	This unit contains the base class for the TMSDump component.
DALoader	This unit contains the base class for the TMSLoader component.
DAScript	This unit contains the base class for the TMSScript component.
DASQLMonitor	This unit contains the base class for the TMSSQLMonitor component.
DBAccess	This unit contains base classes for most of the components.
Devart.Dac.DataAdapter	This unit contains implementation of the DADataAdapter class.
Devart.Sdac.DataAdapter	This unit contains implementation of the MSDataAdapter 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.
MSAccess	This unit contains implementation of most public classes of SDAC.
MSCompactConnection	This unit contains implementation of the TMSCompactConnection class.
MSConnectionPool	This unit contains the TMSConnectionPoolManager class for managing connection pool.
MSDump	This unit contains implementation of the TMSDump component.
MSLoader	This unit contains implementation of the TMSLoader component.
MSScript	This unit contains implementation of the TMSScript component.
MSServiceBroker	This unit contains implementation of the TMSServiceBroker component and auxiliary classes.
MSSQLMonitor	This unit contains implementation of the TMSSQLMonitor component.
MSTransaction	This unit contains implementation of the TMSTransaction component.

OLEDBAccess	This unit contains classes for accessing SQL Server through OLE DB providers
SdacVcl	This unit contains the visual constituent of SDAC.
VirtualTable	This unit contains implementation of the TVirtualTable component.

17.1 CRAccess

This unit contains base classes for accessing databases.

Classes

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

Types

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

Enumerations

Name	Description
TCRIsolationLevel	Specifies how to handle transactions containing database modifications.
TCRTransactionAction	Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

17.1.1 Classes

Classes in the **CRAccess** unit.

Classes

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

17.1.1.1 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

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.

17.1.2 Types

Types in the **CRAccess** unit.

Types

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

17.1.2.1 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.

17.1.3 Enumerations

Enumerations in the **CRAccess** unit.

Enumerations

Name	Description
TCRIsolationLevel	Specifies how to handle transactions containing database modifications.
TCRTransactionAction	Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

17.1.3.1 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.

17.1.3.2 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.

17.2 CRBatchMove

This unit contains implementation of the TCRBatchMove component.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

Types

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

Enumerations

Name	Description
TCRBatchMode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
TCRFieldMappingMode	Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.

17.2.1 Classes

Classes in the **CRBatchMove** unit.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

17.2.1.1 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 SQL Server Access page.

Inheritance Hierarchy

TObject
  TCRBatchMove

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.

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

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.

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.

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.

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.

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.

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.

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

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
```

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.

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.

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

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.

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.

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

Performs the batch operation.

Class

TCRBatchMove

Syntax

```
procedure Execute;
```

Remarks

Call the Execute method to perform the batch operation.

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

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.

17.2.2 Types

Types in the **CRBatchMove** unit.

Types

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

17.2.2.1 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.

17.2.3 Enumerations

Enumerations in the **CRBatchMove** unit.

Enumerations

Name	Description
TCRBatchMode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
TCRFieldMappingMode	Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.

17.2.3.1 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.

17.2.3.2 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.

17.3 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 occuring 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

17.3.1 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 occuring 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

17.3.1.1 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

EDataMappingError class overview.

17.3.1.2 CRDataTypeMap.EDataTypeMappingError Class

Base class for errors occuring 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 occuring at data mapping

Inheritance Hierarchy

TObject
  EDataTypeMappingError

EDataTypeMappingError class overview.

17.3.1.3 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

EInvalidDBTypeMapping class overview.

17.3.1.4 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

EInvalidFieldTypeMapping class overview.

17.3.1.5 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

EUnsupportedDataTypeMapping class overview.

17.3.1.6 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

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

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

Maximum DB field size

Class

TMapRule

Syntax

```
property DBLengthMax: Integer;
```

Minimum DB field size

Class

TMapRule

Syntax

```
property DBLengthMin: Integer;
```

Maximum DB field scale

Class

TMapRule

Syntax

```
property DBScaleMax: Integer;
```

Minimal DB field scale

Class

TMapRule

Syntax

```
property DBScaleMin: Integer;
```

DB type

Class

TMapRule

Syntax

```
property DBType: Word;
```

Delphi field length

Class

TMapRule

Syntax

```
property FieldLength: Integer;
```

---

field name in DataSet

Class

TMapRule

Syntax

```
property FieldName: string;
```

Delphi field scale

Class

TMapRule

Syntax

```
property FieldScale: Integer;
```

Ignore data conversion errors. Default value is False.

Class

TMapRule

Syntax

```
property IgnoreErrors: Boolean;
```

17.4 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.

17.4.1 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.

17.4.1.1 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

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.

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

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.

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.

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.

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.

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.

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

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.

17.4.2 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.

17.4.2.1 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.

17.4.2.2 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.

17.4.2.3 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.

17.4.2.4 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.

17.5 DADump

This unit contains the base class for the TMSDump 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.

17.5.1 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.

17.5.1.1 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 descedants to dump database objects, such as tables, stored procedures, and functions for backup or for transferring the data to another SQL server. The dump contains SQL statements to create the table or other database objects and/or populate the table.

Inheritance Hierarchy

TObject
  TDADump

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 SQL Server raises some error on TDADump.Restore.
OnRestoreProgress	Occurs to indicate the TDADump.Restore, TDADump.RestoreFromFile, or TDADump.RestoreFromStream method execution progress.

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

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

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

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.

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

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

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

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

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

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

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

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

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

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

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 SQL Server 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

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 ero. 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

Occurs when SQL Server raises some error on Restore.

Class

TDADump

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when SQL Server 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.

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

17.5.1.2 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

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.

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

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.

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.

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.

17.5.2 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.

17.5.2.1 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 ero.

ObjectCount
The quantity of database objects to dump.

Percent
The current percentage of the current table data dumped.

17.5.2.2 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.

17.6 DALoader

This unit contains the base class for the TMSLoader 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.

17.6.1 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.

17.6.1.1 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

TObject
  TDAColumn

See Also

- TDALoader
- TDAColumns

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.

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

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.

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

17.6.1.2 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

TDAColumns class overview.

Properties

Name	Description
Items	Used to access individual columns.

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

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

17.6.1.3 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

TDALoader class overview.

Properties

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.

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.

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

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

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

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

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

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

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

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.

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.

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. 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

Puts the value of individual columns by the column name.

Class

`TDALoader`

Syntax

```
procedure PutColumnData(const ColName: string; Row: integer; const
 Value: variant); overload
```

Parameters

ColName
Hods the name of a loading column.

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

Value
Holds the column value.

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

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

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

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.

Example

This handler loads 1000 rows.

```
procedure TfmMain.PutData(Sender: TDALoader);
var
   Count: Integer;
   i: Integer;
begin
   Count := StrToInt(edRows.Text);
   for i := 1 to Count dobegin
      Sender.PutColumnData(0, i, 1);
      Sender.PutColumnData(1, i, Random(100));
      Sender.PutColumnData(2, i, Random*100);
      Sender.PutColumnData(3, i, 'abc01234567890123456789');
      Sender.PutColumnData(4, i, Date);
   end;
end;
```

See Also

- TDALoader.PutColumnData
- Load
- OnGetColumnData

17.6.2 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.

17.6.2.1 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.

17.6.2.2 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.

17.6.2.3 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.

17.7 DAScript

This unit contains the base class for the TMSScript component.

Classes

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

Types

Name	Description
TAfterStatementExecuteEvent	This type is used for the TDAScript.AfterExecute event.
TBeforeStatementExecuteEvent	This type is used for the TDAScript.BeforeExecute event.
TOnErrorEvent	This type is used for the TDAScript.OnError event.

Enumerations

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

17.7.1 Classes

Classes in the **DAScript** unit.

Classes

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

17.7.1.1 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

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 SQL Server raises an error.

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

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

Refers to a dataset that holds the result set of query execution.

Class

`TDAScript`

Syntax

```
property DataSet: TCustomDADataSet;
```

Remarks

Set the DataSet property to retrieve the results of the SELECT statements execution inside a script.

See Also

- ExecuteNext
- Execute

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.

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.

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.

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.

Used to get the end position of the current statement.

Class

TDAScript

Syntax

```
property EndPos: Int64;
```

Remarks

Use the EndPos property to get the end position of the current statement (the position of the last character in the statement) in a script.

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

Class

TDAScript

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL script text in design- or run-time. Macros extend abilities of parameters and allow changing conditions in the WHERE clause or sort order in the ORDER BY clause. You just insert &MacroName in a SQL query text and change value of macro by the Macro property editor in design-time or the MacroByName function in run-time. In time of opening query macro is replaced by its value.

See Also

- TMacro
- MacroByName

Used to get or set script text.

Class

TDAScript

Syntax

```
property SQL: _TStrings;
```

Remarks

Use the SQL property to get or set script text.

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

Class

TDAScript

Syntax

```
property StartLine: Int64;
```

Remarks

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

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

Class

TDAScript

Syntax

```
property StartOffset: Int64;
```

Remarks

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

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

Class

TDAScript

Syntax

```
property StartPos: Int64;
```

Remarks

Use the StartPos property to get the start position of the current statement (the position of the first statement character) in a script.

Contains a list of statements obtained from the SQL property.

Class

TDAScript

Syntax

```
property Statements: TDAStatements;
```

Remarks

Contains a list of statements that are obtained from the SQL property. Use the Access Statements property to view SQL statement, set parameters or execute the specified statement. Statements is a ero-based array of statement records. Index specifies the array element to access.
For example, consider the following script:

```
CREATE TABLE A (FIELD1 INTEGER);
INSERT INTO A VALUES(1);
INSERT INTO A VALUES(2);
INSERT INTO A VALUES(3);
CREATE TABLE B (FIELD1 INTEGER);
INSERT INTO B VALUES(1);
INSERT INTO B VALUES(2);
INSERT INTO B VALUES(3);
```

Note: The list of statements is created and filled when the value of Statements property is requested. That's why the first access to the Statements property can take a long time.

Example

You can use the Statements property in the following way:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   i: integer;
begin
   with Script do
   begin
      for i := 0 to Statements.Count - 1 do
         if Copy(Statements[i].SQL, 1, 6) <> 'CREATE' then
 Statements[i].Execute;
   end;
end;
```

See Also

- TDAStatements

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

Stops script execution.

Class

`TDAScript`

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

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

Executes a script.

Class

`TDAScript`

Syntax

```
procedure Execute; virtual;
```

Remarks

Call the Execute method to execute a script. If SQL Server raises an error, the OnError event occurs.

See Also

- ExecuteNext
- OnError
- ErrorOffset

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.

Executes the next statement in the script and then stops.

Class

TDAScript

Syntax

```
function ExecuteNext: boolean; virtual;
```

Return Value

True, if there are any statements left in the script, False otherwise.

Remarks

Use the ExecuteNext method to execute the next statement in the script statement and stop. If SQL Server raises an error, the OnError event occurs.

See Also

- Execute
- OnError
- ErrorOffset

Executes SQL statements contained in a stream object.

Class

TDAScript

Syntax

```
procedure ExecuteStream(Stream: TStream);
```

Parameters

Stream
Holds the stream object from which the statements will be executed.

Remarks

Call the ExecuteStream method to execute SQL statements contained in a stream object. Reading from the stream and execution is performed by blocks of 64k size.

Indicates whether a specified macro exists in a dataset.

Class

TDAScript

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

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

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 SQL Server raises an error.

See Also

- TDAScript Class
- TDAScript Class Members

Occurs after a SQL script execution.

Class

`TDAScript`

Syntax

```
property AfterExecute: TAfterStatementExecuteEvent;
```

Remarks

Occurs after a SQL script has been executed.

See Also

- Execute

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.

Occurs when SQL Server raises an error.

Class

`TDAScript`

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

Occurs when SQL Server 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

17.7.1.2 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

```
TDAStatement = class(TCollectionItem);
```

Remarks

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

Inheritance Hierarchy

TObject
  TDAStatement

See Also

- TDAScript
- TDAStatements

TDAStatement class overview.

Properties

Name	Description
EndLine	Used to determine the number of the last statement line in a script.
EndOffset	Used to get the offset in the last line of the statement.
EndPos	Used to get the end position of the statement in a script.
Omit	Used to avoid execution of a statement.
Params	Contains parasmeters for an SQL statement.
Script	Used to determine the TDAScript object the SQL Statement belongs to.
SQL	Used to get or set the text of an SQL statement.
StartLine	Used to determine the number of the first statement line in a script.
StartOffset	Used to get the offset in the first line of a statement.
StartPos	Used to get the start position of the statement in a script.

Properties of the **TDAStatement** class.
For a complete list of the **TDAStatement** class members, see the TDAStatement Members topic.

Public

Name	Description
EndLine	Used to determine the number of the last statement line in a script.
EndOffset	Used to get the offset in the last line of the statement.
EndPos	Used to get the end position of the statement in a script.
Omit	Used to avoid execution of a statement.

Params	Contains parasmeters for an SQL statement.
Script	Used to determine the TDAScript object the SQL Statement belongs to.
SQL	Used to get or set the text of an SQL statement.
StartLine	Used to determine the number of the first statement line in a script.
StartOffset	Used to get the offset in the first line of a statement.
StartPos	Used to get the start position of the statement in a script.

See Also

- TDAStatement Class
- TDAStatement Class Members

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

Class

TDAStatement

Syntax

```
property EndLine: integer;
```

Remarks

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

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

Class

TDAStatement

Syntax

```
property EndOffset: integer;
```

Remarks

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

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

Class

TDAStatement

Syntax

```
property EndPos: integer;
```

Remarks

Use the EndPos property to get the end position of the statement (the position of the last character in the statement) in a script.

Used to avoid execution of a statement.

Class

TDAStatement

Syntax

```
property Omit: boolean;
```

Remarks

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

Contains parasmeters for an SQL statement.

Class

TDAStatement

Syntax

```
property Params: TDAParams;
```

Remarks

Contains parameters for an SQL statement.
Access Params at runtime to view and set parameter names, values, and data types dynamically.
Params is a ero-based array of parameter records. Index specifies the array element to access.

See Also

- TDAParam

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

Class

TDAStatement

Syntax

```
property Script: TDAScript;
```

Remarks

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

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

Class

TDAStatement

Syntax

```
property SQL: string;
```

Remarks

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

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

Class

TDAStatement

Syntax

```
property StartLine: integer;
```

Remarks

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

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

Class

TDAStatement

Syntax

```
property StartOffset: integer;
```

Remarks

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

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

Class

TDAStatement

Syntax

```
property StartPos: integer;
```

Remarks

Use the StartPos property to get the start position of the statement (the position of the first statement character) in a script.

17.7.1.3 DAScript.TDAStatements Class

Holds a collection of TDAStatement objects.
For a list of all members of this type, see TDAStatements members.

Unit

DAScript

Syntax

```
TDAStatements = class(TCollection);
```

Remarks

Each TDAStatements holds a collection of TDAStatement objects. TDAStatements maintains an index of the statements in its Items array. The Count property contains the number of statements in the collection. Use TDAStatements class to manipulate script SQL statements.

Inheritance Hierarchy

TObject
  TDAStatements

See Also

- TDAScript
- TDAStatement

TDAStatements class overview.

Properties

Name	Description
Items	Used to access separate script statements.

Properties of the **TDAStatements** class.
For a complete list of the **TDAStatements** class members, see the TDAStatements Members topic.

Public

Name	Description
Items	Used to access separate script statements.

See Also

- TDAStatements Class
- TDAStatements Class Members

Used to access separate script statements.

Class

`TDAStatements`

Syntax

```
property Items[Index: Integer]: TDAStatement; default;
```

Parameters

Index
Holds the index value.

Remarks

Use the Items property to access individual script statements. The value of the Index parameter corresponds to the Index property of TDAStatement.

See Also

- TDAStatement

17.7.2 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.

17.7.2.1 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.

17.7.2.2 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.

17.7.2.3 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.

17.7.3 Enumerations

Enumerations in the **DAScript** unit.

Enumerations

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

17.7.3.1 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.

17.8 DASQLMonitor

This unit contains the base class for the TMSSQLMonitor component.

Classes

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

Types

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

Enumerations

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

17.8.1 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.

17.8.1.1 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

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.

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

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

Used to set options for dbMonitor.

Class

`TCustomDASQLMonitor`

Syntax

```
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

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

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

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

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

17.8.1.2 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

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.

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

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.

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.

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.

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.

17.8.2 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.

17.8.2.1 DASQLMonitor.TDATraceFlags Set

Represents the set of TDATraceFlag.

Unit

DASQLMonitor

Syntax

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

17.8.2.2 DASQLMonitor.TMonitorOptions Set

Represents the set of TMonitorOption.

Unit

DASQLMonitor

Syntax

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

17.8.2.3 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.

17.8.3 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.

17.8.3.1 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.

17.8.3.2 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.

17.9 DBAccess

This unit contains base classes for most of the components.

Classes

Name	Description
EDAError	A base class for exceptions that are raised when an error occurs on the server side.
TCRDataSource	Provides an interface between a DAC dataset components and data-aware controls on a form.
TCustomConnectDialog	A base class for the connect dialog components.
TCustomDAConnection	A base class for components used to establish connections.
TCustomDADataSet	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
TCustomDASQL	A base class for components executing SQL statements that do not return result sets.
TCustomDAUpdateSQL	A base class for components that provide DML statements for more flexible control over data modifications.
TDAConnectionOptions	This class allows setting up the behaviour of the TDAConnection 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 formes rules for Data Type Mapping.
TDAMapRules	Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.
TDAMetaData	A class for retrieving metainformation of the specified database objects in the form of dataset.
TDAParam	A class that forms objects to represent the values of the parameters set.
TDAParams	This class is used to manage a list of TDAParam objects for an object that uses field parameters.
TDATransaction	A base class that implements functionality for controlling transactions.
TMacro	Object that represents the value of a macro.
TMacros	Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataSet components.

TPoolingOptions	This class allows setting up the behaviour of the connection pool.

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 languauge of labels in the connect dialog.
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.

17.9.1 Classes

Classes in the **DBAccess** unit.

Classes

Name	Description
EDAError	A base class for exceptions that are raised when an error occurs on the server side.
TCRDataSource	Provides an interface between a DAC dataset components and data-aware controls on a form.
TCustomConnectDialog	A base class for the connect dialog components.
TCustomDAConnection	A base class for components used to establish connections.
TCustomDADataSet	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
TCustomDASQL	A base class for components executing SQL statements that do not return result sets.
TCustomDAUpdateSQL	A base class for components that provide DML statements for more flexible control over data modifications.
TDAConnectionOptions	This class allows setting up the behaviour of the TDAConnection 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 formes rules for Data Type Mapping.
TDAMapRules	Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.
TDAMetaData	A class for retrieving metainformation of the specified database objects in the form of dataset.
TDAParam	A class that forms objects to represent the values of the parameters set.
TDAParams	This class is used to manage a list of TDAParam objects for an object that uses field parameters.
TDATransaction	A base class that implements functionality for controlling transactions.
TMacro	Object that represents the value of a macro.
TMacros	Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataSet components.
TPoolingOptions	This class allows setting up the behaviour of the connection pool.

17.9.1.1 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

EDAError class overview.

Properties

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

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

Contains the component that caused the error.

Class

EDAError

Syntax

```
property Component: TObject;
```

Remarks

The Component property contains the component that caused the error.

Determines the error code returned by the server.

Class

EDAError

Syntax

```
property ErrorCode: integer;
```

Remarks

Use the ErrorCode property to determine the error code returned by SQL Server. This value is always positive.
In SQL Server it's preferable to use EOLEDBError.OLEDBErrorCode and EMSError.MSSQLErrorCode instead of EDAError.ErrorCode.

17.9.1.2 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

TCRDataSource class overview.

17.9.1.3 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

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.

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

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.

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.

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.

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 SDAC installation directory.

See Also

- GetServerList

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.

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.

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.

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.

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.

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.

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.

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

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.

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

17.9.1.4 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

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.

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 customi ing 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

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

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 TEXT 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.

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.

See Also

- StartTransaction
- Commit
- Rollback

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 SdacVcl unit appears to the uses clause.

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

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.
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

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 , Server, Username, Password, TCustomMSConnection.Database, TCustomMSConnection.IsolationLevel, TMSConnection.Authentication, QuotedIdentifier, Provider, Language, Encrypt, PersistSecurityInfo, AutoTranslate, NetworkLibrary, ApplicationName, WorkstationID, PacketSize.
Note: Using Pooling := True can cause errors with working with temporary tables.

See Also

- Username
- Password
- PoolingOptions
- Connection Pooling

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

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. If this property is not set, SDAC tries to connect to '(local)'.
If this property is not set, SDAC tries to connect to '(local)'.

See Also

- Username
- Password

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, SDAC tries to connect with the sa user name.
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

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

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.

Applies changes from all active datasets.

Class

TCustomDAConnection

Syntax

```
procedure ApplyUpdates; overload; virtual
```

Remarks

Call the ApplyUpdates method to write all pending cached updates from all active datasets attached to this connection to a database or from specific datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions, and clearing the cache when the operation is successful.
Using ApplyUpdates for connection is a preferred method of updating datasets rather than calling each individual dataset's ApplyUpdates method.

See Also

- TMemDataSet.CachedUpdates
- TMemDataSet.ApplyUpdates

Applies changes from the specified datasets.

Class

TCustomDAConnection

Syntax

```
procedure ApplyUpdates(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.

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
- TCustomMSDataSet.FetchAll

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 LoginPrompt 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.
Note, if you would like to use SDAC in service, console or just at a separate thread, you need to call CoInitialize for each thread. Also remember to call CoUnInitialize at the end of a thread.

See Also

- Disconnect
- Username
- Password
- Server
- ConnectDialog

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.

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.

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 rolls back 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

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)
Label1.Caption:= MyMSConnection1.ExecProc('My_Sum', [10, 20]);
Label2.Caption:= MyMSConnection1.ParamByName('Result').AsString;
```

See Also

-

ExecProcEx
- ExecSQL
- ExecSQLEx

Allows to execute a stored procedure or function.

Class

TCustomDAConnection

Syntax

```
function ExecProcEx(Name: string; const Params: array of variant):
  variant; virtual;
```

Parameters

Name
Holds the stored procedure name.

Params
Holds an array of pairs of parameters' names and values.

Return Value

the result of the stored procedure.

Remarks

Allows to execute a stored procedure or function. Provide the stored procedure name and its parameters to the call of ExecProcEx.
Use the following Name value syntax for executing specific overloaded routine: "StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.
Assign pairs of parameters' names and values to a Params array so that every name comes before its corresponding value when an array is being indexed.
Out parameters of the procedure can be accessed with the ParamByName procedure. If the value for an input parameter was not included to the Params array, the parameter default value is taken. If the parameter has no default value, the NULL value is sent.
Note: Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. Stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.
For an example of parameters usage see ExecSQLEx.

Example

If you have some stored procedure accepting four parameters, and you want to provide values only for the first and fourth parameters, you should call ExecProcEx in the following way:

```
Connection.ExecProcEx('Some_Stored_Procedure', ['Param_Name1', 'Param_Valu
```

See Also

- ExecSQL
- ExecSQLEx
- ExecProc

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.

See Also

- ExecSQLEx
- ExecProc

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

```
MSConnection.ExecSQLEx('begin :A:= :B + :C; end;',
    ['A', 0, 'B', 5, 'C', 3]);
A:= MSConnection.ParamByName('A').AsInteger;
```

See Also

- ExecSQL

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

Returns a list of stored procedures from the server.

Class

`TCustomDAConnection`

Syntax

```
procedure GetStoredProcNames(List: _TStrings; AllProcs: boolean =
  False); virtual;
```

Parameters

List
A TStrings descendant that will be filled with the names of stored procedures in the database.

AllProcs
True, if stored procedures from all schemas or including system procudures (depending on the server) are returned. False otherwise.

Remarks

Call the GetStoredProcNames method to get the names of available stored procedures and functions. GetStoredProcNames populates a string list with the names of stored procs in the database. If AllProcs = True, the procedure returns to the List parameter the names of the stored procedures that belong to all schemas; otherwise, List will contain the names of functions that belong to the current schema.
Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetStoredProcNames.

See Also

- GetDatabaseNames
- GetTableNames
- GetTableNames

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

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

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

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
- TCustomMSDataSet.FetchAll

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.
In SQL Server real transaction begins only on the first execute of data modification SQL statement.
Note: In some cases TCustomMSDataSet.FetchAll may conflict with transaction control (EOLEDBError 'Cannot create new connection because in manual or distributed transaction mode.') or may cause deadlocking on Post on editing queries with ORDER BY clause.
Also no transactions can be started and there are underfetched datasets within the connection.

See Also

- Commit
- Rollback
- InTransaction
- TCustomMSConnection.IsolationLevel

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

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.

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.

17.9.1.5 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

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.

Properties of the **TCustomDADataSet** class.
For a complete list of the **TCustomDADataSet** class members, see the TCustomDADataSet Members topic.

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

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

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.

See Also

- TCustomMSConnection

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 SdacVcl 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 SdacVcl unit to your project.

See Also

- TCustomDASQL.Debug

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

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.

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.

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.

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

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

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.

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 uses the metainformation sent by the server together with data.

See Also

- SQLDelete
- SQLInsert
- SQLRefresh
- SQLUpdate

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

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

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

See Also

-
TMacro
- MacroByName
- Params

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

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

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

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

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

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 ero-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

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.

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

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.

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

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

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.
If you specify SQLInsert not depending on TCustomMSDataSet.Options.QueryIdentity, the value of the Identity filed won't be returned on execution Insert(Append).Post. To avoid the problem, you should add the following the code in the end of SQLInsert:

```
INSERT INTO Orders
      (Shipname)
   VALUES
      (:Shipname)
```

See Also

- SQL
- SQLUpdate
- SQLDelete
- SQLRefresh

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

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

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

Used if an application does not need bidirectional access to records in the result set.

Class

TCustomDADataSet

Syntax

```
property UniDirectional: boolean default False;
```

Remarks

Traditionally SQL cursors are unidirectional. They can travel only forward through a dataset. TCustomDADataset, however, permits bidirectional travelling by caching records. If an application does not need bidirectional access to the records in the result set, set UniDirectional to True. When UniDirectional is True, an application requires less memory and performance is improved. However, UniDirectional datasets cannot be modified.
In FetchAll=False mode data is fetched on demand. When UniDirectional is set to True, data is fetched on demand as well, but obtained rows are not cached except for the current row. So, FetchAll=False mode is a component of UniDirectional=True mode, and setting UniDirectional to True requires FetchAll to be set to False. Pay attention to the restrictions of TCustomMSDataSet.FetchAll = False mode.
The default value of UniDirectional is False, enabling forward and backward navigation.

See Also

- TCustomMSDataSet.FetchAll

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	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.
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	Locks the current record.
MacroByName	Finds a Macro with the name passed in Name.
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.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
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.
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

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

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.
It makes sense to call BreakExec only from another thread.
Also you should remeber that calling BreakExec to interrupt dataset opening in the NonBlocking mode may not have effect if fetch has already begun (this happens when BreakExec falls between two fetch

operations).

See Also

- Execute
- TMSConnection.OnInfoMessage
- TMSSQL.BreakExec

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.

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

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

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.

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

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

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

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.

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

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

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

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.

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.

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

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

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

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

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.
The Lock method sets the savepoint with the name LOCK + <component name>.

See Also

- UnLock

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

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

See Also

- TMacro
- Macros
- FindMacro

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

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.
SQL statements which have output parameters and aren't stored procedures calls or some of system functions such as sp_setapprole, should be executed without prior call to the Prepare method.
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

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

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

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.

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

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

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.

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

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**)	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.

See Also

- TCustomDADataSet Class
- TCustomDADataSet Class Members

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

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.
Note: In TCustomMSDataSet.Options mode this event occurs in context of calling thread.

See Also

- BeforeFetch
- TCustomMSDataSet.Options
- NonBlocking demo project

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.

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 TCustomMSDataSet.Options mode event handler is called from the fetching thread. Therefore, if you have set NonBlocking property to True, you should use thread synchroni ation mechanisms in the code of BeforeFetch event handler.

See Also

- AfterFetch
- TCustomMSDataSet.Options
- NonBlocking demo project

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

17.9.1.6 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

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.

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

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.

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.

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 SdacVcl 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 SdacVcl unit to your project.

See Also

- TCustomDADataSet.Debug

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.

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

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

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

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.

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 ero-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 MSSQL 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

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.

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

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.

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

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

Executes SQL commands.

Class

`TCustomDASQL`

Overload List

Name	Description
Execute	Executes SQL commands.
Execute(Iters: integer)	Is not used in SDAC.

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.

Is not used in SDAC.

Class

TCustomDASQL

Syntax

```
procedure Execute(Iters: integer); overload; virtual
```

Parameters

Iters
Is not used in SDAC.

Remarks

Is not used in SDAC.

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.

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

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

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

Parameters

Value
Holds the parameter name to search for.

Return Value

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

Remarks

Call the FindParam method to find a parameter with the specified name in a dataset.

See Also

- ParamByName

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

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

```
MSSQL.Execute;
Edit1.Text := MSSQL.ParamsByName('Contact').AsString;
```

See Also

- FindParam

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.
SQL statements which have output parameters and aren't stored procedures calls or some of system functions such as sp_setapprole, should be executed without prior call to the Prepare method.
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

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

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

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

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

17.9.1.7 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

- TCustomMSDataSet.UpdateObject

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.

Properties of the **TCustomDAUpdateSQL** class.
For a complete list of the **TCustomDAUpdateSQL** class members, see the TCustomDAUpdateSQL Members topic.

Public

Name	Description
DataSet	Used to hold a reference to the TCustomDADataSet object that is being updated.
SQL	Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Published

Name	Description
DeleteObject	Provides ability to perform advanced adjustment of the delete operations.
DeleteSQL	Used when deleting a record.
InsertObject	Provides ability to perform advanced adjustment of insert operations.
InsertSQL	Used when inserting a record.
LockObject	Provides ability to perform advanced adjustment of lock operations.

LockSQL	Used to lock the current record.
ModifyObject	Provides ability to perform advanced adjustment of modify operations.
ModifySQL	Used when updating a record.
RefreshObject	Provides ability to perform advanced adjustment of refresh operations.
RefreshSQL	Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.

See Also

- TCustomDAUpdateSQL Class
- TCustomDAUpdateSQL Class Members

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.

Provides ability to perform advanced adjustment of the delete operations.

Class

TCustomDAUpdateSQL

Syntax

```
property DeleteObject: TComponent;
```

Remarks

Assign SQL component or a TCustomMSDataSet 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

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.

Provides ability to perform advanced adjustment of insert operations.

Class

TCustomDAUpdateSQL

Syntax

```
property InsertObject: TComponent;
```

Remarks

Assign SQL component or TCustomMSDataSet 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

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.

Provides ability to perform advanced adjustment of lock operations.

Class

TCustomDAUpdateSQL

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomMSDataSet 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

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.

Provides ability to perform advanced adjustment of modify operations.

Class

TCustomDAUpdateSQL

Syntax

```
property ModifyObject: TComponent;
```

Remarks

Assign a SQL component or TCustomMSDataSet 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

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.

Provides ability to perform advanced adjustment of refresh operations.

Class

TCustomDAUpdateSQL

Syntax

```
property RefreshObject: TComponent;
```

Remarks

Assign a SQL component or TCustomMSDataSet 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

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

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.

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

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

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

17.9.1.8 DBAccess.TDAConnectionOptions Class

This class allows setting up the behaviour of the TDAConnection class.
For a list of all members of this type, see TDAConnectionOptions members.

Unit

DBAccess

Syntax

```
TDAConnectionOptions = class(TPersistent);
```

Inheritance Hierarchy

TObject
  TDAConnectionOptions

TDAConnectionOptions 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.

Properties of the **TDAConnectionOptions** class.
For a complete list of the **TDAConnectionOptions** class members, see the TDAConnectionOptions Members topic.

Public

Name	Description
DefaultSortType	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet. IndexFieldNames property of a dataset.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.

LocalFailover	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

See Also

- TDAConnectionOptions Class
- TDAConnectionOptions Class Members

Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

Class

TDAConnectionOptions

Syntax

```
property DefaultSortType: TSortType default stCaseSensitive;
```

Remarks

Use the DefaultSortType property to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

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

Class

TDAConnectionOptions

Syntax

```
property DisconnectedMode: boolean default False;
```

Remarks

If True, connection opens only when needed for performing a server call and closes after performing the operation. Datasets remain opened when connection closes. May be useful to save server resources and operate in unstable or expensive network. Drawback of using disconnect mode is that each connection establishing requires some time for authorization. If connection is often closed and opened it can slow down the application work. See the Disconnected Mode topic for more information.

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

Class

TDAConnectionOptions

Syntax

```
property KeepDesignConnected: boolean default True;
```

Remarks

At the time of startup prevents application from establishing a connection even if the Connected property was set to True at design-time. Set KeepDesignConnected to False to initialize the connected property to False, even if it was True at design-time.

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.

17.9.1.9 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

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 TCustomMSDataSet.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.

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.

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.

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.

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.

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.

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.

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.

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.

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 UpdateBatchSi e 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.

17.9.1.10 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

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.

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

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.

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.

17.9.1.11 DBAccess.TDAMapRule Class

Class that formes rules for Data Type Mapping.
For a list of all members of this type, see TDAMapRule members.

Unit

`DBAccess`

Syntax

```
TDAMapRule = class(TMapRule);
```

Remarks

Using properties of this class, it is possible to change parameter values of the specified rules from the TDAMapRules set.

Inheritance Hierarchy

TObject
  TMapRule
    TDAMapRule

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.

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

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.

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.

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.

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.

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.

The resultant field length in Delphi.

Class

TDAMapRule

Syntax

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

Remarks

Setting the Delphi field length after conversion.

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.

The resultant field Scale in Delphi.

Class

TDAMapRule

Syntax

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

Remarks

Setting the Delphi field Scale after conversion.

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.

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.

17.9.1.12 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

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.

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

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; FieldType: TFieldType; 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.

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.

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.

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.

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.

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.

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.

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.

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

---

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.

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.

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.

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 rlAny constant for them instead of a specific value. If the rlAny 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 rlAny.
If it is necessary to set scale to 5 or less, then DBScaleMin should be set to rlAny, and DBScaleMax - to 5.

17.9.1.13 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

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.

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

Used to specify a connection object to use to connect to a data store.

Class

`TDAMetaData`

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object to use to connect to a data store.
Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.
At runtime, set the Connection property to reference an instanciated TCustomDAConnection object.

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

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

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**)	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.
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.

See Also

- TDAMetaData Class
- TDAMetaData Class Members

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

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

17.9.1.14 DBAccess.TDAParam Class

A class that forms objects to represent the values of the parameters set.
For a list of all members of this type, see TDAParam members.

Unit

`DBAccess`

Syntax

```
TDAParam = class(TParam);
```

Remarks

Use the properties of TDAParam to set the value of a parameter. Objects that use parameters create TDAParam objects to represent these parameters. For example, TDAParam objects are used by TCustomDASQL, TCustomDADataSet.
TDAParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding and the way the field is displayed, edited, or calculated, that are not needed in a TDAParam object. Conversely, TDAParam includes properties that indicate how the field value is passed as a parameter.

Inheritance Hierarchy

TObject
  TDAParam

See Also

- TCustomDADataSet
- TCustomDASQL
- TDAParams

TDAParam class overview.

Properties

Name	Description
AsBlob	Used to set and read the value of the BLOB parameter as string.
AsBlobRef	Used to set and read the value of the BLOB parameter as a TBlob object.
AsFloat	Used to assign the value for a float field to a parameter.
AsInteger	Used to assign the value for an integer field to the parameter.
AsLargeInt	Used to assign the value for a LargeInteger field to the parameter.
AsMemo	Used to assign the value for a memo field to the parameter.
AsMemoRef	Used to set and read the value of the memo parameter as a TBlob object.
AsSQLTimeStamp	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString	Used to assign the string value to the parameter.
AsWideString	Used to assign the Unicode string value to the parameter.
DataType	Indicates the data type of the parameter.
IsNull	Used to indicate whether the value assigned to a parameter is NULL.
ParamType	Used to indicate the type of use for a parameter.
Size	Specifies the size of a string type parameter.

Value	Used to represent the value of the parameter as Variant.

Methods

Name	Description
AssignField	Assigns field name and field value to a param.
AssignFieldValue	Assigns the specified field properties and value to a parameter.
LoadFromFile	Places the content of a specified file into a TDAParam object.
LoadFromStream	Places the content from a stream into a TDAParam object.
SetBlobData	Overloaded. Writes the data from a specified buffer to BLOB.

Properties of the **TDAParam** class.
For a complete list of the **TDAParam** class members, see the TDAParam Members topic.

Public

Name	Description
AsBlob	Used to set and read the value of the BLOB parameter as string.
AsBlobRef	Used to set and read the value of the BLOB parameter as a TBlob object.
AsFloat	Used to assign the value for a float field to a parameter.
AsInteger	Used to assign the value for an integer field to the parameter.
AsLargeInt	Used to assign the value for a LargeInteger field to the parameter.
AsMemo	Used to assign the value for a memo field to the parameter.
AsMemoRef	Used to set and read the value of the memo parameter as a TBlob object.
AsSQLTimeStamp	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString	Used to assign the string value to the parameter.
AsWideString	Used to assign the Unicode string value to the parameter.
IsNull	Used to indicate whether the value assigned to a parameter is NULL.

Published

Name	Description
DataType	Indicates the data type of the parameter.
ParamType	Used to indicate the type of use for a parameter.
Size	Specifies the size of a string type parameter.

Value	Used to represent the value of the parameter as Variant.

See Also

- TDAParam Class
- TDAParam Class Members

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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

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

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.

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.

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.

17.9.1.15 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

TDAParams class overview.

Properties

Name	Description
Items	Used to interate through all parameters.

Methods

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

Properties of the **TDAParams** class.
For a complete list of the **TDAParams** class members, see the TDAParams Members topic.

Public

Name	Description
Items	Used to interate through all parameters.

See Also

- TDAParams Class
- TDAParams Class Members

Used to interate through all parameters.

Class

`TDAParams`

Syntax

```
property Items[Index: integer]: TDAParam; default;
```

Parameters

Index
Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all parameters. Index identifies the index in the range 0.. Count - 1. Items can reference a particular parameter by its index, but the ParamByName method is preferred in order to avoid depending on the order of the parameters.

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

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.

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.

17.9.1.16 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

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.

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

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.

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.

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

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

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 across all database connections, and finish the transaction.

See Also

- Commit
- StartTransaction

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

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

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 control statements such as Commit, Rollback. Check the E parameter to get the error code.

See Also

- Commit
- Rollback
- StartTransaction

17.9.1.17 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

TMacro class overview.

Properties

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

AsDateTime	Used to set the TDataTime value to a macro.
AsFloat	Used to set the float value to a macro.
AsInteger	Used to set the integer value to a macro.
AsString	Used to assign the string value to a macro.
Name	Used to identify a particular macro.
Value	Used to set the value to a macro.

Properties of the **TMacro** class.
For a complete list of the **TMacro** class members, see the TMacro Members topic.

Public

Name	Description
AsDateTime	Used to set the TDataTime value to a macro.
AsFloat	Used to set the float value to a macro.
AsInteger	Used to set the integer value to a macro.
AsString	Used to assign the string value to a macro.

Published

Name	Description
Active	Used to determine if the macro should be expanded.
Name	Used to identify a particular macro.
Value	Used to set the value to a macro.

See Also

- TMacro Class
- TMacro Class Members

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

```
MSQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';
MSQuery.Macros[0].Value := 'and DName is NULL';
MSQuery.Macros[0].Active:= False;
```

Used to set the TDataTime value to a macro.

Class

TMacro

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

Use the AsDataTime property to set the TDataTime value to a macro.

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.

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.

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.

Used to identify a particular macro.

Class

TMacro

Syntax

```
property Name: string;
```

Remarks

Use the Name property to identify a particular macro.

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.

17.9.1.18 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

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.

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

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.

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

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.

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.

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.

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.

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.

17.9.1.19 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

TPoolingOptions class overview.

Properties

Name	Description
ConnectionLifetime	Used to specify the maximum time during which an opened connection can be used by connection pool.
MaxPoolSi e	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSi e	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.

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

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.

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.

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.

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.

17.9.2 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.

17.9.2.1 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.

17.9.2.2 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.

17.9.2.3 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.

17.9.2.4 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.

17.9.2.5 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 .

17.9.2.6 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.

17.9.2.7 DBAccess.TRefreshOptions Set

Represents the set of TRefreshOption.

Unit

DBAccess

Syntax

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

17.9.2.8 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.

17.9.3 Enumerations

Enumerations in the **DBAccess** unit.

Enumerations

Name	Description
TLabelSet	Sets the languauge of labels in the connect dialog.
TRefreshOption	Indicates when the editing record will be refreshed.
TRetryMode	Specifies the application behavior when connection is lost.

17.9.3.1 DBAccess.TLabelSet Enumeration

Sets the languauge of labels in the connect dialog.

Unit

DBAccess

Syntax

```
TLabelSet = (lsCustom, lsEnglish, lsFrench, lsGerman, lsItalian,
 lsPolish, lsPortuguese, lsRussian, lsSpanish);
```

Values

Value	Meaning
lsCustom	Set the language of labels in the connect dialog manually.
lsEnglish	Set English as the language of labels in the connect dialog.
lsFrench	Set French as the language of labels in the connect dialog.
lsGerman	Set German as the language of labels in the connect dialog.
lsItalian	Set Italian as the language of labels in the connect dialog.
lsPolish	Set Polish as the language of labels in the connect dialog.
lsPortuguese	Set Portuguese as the language of labels in the connect dialog.
lsRussian	Set Russian as the language of labels in the connect dialog.
lsSpanish	Set Spanish as the language of labels in the connect dialog.

17.9.3.2 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.

17.9.3.3 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.

17.9.4 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 TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.

17.9.4.1 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 SDAC 3.55.2.22. To restore old behavior, set the BaseSQLOldBehavior variable to True.

17.9.4.2 DBAccess.ChangeCursor Variable

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

Unit

DBAccess

Syntax

```
ChangeCursor: boolean;
```

17.9.4.3 DBAccess.MacroChar Variable

Determinates what character is used for macros.

Unit

DBAccess

Syntax

```
MacroChar: _char;
```

17.9.4.4 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, SDAC 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 SDAC 4.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

17.10 Devart.Dac.DataAdapter

This unit contains implementation of the DADataAdapter class.

Classes

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

17.10.1 Classes

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

Classes

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

17.10.1.1 Devart.Dac.DataAdapter.DADataAdapter 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 DADataAdapter members.

Unit

`Devart.Dac.DataAdapter`

Syntax

```
DADataAdapter = 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 DADataAdapter.Fill, which changes the data in the System.Data.DataSet to match the data in the data source, and DADataAdapter.Update, which changes the data in the data source to match the data in the System. Data.DataSet.

Inheritance Hierarchy

TObject
  DADataAdapter

DADataAdapter class overview.

Properties

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

Methods

Name	Description
Fill	Adds or refreshes rows in System. Data.DataSet to match those in TDataSet and creates 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 DataTable.

Properties of the **DADataAdapter** class.
For a complete list of the **DADataAdapter** class members, see the DADataAdapter Members topic.

Public

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

See Also

- DADataAdapter Class
- DADataAdapter Class Members

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

Class

DADataAdapter

Syntax

```
property DataSet: TDataSet;
```

Remarks

Specify a TDataSet object which will be used as a data source for the DADataAdapter component.

Methods of the **DADataAdapter** class.
For a complete list of the **DADataAdapter** class members, see the DADataAdapter Members topic.

Public

Name	Description
Fill	Adds or refreshes rows in System.Data.DataSet to match those in TDataSet and creates 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 DataTable.

See Also

- DADataAdapter Class
- DADataAdapter Class Members

Adds or refreshes rows in System.Data.DataSet to match those in TDataSet and creates DataTable.

Class

DADataAdapter

Syntax

```
function Fill(Data: DataSet; tableName: string): integer;
```

Parameters

Data
holds the dataset updates of which are to be commented to database.

tableName

holds the name of DataTable.

Return Value

the number of rows successfully inserted into DataSet.

Remarks

Adds or refreshes rows in System.Data.DataSet to match those in TDataSet using the DataSet parameter, and creates DataTable named tableName. Function returns the number of rows successfully inserted into DataSet.
The TDataSet object associated with DADataAdapter 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 a 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 DataSet, and no exception is raised.

See Also

- 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 DataTable.

Class

DADataAdapter

Syntax

```
function Update(Data: DataSet; tableName: string): integer;
```

Parameters

Data
holds the dataset updates of which are to be commented to database.

tableName
holds the name of DataTable.

Return Value

the number of rows successfully updated from 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 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 DataSet.

See Also

- Fill

17.11 Devart.Sdac.DataAdapter

This unit contains implementation of the MSDataAdapter class.

Classes

Name	Description
MSDataAdapter	A class for using with TCustomMSDataSet components and as data source for retrieving and saving data.

17.11.1 Classes

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

Classes

Name	Description
MSDataAdapter	A class for using with TCustomMSDataSet components and as data source for retrieving and saving data.

17.11.1.1 Devart.Sdac.DataAdapter.MSDataAdapter Class

A class for using with TCustomMSDataSet components and as data source for retrieving and saving data.
For a list of all members of this type, see MSDataAdapter members.

Unit

`Devart.Sdac.DataAdapter`

Syntax

```
MSDataAdapter = class(DADataAdapter);
```

Remarks

The MSDataAdapter class is designed for using with TCustomMSDataSet components and as data source for retrieving and saving data. MSDataAdapter provides this bridge by mapping DADataAdapter.Fill, which changes the data in the System.Data.DataSet to match the data in the data source, and DADataAdapter.Update, which changes the data in the data source to match the data in the System.Data.DataSet.

Inheritance Hierarchy

TObject
  DADataAdapter
    MSDataAdapter

See Also

- DADataAdapter

MSDataAdapter class overview.

Properties

Name	Description
DataSet (inherited from **DADataAdapter**)	Used to specify a TDataSet object which will be used as a data source for the DADataAdapter component.

Methods

Name	Description
Fill (inherited from **DADataAdapter**)	Adds or refreshes rows in System.Data.DataSet to match those in TDataSet and creates DataTable.
Update (inherited from **DADataAdapter**)	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 DataTable.

17.12 MemData

This unit contains classes for storing data in memory.

Classes

Name	Description
TAttribute	TAttribute is not used in SDAC.
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	This class is not used.
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.

17.12.1 Classes

Classes in the **MemData** unit.

Classes

Name	Description
TAttribute	TAttribute is not used in SDAC.
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	This class is not used.
TSharedObject	A base class that allows to simplify memory management for object referenced by several other objects.

17.12.1.1 MemData.TAttribute Class

TAttribute is not used in SDAC.
For a list of all members of this type, see TAttribute members.

Unit

MemData

Syntax

```
TAttribute = class(System.TObject);
```

Inheritance Hierarchy

TObject
  TAttribute

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.

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

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

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.

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.

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

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.

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.

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.

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

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

17.12.1.2 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

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.

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

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

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

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.

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.

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

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

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.

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

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

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

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

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

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.

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

17.12.1.3 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: I
   TUncompressProc = function(dest: IntPtr; destlen: IntPtr; source: IntPt
var
   CompressProc: TCompressProc;
   UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

TObject
  TSharedObject
    TBlob
      TCompressedBlob

See Also

- TBlob
- TMemDataSet.GetBlob
- TCustomDADataSet.Options

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.

17.12.1.4 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

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.

17.12.1.5 MemData.TObjectType Class

This class is not used.
For a list of all members of this type, see TObjectType members.

Unit

MemData

Syntax

```
TObjectType = class(TSharedObject);
```

Inheritance Hierarchy

TObject
  TSharedObject
    TObjectType

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.

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

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.

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

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

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

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

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 TAttibute 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 TAttibute object for the specified attribute. If the attribute can not be found, an exception is raised.

See Also

- TAttribute
- FindAttribute
- Attributes

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

17.12.1.6 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

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.

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

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.

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

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

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

17.12.2 Types

Types in the **MemData** unit.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption.
TUpdateRecKinds	Represents the set of TUpdateRecKind.

17.12.2.1 MemData.TLocateExOptions Set

Represents the set of TLocateExOption.

Unit

MemData

Syntax

```
TLocateExOptions = set of TLocateExOption;
```

17.12.2.2 MemData.TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

MemData

Syntax

```
TUpdateRecKinds = set of TUpdateRecKind;
```

17.12.3 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.

17.12.3.1 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.

17.12.3.2 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.

17.12.3.3 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.

17.12.3.4 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.

17.12.3.5 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.

17.13 MemDS

This unit contains implementation of the TMemDataSet class.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

Variables

Name	Description
DoNotRaiseExcetionOnUaFail	An exception will be raised if the value of the UpdateAction parameter is uaFail.
SendDataSetChangeEventAfterOpen	The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

17.13.1 Classes

Classes in the **MemDS** unit.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

17.13.1.1 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

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	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.

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.

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

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

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';
```

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

Class

TMemDataSet

Syntax

```
property LocalConstraints: boolean default True;
```

Remarks

Use the LocalConstraints property to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet. When LocalConstrains is True, TMemDataSet ignores NOT NULL server constraints. It is useful for tables that have fields updated by triggers.
LocalConstraints is obsolete, and is only included for backward compatibility.
The default value is True.

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.

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

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

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

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

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.

Writes dataset's pending cached updates to a database.

Class

TMemDataSet

Syntax

```
procedure ApplyUpdates; overload; virtual
```

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.
Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.
Note: The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

Example

The following procedure illustrates how to apply a dataset's cached updates to a database in response to a button click:

```
procedure ApplyButtonClick(Sender: TObject);
begin
   with MyQuery do
   begin
      Session.StartTransaction;
      try
         ... {Modify data}
         ApplyUpdates; {try to write the updates to the database}
         Session.Commit; {on success, commit the changes}
      except
         RestoreUpdates; {restore update result for applied records}
         Session.Rollback; {on failure, undo the changes}
         raise; {raise the exception to prevent a call to CommitUpdates!}
      end;
      CommitUpdates; {on success, clear the cache}
   end;
end;
```

See Also

- TMemDataSet.CachedUpdates
- TMemDataSet.CancelUpdates
- TMemDataSet.CommitUpdates
- TMemDataSet.UpdateStatus

Writes dataset's pending cached updates of specified records to a database.

Class

`TMemDataSet`

Syntax

```
procedure ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds);
 overload; virtual
```

Parameters

UpdateRecKinds
Indicates records for which the ApplyUpdates method will be performed.

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates of specified records to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.
Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.
Note: The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

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

Clears the cached updates buffer.

Class

TMemDataSet

Syntax

```
procedure CommitUpdates;
```

Remarks

Call the CommitUpdates method to clear the cached updates buffer after both a successful call to ApplyUpdates and a database component's Commit method. Clearing the cache after applying updates ensures that the cache is empty except for records that could not be processed and were skipped by the OnUpdateRecord or OnUpdateError event handlers. An application can attempt to modify the records still in cache.
CommitUpdates also checks wether there are pending updates in dataset. And if there are, it calls ApplyUpdates.
Record modifications made after a call to CommitUpdates repopulate the cached update buffer and require a subsequent call to ApplyUpdates to move them to the database.

See Also

- CachedUpdates
- TMemDataSet.ApplyUpdates
- UpdateStatus

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.

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.

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.

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

```
MSQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

See Also

- TBlob

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.

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.

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

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.

Class

TMemDataSet

Overload List

Name	Description
LocateEx(**const** KeyFields: **array of** TField; **const** KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.
LocateEx(**const** KeyFields: string; **const** KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.

Class

TMemDataSet

Syntax

```
function LocateEx(const KeyFields: array of TField; const
  KeyValues: variant; Options: TLocateExOptions): boolean;
  overload
```

Parameters

KeyFields
Holds TField objects to search in.

KeyValues
Holds the values of the fields to search for.

Options
Holds additional search parameters which will be used by the LocateEx method.

Return Value

True, if a matching record was found. Otherwise returns False.

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.

Class

TMemDataSet

Syntax

```
function LocateEx(const KeyFields: string; const KeyValues:
  variant; Options: TLocateExOptions): boolean; overload
```

Parameters

KeyFields
Holds the fields to search in.

KeyValues
Holds the values of the fields to search for.

Options
Holds additional search parameters which will be used by the LocateEx method.

Return Value

True, if a matching record was found. Otherwise returns False.

Remarks

Call the LocateEx method when you need some features not to be included to the TMemDataSet.Locate method of TDataSet.
LocateEx returns True if it finds a matching record, and makes that record the current one. Otherwise LocateEx returns False.
The LocateEx function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the TMemDataSet.IndexFieldNames property.
Note: Please add the MemData unit to the "uses" list to use the TLocalExOption enumeration.

See Also

- TMemDataSet.IndexFieldNames
- TMemDataSet.Locate

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. 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.

See Also

- Prepared
- UnPrepare

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

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

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.

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

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.

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

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

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

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

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

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

17.13.2 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.

17.13.2.1 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 SDAC 4.20.0.13, 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.

17.13.2.2 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 SDAC 4.20.0.12, 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.

17.14 MSAccess

This unit contains implementation of most public classes of SDAC.

Classes

Name	Description
TCustomMSConnection	A base class defining functionality for classes derived from it, and introducing OLE DB specific properties.
TCustomMSConnectionOptions	This class allows setting up the behaviour of the TCustomMSConnection class.
TCustomMSDataSet	A component for defining the functionality for the classes derived from it.
TCustomMSStoredProc	A component used to access stored procedures on a database server.
TCustomMSTable	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
TMSChangeNotification	A component for keeping information in local dataset up-to-date through receiving notifications.
TMSConnection	A component for establishing connection to the database server, providing customized login support and performing transaction control.
TMSConnectionOptions	This class allows setting up the behaviour of the TMSConnection class.
TMSDataSetOptions	This class allows setting up the behaviour of the TMSDataSet class.
TMSDataSource	TMSDataSource provides an interface between a SDAC dataset components and data-aware controls on a form.
TMSEncryptor	The class that performs encrypting and decrypting of data.
TMSFileStream	A class for managing FILESTREAM data using Win32 API.
TMSMetadata	A component for obtaining metainformation about database objects from the server.
TMSParam	A class that is used to set the values of individual parameters passed with queries or stored procedures.
TMSParams	Used to control TMSParam objects.
TMSQuery	A component for executing queries and operating record sets. It also provides flexible way to update data.
TMSSQL	A component for executing SQL statements and calling stored procedures on the database server.

TMSStoredProc	A component for accessing and executing stored procedures and functions.
TMSTable	A component for retrieving and updating data in a single table without writing SQL statements.
TMSTableData	A component for working with user-defined table types in SQL Server 2008.
TMSUDTField	A field class providing native access to the CLR User-defined Types (UDT) fields of SQL Server 2005.
TMSUpdateSQL	A component for tuning update operations for the DataSet component.
TMSXMLField	A class providing access to the SQL Server 2005 xml data type.

Types

Name	Description
TMSChangeNotificationEvent	This type is used for the TMSChangeNotification.OnChange event.
TMSUpdateExecuteEvent	Description of TMSUpdateExecuteEvent is not available at the moment

Enumerations

Name	Description
TIsolationLevel	Specifies the extent to which all outside transactions interfere with the subsequent transactions of the current connection.
TMSLockType	Specifies the parameters for locking the current record.
TMSNotificationInfo	Indicates the reason of the notification.
TMSNotificationSource	Indicates the source of notification.
TMSNotificationType	Indicates if this notification is generated because of change or by subscription.
TMSObjectType	Enumerates the object types supported by TMSMetadata.

Variables

Name	Description
__UseUpdateOptimization	In SDAC 4.00.0.4 update statements execution was optimized. This optimization changed the behaviour of affected rows count retrieval for the tables with triggers.

Constants

Name	Description
SdacVersion	Read this constant to get the current version number for SDAC.

17.14.1 Classes

Classes in the **MSAccess** unit.

Classes

Name	Description
TCustomMSConnection	A base class defining functionality for classes derived from it, and introducing OLE DB specific properties.
TCustomMSConnectionOptions	This class allows setting up the behaviour of the TCustomMSConnection class.
TCustomMSDataSet	A component for defining the functionality for the classes derived from it.
TCustomMSStoredProc	A component used to access stored procedures on a database server.
TCustomMSTable	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
TMSChangeNotification	A component for keeping information in local dataset up-to-date through receiving notifications.
TMSConnection	A component for establishing connection to the database server, providing customized login support and performing transaction control.
TMSConnectionOptions	This class allows setting up the behaviour of the TMSConnection class.
TMSDataSetOptions	This class allows setting up the behaviour of the TMSDataSet class.
TMSDataSource	TMSDataSource provides an interface between a SDAC dataset components and data-aware controls on a form.
TMSEncryptor	The class that performs encrypting and decrypting of data.
TMSFileStream	A class for managing FILESTREAM data using Win32 API.
TMSMetadata	A component for obtaining metainformation about database objects from the server.
TMSParam	A class that is used to set the values of individual parameters passed with queries or stored procedures.
TMSParams	Used to control TMSParam objects.
TMSQuery	A component for executing queries and operating record sets. It also provides flexible way to update data.
TMSSQL	A component for executing SQL statements and calling stored procedures on the database server.
TMSStoredProc	A component for accessing and executing stored procedures and functions.

TMSTable	A component for retrieving and updating data in a single table without writing SQL statements.
TMSTableData	A component for working with user-defined table types in SQL Server 2008.
TMSUDTField	A field class providing native access to the CLR User-defined Types (UDT) fields of SQL Server 2005.
TMSUpdateSQL	A component for tuning update operations for the DataSet component.
TMSXMLField	A class providing access to the SQL Server 2005 xml data type.

17.14.1.1 MSAccess.TCustomMSConnection Class

A base class defining functionality for classes derived from it, and introducing OLE DB specific properties.
For a list of all members of this type, see TCustomMSConnection members.

Unit

MSAccess

Syntax

```
TCustomMSConnection = class(TCustomDAConnection);
```

Remarks

TCustomMSConnection is a base connection class that defines functionality for classes derived from it, and introduces OLE DB specific properties. Applications should never use TCustomMSConnection objects directly. Descendants of TCustomMSConnection, such as TMSConnection, TMSCompactConnection should be used instead.

Inheritance Hierarchy

TObject
- TCustomDAConnection
  - **TCustomMSConnection**

See Also

- TMSConnection
- TMSCompactConnection

TCustomMSConnection class overview.

Properties

Name	Description
ClientVersion	Contains the version of Microsoft OLE DB Provider for SQL Server.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectString	Used to specify the connection information for the data store.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.

Database	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
IsolationLevel	Used to specify the extent to which all outside transactions interfere with subsequent transactions of the current connection.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Used to specify the behaviour of a TCustomMSConnection object.
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.
Server (inherited from **TCustomDAConnection**)	Serves to supply the server name for login.
ServerVersion	Contains the exact number of the SQL Server version.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

Methods

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect	Shares database connection between the TCustomMSConnection components.
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	Returns a new instance of the TMSSQL class and associates it with this connection object.
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.

MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
OpenDatasets	Opens several datasets as one batch.
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.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.

Events

Name	Description
OnConnectionLost (inherited from **TCustomDAConnection**)	This event occurs when connection was lost.
OnError (inherited from **TCustomDAConnection**)	This event occurs when an error has arisen in the connection.

Properties of the **TCustomMSConnection** class.
For a complete list of the **TCustomMSConnection** class members, see the TCustomMSConnection Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
ClientVersion	Contains the version of Microsoft OLE DB Provider for SQL Server.
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.
ConnectString	Used to specify the connection information for the data store.
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 specify the database name that is a default source of data for SQL queries once a connection is established.
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.
IsolationLevel	Used to specify the extent to which all outside transactions interfere with subsequent transactions 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.
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	Used to specify the behaviour of a TCustomMSConnection object.
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.
ServerVersion	Contains the exact number of the SQL Server version.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

See Also

- TCustomMSConnection Class
- TCustomMSConnection Class Members

Contains the version of Microsoft OLE DB Provider for SQL Server.

Class

TCustomMSConnection

Syntax

```
property ClientVersion: string;
```

Remarks

The version of Microsoft OLE DB Provider for SQL Server (sqloledb.dll).
To get the value of this property, connection to the server must be established.

See Also

- TCustomDAConnection.Connect
- ServerVersion

Used to specify the connection information for the data store.

Class

`TCustomMSConnection`

Syntax

```
property ConnectString: string;
```

Remarks

Specifies the connection information for the data store.
SDAC recognizes an ODBC-like syntax in provider string property values. Within the string, elements are delimited by using a semicolon. Each element consists of a keyword, an equal sign character, and the value passed on initialization.

Example

For example:

```
Server=London1;User ID=nancyd
```

See Also

- TCustomDAConnection.Server
- TCustomDAConnection.Username
- TCustomDAConnection.Password

Used to specify the database name that is a default source of data for SQL queries once a connection is established.

Class

`TCustomMSConnection`

Syntax

```
property Database: string;
```

Remarks

Use the Database property to specify the database name that is a default source of data for SQL queries once a connection is established.
Altering the Database property makes new database name take effect immediately.
When Database is not assigned, SDAC 4.20 and higher will use the default database for the current SQL Server login specified in the TCustomDAConnection.Username property. Preceding SDAC versions use the 'master' database by default.
Setting Database='Northwind' allows you to omit database specifier in the SELECT statements. That is, instead of

```
SELECT * FROM Northwind..Products;
```

you may just write

```
SELECT * FROM Products
```

See Also

- TCustomDAConnection.Server
- TCustomDAConnection.Username
- TCustomDAConnection.Password

Used to specify the extent to which all outside transactions interfere with subsequent transactions of the current connection.

Class

`TCustomMSConnection`

Syntax

```
property IsolationLevel: TIsolationLevel default ilReadCommitted;
```

Remarks

Use the IsolationLevel property to specify the extent to which all outside transactions interfere with subsequent transactions of the current connection.
Changes to IsolationLevel take effect at a time of starting new transaction or opening new connection.

Used to specify the behaviour of a TCustomMSConnection object.

Class

`TCustomMSConnection`

Syntax

```
property Options: TCustomMSConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TCustomMSConnection object.
Descriptions of all options are in the table below.

Option Name	Description
Encrypt	Specifies if data should be encrypted before sending it over the network.
NumericType	Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.
Provider	Used to specify a provider from the list of supported providers.
QuotedIdentifier	Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.
UseWideMemos	Used to manage field type creation for the NTEXT data type.

Contains the exact number of the SQL Server version.

Class

`TCustomMSConnection`

Syntax

```
property ServerVersion: string;
```

Remarks

The version of SQL Server.
To get the value of this property, connection to the server must be established.

See Also

- TCustomDAConnection.Connect
- ClientVersion

Methods of the **TCustomMSConnection** class.
For a complete list of the **TCustomMSConnection** class members, see the TCustomMSConnection Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect	Shares database connection between the TCustomMSConnection components.
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	Returns a new instance of the TMSSQL class and associates it with this connection object.
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.
OpenDatasets	Opens several datasets as one batch.
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.

See Also

- TCustomMSConnection Class
- TCustomMSConnection Class Members

Shares database connection between the TCustomMSConnection components.

Class

`TCustomMSConnection`

Syntax

```
procedure AssignConnect(Source: TCustomMSConnection);
```

Parameters

Source
Preconnected TCustomMSConnection component which connection is to be shared with the current TCustomMSConnection component.

Remarks

Use the AssignConnect method to share database connection between the TCustomMSConnection components.
AssignConnect assumes that the Source parameter points to a preconnected TCustomMSConnection component which connection is to be shared with the current TCustomMSConnection component. Note that AssignConnect doesn't make any references to the Source TCustomMSConnection component. So before disconnecting parent TCustomMSConnection component call AssignConnect(Nil) or the Disconnect method for all assigned connections.

Returns a new instance of the TMSSQL class and associates it with this connection object.

Class

`TCustomMSConnection`

Syntax

```
function CreateSQL: TCustomDASQL; override;
```

Return Value

a new instance of the TMSSQL class.

Remarks

CreateSQL returns a new instance of the TMSSQL class and associates it with this connection object.

See Also

- TCustomDAConnection.CreateDataSet

Opens several datasets as one batch.

Class

`TCustomMSConnection`

Syntax

```
procedure OpenDatasets(const ds: array of TCustomMSDataSet);
```

Parameters

ds
an array of datasets that will be opened.

Remarks

Call the OpenDatasets method to open several datasets as one batch. This method can significantly increase performance when opening queries through remote connection (e. g. Internet).
When you execute a query through remote connection, a delay occurs. If you open more than one query, the time of the delay increases proportionally to the number of opened queries. The OpenDatasets method puts all SQL queries from the received datasets together and executes them as one package. The received results are redistributed to the original dataset.
Note, that when this operation is performed, each one of the opened datasets should return only one resultset.

17.14.1.2 MSAccess.TCustomMSConnectionOptions Class

This class allows setting up the behaviour of the TCustomMSConnection class.
For a list of all members of this type, see TCustomMSConnectionOptions members.

Unit

`MSAccess`

Syntax

```
TCustomMSConnectionOptions = class(TDAConnectionOptions);
```

Inheritance Hierarchy

TObject
  TDAConnectionOptions
    TCustomMSConnectionOptions

TCustomMSConnectionOptions class overview.

Properties

Name	Description
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet. IndexFieldNames property of a dataset.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
Encrypt	Specifies if data should be encrypted before sending it over the network.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection. OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
NumericType	Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.
Provider	Used to specify a provider from the list of supported providers.
QuotedIdentifier	Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.
UseWideMemos	Used to manage field type creation for the NTEXT data type.

Properties of the **TCustomMSConnectionOptions** class.
For a complete list of the **TCustomMSConnectionOptions** class members, see the TCustomMSConnectionOptions Members topic.

Public

Name	Description
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet. IndexFieldNames property of a dataset.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
Encrypt	Specifies if data should be encrypted before sending it over the network.

KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
NumericType	Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.
Provider	Used to specify a provider from the list of supported providers.
QuotedIdentifier	Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.
UseWideMemos	Used to manage field type creation for the NTEXT data type.

See Also

- TCustomMSConnectionOptions Class
- TCustomMSConnectionOptions Class Members

Specifies if data should be encrypted before sending it over the network.

Class

`TCustomMSConnectionOptions`

Syntax

```
property Encrypt: boolean default False;
```

Remarks

Use the Encrypt property to specify if data should be encrypted before sending it over the network. The default value is False.

Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.

Class

`TCustomMSConnectionOptions`

Syntax

```
property NumericType: TDANumericType default ntFloat;
```

Remarks

Use the NumericType property to specify the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.

Used to specify a provider from the list of supported providers.

Class

`TCustomMSConnectionOptions`

Syntax

```
property Provider: TOLEDBProvider default prAuto;
```

Remarks

Use the Provider property to specify a provider from the list of supported providers.

Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.

Class

TCustomMSConnectionOptions

Syntax

```
property QuotedIdentifier: boolean default True;
```

Remarks

Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings. Identifiers delimited by double quotation marks can be either Transact-SQL reserved keywords or can contain characters not usually allowed by the Transact-SQL syntax rules for identifiers.
True (the default value):
Identifiers can be delimited by double quotation marks, and literals must be delimited by single quotation marks.
All strings delimited by double quotation marks are interpreted as object identifiers. Therefore, quoted identifiers do not have to follow the Transact-SQL rules for identifiers. They can be reserved keywords and can include characters not always allowed in the Transact-SQL identifiers. Double quotation marks cannot be used to delimit literal string expressions; single quotation marks must be used to enclose literal strings. If a single quotation mark (') is a part of the literal string, it can be represented by two single quotation marks ("). QuotedIdentifier must be True when reserved keywords are used for object names in the database.
False (BDE compatibility):
Identifiers cannot be quoted and must follow all Transact-SQL rules for identifiers. Literals can be delimited by either single or double quotation marks. If a literal string is delimited by double quotation marks, the string can contain embedded single quotation marks such as apostrophes.
QuotedIdentifier must be True when creating or manipulating indexes on computed columns or indexed views. If QuotedIdentifier is False, CREATE, UPDATE, INSERT, and DELETE statements on tables with indexes on computed columns or indexed views will fail.

Used to manage field type creation for the NTEXT data type.

Class

TCustomMSConnectionOptions

Syntax

```
property UseWideMemos: boolean default True;
```

Remarks

If True (the default value), then TWideMemo fields are created for the NTEXT data type. If False, TMemo fields are created. This option is available for Delphi 2006 and higher.

17.14.1.3 MSAccess.TCustomMSDataSet Class

A component for defining the functionality for the classes derived from it.
For a list of all members of this type, see TCustomMSDataSet members.

Unit

`MSAccess`

Syntax

```
TCustomMSDataSet = class(TCustomDADataSet);
```

Remarks

TCustomMSDataSet is a base dataset component that defines the functionality for the classes derived from it. Applications never use TCustomMSDataSet objects directly. Instead they use descendants of TCustomMSDataSet, such as TMSQuery, TMSTable and TMSStoredProc, that inherit its dataset-related properties and methods.

Inheritance Hierarchy

TObject
  TMemDataSet
    TCustomDADataSet
      TCustomMSDataSet

See Also

- TMSQuery
- TCustomMSTable
- TCustomMSStoredProc
- Performance of Obtaining Data
- Master/Detail Relationships

TCustomMSDataSet 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	Points to a TMSChangeNotification component.
CommandTimeout	Used to specify the amount of time to attempt to execute a command.
Connection	Used to specify a connection object that will be used to connect to a data store.
CursorType	Cursor types supported by SQL Server.
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.
Encryption (inherited from **TCustomDADataSet**)	Used to specify the options of the data encryption in a dataset.

FetchAll	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 (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.
Options	Used to specify the behaviour of a TCustomMSDataSet object.
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 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.

RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
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.
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.
UpdateObject	Used to point to an update object component which provides SQL statements that perform updates of 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	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	Serves for the creating of a stored procedures 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.
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 **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.
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	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable	Locks a table to prevent multiple access to it.

MacroByName (inherited from **TCustomDADataSet**)	Finds a Macro with the name passed in Name.
OpenNext	Opens next rowset in the statement.
ParamByName	Provides access to a parameter by its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshQuick	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute	Occurs after executing insert, delete, update, lock and refresh operation.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.

BeforeUpdateExecute	Occurs before executing insert, delete, update, lock and refresh operation.
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.

Properties of the **TCustomMSDataSet** class.
For a complete list of the **TCustomMSDataSet** class members, see the TCustomMSDataSet 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	Points to a TMSChangeNotification component.
CommandTimeout	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection	Used to specify a connection object that will be used 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.

CursorType	Cursor types supported by SQL Server.
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.
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 decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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**)	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 **TCustomDADataSet**)	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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **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.
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 a TCustomMSDataSet object.
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 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.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
UpdateObject	Used to point to an update object component which provides SQL statements that perform updates of 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

- TCustomMSDataSet Class
- TCustomMSDataSet Class Members

Points to a TMSChangeNotification component.

Class

`TCustomMSDataSet`

Syntax

```
property ChangeNotification: TMSChangeNotification;
```

Remarks

Points to a TMSChangeNotification component used to handle events related to the server side changes.
Note: This property is not available of users of SDAC Standard Edition.

See Also

- TMSChangeNotification
- Options
- Options

Used to specify the amount of time to attempt to execute a command.

Class

TCustomMSDataSet

Syntax

```
property CommandTimeout: integer default 0;
```

Remarks

Specifies the time during which an attempt ot execute a command can be undertaken.
Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Is measured in seconds.
If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.
The default value is 0 (infinite).

See Also

- TMSConnection.ConnectionTimeout

Used to specify a connection object that will be used to connect to a data store.

Class

TCustomMSDataSet

Syntax

```
property Connection: TCustomMSConnection;
```

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 TCustomMSConnection or its descendant class objects.
At runtime, set the Connection property to reference an existing TCustomMSConnection object.

See Also

- TCustomMSConnection

Cursor types supported by SQL Server.

Class

TCustomMSDataSet

Syntax

```
property CursorType: TMSCursorType default ctDefaultResultSet;
```

Remarks

Depending on the text of the SQL statement cursor type and the value of the TCustomDADataSet.ReadOnly property when Options is True, cursor type can be modified while opening a dataset. To learn more about implicit alteration of cursors, refer to MSDN .
ctStatic, ctKeyset and ctDynamic cursors are server cursors. So the TCustomDADataSet.FetchRows, FetchAll, TMemDataSet.CachedUpdates properties don't have any influence on such cursors and only the

Options.CursorUpdate option does.
To learn how to choose cursor type, refer to MSDN .
The default value is ctDefaultResultSet.

See Also

- Performance of Obtaining Data
- Options

Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.

Class

`TCustomMSDataSet`

Syntax

```
property FetchAll: boolean default True;
```

Remarks

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. The default value is True.
Since SDAC 4.20, changing the value of the FetchAll option to True for a dataset open in the FetchAll=False mode will not lead to closing this dataset. This fill forces all records to be fetched to the client.
Note: When setting TCustomMSDataSet.FetchAll = False you should keep in mind that execution of such queries blocks the current session. In order to avoid blocking OLE DB creates additional session that causes the following problems:

- Each additional session runs outside the transaction context thus the TCustomDAConnection.Commit and TCustomDAConnection.Rollback operations in the main session won't apply changes made in additional sessions. This also concerns changes made by TDataSet.Post.
- No transactions can be started if there are underfetched datasets within the connection.
- Temporary tables created in one session are not accessible from other sessions therefore simultaneous using of FetchAll = False and temporary tables is impossible.
- When editing compound queries with ORDER BY clause setting FetchAll = False may lead to deadlock during TDataSet.Post.

Important: If there is more than one dataset attached to TMSConnection, setting FetchAll = False even in one of them may lead to the problems described above.
To prevent the TMSConnection object from creating additional connections for datasets that work in the FetchAll=False mode, you should enable the TMSConnectionOptions.MultipleActiveResultSets option. This option is only supported by SQL Server 2005 with using SQL Native Client as OLE DB provider.

See Also

- Performance of Obtaining Data
- TMSConnectionOptions.MultipleActiveResultSets
- TMSConnectionOptions.Provider

Used to specify the behaviour of a TCustomMSDataSet object.

Class

`TCustomMSDataSet`

Syntax

```
property Options: TMSDataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMSDataSet object.
Descriptions of all options are in the table below.

Option Name	Description
AllFieldsEditable	Not supported.
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on a query execution.
AutoRefresh	Used to enable automatic refresh of a dataset every AutoRefreshInterval seconds.
AutoRefreshInterval	Used to define in what time interval in seconds the Refresh or RefreshQuick method of DataSet is called.
CheckRowVersion	Used to determine whether dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.
CursorUpdate	Used to specify the way data updates reflect on database when modifying dataset by using server cursors ctKeySet and ctDynamic.
DefaultValues	Used to enable TCustomMSDataSet to fill the DefaultExpression property of TField objects by an appropriate value.
DescribeParams	Used to specify whether to query TMSParam properties from the server when executing the TCustomDADataSet.Prepare method.
DisableMultipleResults	Used to forbid multiple results usage by a command.
DMLRefresh	Used to refresh a record when insertion or update is performed.
EnableBCD	Used to specify whether to treat numeric fields as floating-point or BCD.
FullRefresh	Used to specify the fields to include in the automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method.
LongStrings	Represents string fields with the length that is greater than 255 as TStringField.
NonBlocking	Used to fetch rows in a separate thread.
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryIdentity	Used to specify whether to request the Identity field value on execution of the Insert or Append method.
QueryRecCount	Used to perform additional query to get record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomMSDataSet to quote all field names in autogenerated SQL statements.
ReflectChangeNotify	Indicates whether DataSet will be automatically refreshed when the underlying data on the server is changed.
RemoveOnRefresh	Used for dataset to locally remove record on refresh if it does not match filter condition (WHERE clause for refresh SQL) anymore.
RequiredFields	Used for TCustomMSDataSet to set the Required property of TField objects for the NOT NULL fields.
ReturnParams	Used to return the new values of fields to dataset after insert or update.

StrictUpdate	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar	Used to specify whether to discard all trailing spaces in the fixed-length string fields of a dataset.
TrimVarChar	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
UniqueRecords	Used to specify whether to query additional keyfields from the server.

Contains parameters for a query's SQL statement.

Class

`TCustomMSDataSet`

Syntax

```
property Params: TMSParams stored False;
```

Remarks

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 ero-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

- TMSParam
- ParamByName

Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.

Class

`TCustomMSDataSet`

Syntax

```
property UpdateObject: TMSUpdateSQL;
```

Remarks

The UpdateObject property points to an update object component which provides SQL statements that perform updates of read-only datasets when cached updates are enabled.

Methods of the **TCustomMSDataSet** class.
For a complete list of the **TCustomMSDataSet** class members, see the TCustomMSDataSet 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	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	Serves for the creating of a stored procedures 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.
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 **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**)	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	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable	Locks a table to prevent multiple access to it.
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.
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.
OpenNext	Opens next rowset in the statement.
Options (inherited from **TCustomDADataSet**)	Used to specify the behaviour of TCustomDADataSet object.
ParamByName	Provides access to a parameter by 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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**)	Resynchronizes 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.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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

- TCustomMSDataSet Class
- TCustomMSDataSet Class Members

Breaks execution of a SQL statement on the server.

Class

TCustomMSDataSet

Syntax

```
procedure BreakExec; override;
```

Remarks

Call the BreakExec method to break execution of a SQL statement on the server.
It makes sense to call BreakExec only from another thread.
Also you should remeber that calling BreakExec to interrupt dataset opening in the NonBlocking mode may not have effect if fetch has already begun (this happens when BreakExec falls between two fetch operations).

See Also

- TCustomDADataSet.Execute
- TMSConnection.OnInfoMessage
- TMSSQL.BreakExec

Serves for the creating of a stored procedures call.

Class

TCustomMSDataSet

Syntax

```
procedure CreateProcCall(Name: string);
```

Parameters

Name
Holds the name of a stored routine.

Remarks

Using the name of a stored procedure, a command for the call is generated and parameters are created. After a call to CreateProcCall the values of the parameters should be defined and and the procedure should be executed.

Indicates whether a parameter with the specified name exists in a dataset.

Class

TCustomMSDataSet

Syntax

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

Parameters

Value
Holds the name of the parameter to search for.

Return Value

A TMSParam object, if a param with the matching name was found.

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

See Also

- Params
- ParamByName

Used to create the TMSFileStream object for working with FILESTREAM data.

Class

TCustomMSDataSet

Syntax

```
function GetFileStreamForField(const FieldName: string; const
  DesiredAccess: TMSSqlFilestreamDesiredAccess = daReadWrite;
  const OpenOptions: TMSSqlFilestreamOpenOptions = []; const
  AllocationSize: Int64 = 0): TMSFileStream;
```

Parameters

FieldName
Contains the existing field name of a VARBINARY(MAX) column.

DesiredAccess
Determines the mode that is used to access FILESTREAM data.
The following values can be used:

- *daRead* - data can be read from the file.
- *daWrite* - data can be written to the file.
- *daReadWrite* - data can be read and written from the file.

OpenOptions
Determines file attributes and flags. By default, the file is being opened or created with no special options. The following values can be used:

- *ooAsync* - The file is being opened or created for asynchronous I/O.
- *ooNoBuffering* - The system opens the file without system caching.
- *ooNoWriteThrough* - The system does not write through an intermediate cache. Writes go directly to disk.
- *ooSequentialScan* - The file is being accessed sequentially from beginning to end. The system can use this as a hint to optimize file caching. If an application moves the file pointer for random access, optimal caching may not occur.
- *ooRandomAccess* - The file is being accessed randomly. The system can use this as a hint to

optimize file caching.

AllocationSize

Determines the initial allocation size of the data file in bytes. It is ignored in read mode. If this parameter is 0, the default file system behavior is used.

Return Value

The TMSFileStream object for working with FILESTREAM data.

Remarks

Creates the TMSFileStream object for working with FILESTREAM data of specified column. Transaction must be started before calling this method.
Note that this method requests server to obtain the Win32 compatible file handle for a FILESTREAM data.
To obtain the file handle, the following steps are performed:

1. getting the current transaction context of a session by calling the GET_FILESTREAM_TRANSACTION_CONTEXT Transact-SQL function. (http://msdn.microsoft.com/en-us/library/bb934014.aspx)
2. obtaining the Win32 file handle by executing the OpenSqlFilestream API. (http://msdn.microsoft.com/en-us/library/bb933972.aspx)

Note: You can find more information about working with FILESTREAM data in MSDN at http://msdn.microsoft.com/en-us/library/cc949109(v=sql.100).aspx

See Also

- TMSFileStream

Locks the current records to prevent multiple users' access to it.

Class

TCustomMSDataSet

Overload List

Name	Description
Lock	Locks the current records to prevent multiple users' access to it.
Lock(LockType: TMSLockType)	Locks the current records to prevent multiple users' access to it.

Locks the current records to prevent multiple users' access to it.

Class

TCustomMSDataSet

Syntax

```
procedure Lock; overload; override
```

Locks the current records to prevent multiple users' access to it.

Class

TCustomMSDataSet

Syntax

```
procedure Lock(LockType: TMSLockType); reintroduce; overload
```

Parameters

LockType
Holds the lock type.

Remarks

This method locks the current record in dataset to prevent multiple users' access to it.
Record lock can be performed only within a transaction. If an application cannot update/lock a record because it has already been locked, it will wait until the lock is released. When the server lock timeout has expired, but lock is not acquired, an exception will be raised. Lock is released when the transaction is committed/rolled back.
You should also be aware of the Lock Escalation mechanism of SQL Server using locking in SDAC. Locking multiple records in the same table may lead to the locking of a whole table. This will avoid the server's resources overrun.
Note There is an optimization for exclusive locks - SQL Server checks whether data has been changed since the transaction was started. If not, then a lock request is ignored. For more information see this topic of MSDN.

Example

To avoid this issue, you can refresh only locked record:

```
if not MSQuery.Connection.InTransaction then   // check whether the transa
      MSQuery.Connection.StartTransaction;            // run the transacti
   // setup how much time to wait before raising an exception
   // if the record is already locked by someone else
   MSQuery.Connection.ExecSQL('SET LOCK TIMEOUT ' + IntToStr(StrToInt(edL
   MSQuery.Lock(ltExclusive);         // perform exclusive lock
   MSQuery.RefreshRecord;               // make sure that the record is l
```

See Also

- TCustomMSDataSet.LockTable
- TCustomDAConnection.StartTransaction
- TCustomDAConnection.Commit
- TCustomDAConnection.Rollback

Locks a table to prevent multiple access to it.

Class

TCustomMSDataSet

Syntax

```
procedure LockTable(LockType: TMSLockType);
```

Parameters

LockType
Holds the lock type.

Remarks

This method locks the underlying dataset's table to prevent multiple users' access to it.
Table lock can be performed only within a transaction. If an application cannot update/lock a table because it has already been locked, it will wait until the lock is released. When the server lock timeout has expired, but lock is not acquired, an exception will be raised. Lock is released when the transaction is committed/rolled back.

See Also

- TCustomMSDataSet.Lock
- TCustomDAConnection.StartTransaction
- TCustomDAConnection.Commit
- TCustomDAConnection.Rollback

Opens next rowset in the statement.

Class

TCustomMSDataSet

Syntax

```
function OpenNext: boolean;
```

Return Value

True, if DataSet opens.

Remarks

Call the OpenNext method to get the second and other ResultSets while executing a multiresult query. If DataSet opens, it returns True. If there are no record sets to be represented, it will return False, and the current record set will be closed. Has effect only for the ctDefaultResultSet cursor. The OpenNext method isn't compatible with TCustomDADataSet.Prepare.

Example

Here is a small piece of code that demonstrates the approach of working with multiple datasets returned by a multi-statement query:

```
MSQuery.SQL.Clear;
   MSQuery.SQL.Add('SELECT * FROM Table1;');
   MSQuery.SQL.Add('SELECT * FROM Table2;');
   MSQuery.SQL.Add('SELECT * FROM Table3;');
   MSQuery.SQL.Add('SELECT * FROM Table4;');
   MSQuery.SQL.Add('SELECT * FROM Table5;');
   MSQuery.FetchAll := False;
   MSQuery.Open;
   repeat
      //   < do something >
      until not MSQuery.OpenNext;
```

See Also

- TCustomDADataSet.Execute

Provides access to a parameter by its name.

Class

TCustomMSDataSet

Syntax

```
function ParamByName(const Value: string): TMSParam;
```

Parameters

Value

Holds the name of the parameter to retrieve information for.

Return Value

a TMSParam 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 TMSParam 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

- TMSParam
- Params
- FindParam

An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.

Class

TCustomMSDataSet

Syntax

```
procedure RefreshQuick(const CheckDeleted: boolean);
```

Parameters

CheckDeleted

if True, records deleted by other clients will be checked additionally. If False, remote records are not checked.

Remarks

Call the RefreshQuick method to quickly retrieve to the client side changes applied to the server by other clients. The main difference between the RefreshQuick and Refresh methods is that RefreshQuick does not transfer to the client all data like the Refresh method does. The only rows that were added or modified from the moment of the last refresh are returned to a client. The necessity of data inquiry for each row is defined by the TIMESTAMP field. So the RefreshQuick method requires query to include a unique key fields and a TIMESTAMP field. If the CheckDeleted parameter value is True, records deleted by other clients will be checked additionally.
This method is especially effective for queries with huge data level in the single row.
This feature does not work with SQL Server Compact Edition.
Note: If RefreshQuick is called for a dataset which is ordered on the server (query includes the ORDER BY clause), dataset records ordering can be violated because not all records will be retrieved by this method. You can use local ordering to solve this problem. For more information about local ordering, see the TMemDataSet.IndexFieldNames property description.

Events of the **TCustomMSDataSet** class.
For a complete list of the **TCustomMSDataSet** class members, see the TCustomMSDataSet 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	Occurs after executing insert, delete, update, lock and refresh operation.
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	Occurs before executing insert, delete, update, lock and refresh operation.
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.
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.
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 **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**)	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 **TCustomDADataSet**)	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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **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.
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 **TCustomDADataSet**)	Used to specify the behaviour of TCustomDADataSet object.
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 (inherited from **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
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.
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**)	Resynchronizes 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.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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

- TCustomMSDataSet Class
- TCustomMSDataSet Class Members

Occurs after executing insert, delete, update, lock and refresh operation.

Class

TCustomMSDataSet

Syntax

```
property AfterUpdateExecute: TMSUpdateExecuteEvent;
```

Remarks

The AfterUpdateExecute event occurs after executing insert, delete, update, lock and refresh operation.
You can use AfterUpdateExecute to read parameters of corresponding statements.

See Also

- BeforeUpdateExecute

Occurs before executing insert, delete, update, lock and refresh operation.

Class

TCustomMSDataSet

Syntax

```
property BeforeUpdateExecute: TMSUpdateExecuteEvent;
```

Remarks

The BeforeUpdateExecute event occurs before executing insert, delete, update, lock and refresh operation. You can use BeforeUpdateExecute to set parameters of corresponding statements.

See Also

- AfterUpdateExecute

17.14.1.4 MSAccess.TCustomMSStoredProc Class

A component used to access stored procedures on a database server.
For a list of all members of this type, see TCustomMSStoredProc members.

Unit

MSAccess

Syntax

```
TCustomMSStoredProc = class(TCustomMSDataSet);
```

Remarks

TCustomMSStoredProc implements functionality to access stored procedures on a database server.
You need only to define the StoredProcName property, while not bothering about writing SQL statement by hand.
Use the Execute method at runtime to generate a request that instructs server to execute procedure and return parameters in the Params property.

Inheritance Hierarchy

TObject
  TMemDataSet
    TCustomDADataSet
      TCustomMSDataSet
        TCustomMSStoredProc

See Also

- TCustomMSDataSet
- TMSStoredProc
- Performance of Obtaining Data
- Master/Detail Relationships

TCustomMSStoredProc 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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used to connect to a data store.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
Encryption (inherited from **TCustomDADataSet**)	Used to specify the options of the data encryption in a dataset.
FetchAll (inherited from **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 (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.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
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.
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 stored procedure name that is to be called on the server.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.
UpdatingTable	Specifies 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 **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures 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.

ExecProc	Executes SQL statements 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 **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.
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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.

MacroByName (inherited from **TCustomDADataSet**)	Finds a Macro with the name passed in Name.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Builds a query for TCustomMSStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.

AfterUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
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.

Properties of the **TCustomMSStoredProc** class.
For a complete list of the **TCustomMSStoredProc** class members, see the TCustomMSStoredProc 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.

Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
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 **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
StoredProcName	Used to specify the stored procedure name that is to be called on the server.
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 **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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.

UpdatingTable	Specifies 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

- TCustomMSStoredProc Class
- TCustomMSStoredProc Class Members

Used to specify the stored procedure name that is to be called on the server.

Class

`TCustomMSStoredProc`

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 the existing stored procedure on the server, then when an application attempts to prepare the procedure prior to execution, an exception is raised.

Specifies 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

`TCustomMSStoredProc`

Syntax

```
property UpdatingTable: string;
```

Remarks

Ose the UpdatingTable property on Insert, Update, Delete, or RefreshRecord (see also TCustomMSDataSet.Options) if appropriate SQL (SQLInsert, SQLUpdate or SQLDelete) is not provided.
If UpdatingTable is not set then the first table used in query is assumed to be the target.
If a query is addressed to the View then entire View is taken as a target for subsequent modifications.
All fields from other than target table have their ReadOnly properties set to True (if TCustomMSDataSet.Options is True).
With TCustomMSDataSet.CursorType UpdatingTable can be used only if TCustomMSDataSet.Options = False.

Methods of the **TCustomMSStoredProc** class.
For a complete list of the **TCustomMSStoredProc** class members, see the TCustomMSStoredProc 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
ExecProc	Executes SQL statements 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.

FetchAll (inherited from **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.

IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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	Builds a query for TCustomMSStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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

- TCustomMSStoredProc Class
- TCustomMSStoredProc Class Members

Executes SQL statements on the server.

Class

TCustomMSStoredProc

Syntax

```
procedure ExecProc;
```

Remarks

Call the ExecProc method to execute a SQL statement on the server. If SQL statement is a query, ExecProc calls the Open method.
Internally ExecProc calls inherited TCustomDADataSet.Execute method and is only included for compatibility with BDE.

See Also

- TCustomDADataSet.Execute

Builds a query for TCustomMSStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property.

Class

TCustomMSStoredProc

Syntax

```
procedure PrepareSQL;
```

Remarks

Call the PrepareSQL method to build a query for TCustomMSStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property. Generated query is then verified to be valid and, if necessary, the list of parameters is modified.
PrepareSQL is called implicitly when TCustomMSStoredProc is executed.

See Also

- TCustomDADataSet.Params
- StoredProcName
- ExecProc

17.14.1.5 MSAccess.TCustomMSTable Class

A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
For a list of all members of this type, see TCustomMSTable members.

Unit

MSAccess

Syntax

```
TCustomMSTable = class(TCustomMSDataSet);
```

Remarks

TCustomMSTable implements functionality to access data in a table. Use TCustomMSTable properties and methods to gain direct access to records and fields in an underlying server database without writing SQL statements.

Inheritance Hierarchy

TObject
  TMemDataSet
    TCustomDADataSet
      TCustomMSDataSet
        TCustomMSTable

See Also

- TMSTable
- TMSStoredProc
- Performance of Obtaining Data

TCustomMSTable 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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used to connect to a data store.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
Encryption (inherited from **TCustomDADataSet**)	Used to specify the options of the data encryption in a dataset.
FetchAll (inherited from **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 (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.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
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.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
TableName	Used to specify the name of the database table that 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 **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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 **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures 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.
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 **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.
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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
MacroByName (inherited from **TCustomDADataSet**)	Finds a Macro with the name passed in Name.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Determines KeyFields and build query of TCustomMSTable.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.

SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
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.

Properties of the **TCustomMSTable** class.
For a complete list of the **TCustomMSTable** class members, see the TCustomMSTable 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
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 **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.

IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
OrderFields	Used to build ORDER BY clause of SQL statements.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
TableName	Used to specify the name of the database table that this component encapsulates.
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 **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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

- TCustomMSTable Class
- TCustomMSTable Class Members

Used to build ORDER BY clause of SQL statements.

Class

TCustomMSTable

Syntax

```
property OrderFields: string;
```

Remarks

TCustomMSTable uses the OrderFields property to build ORDER BY clause of SQL statements.
Place commas to separate fields in a single string.
TCustomMSTable is reopened when the OrderFields property is being changed.

See Also

- TCustomMSTable

Used to specify the name of the database table that this component encapsulates.

Class

TCustomMSTable

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to specify the name of the database table that this component encapsulates. At design-time select a valid table name from the TableName drop-down list in the Object Inspector.
Note: To work with temporary tables you must set TCustomMSDataSet.FetchAll to True (for details see the FetchAll description).

See Also

- TCustomMSTable
- TCustomDAConnection.GetTableNames

Methods of the **TCustomMSTable** class.
For a complete list of the **TCustomMSTable** class members, see the TCustomMSTable 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.

CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
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 **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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 build query of TCustomMSTable.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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

- TCustomMSTable Class
- TCustomMSTable Class Members

Determines KeyFields and build query of TCustomMSTable.

Class

`TCustomMSTable`

Syntax

```
procedure PrepareSQL;
```

Remarks

Call the PrepareSQL method to determine KeyFields and build query of TCustomMSTable. PrepareSQL is called implicitly when TCustomMSTable is being opened.

See Also

- OrderFields
- TableName
- TCustomDADataSet.FilterSQL

17.14.1.6 MSAccess.TMSChangeNotification 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 TMSChangeNotification members.

Unit

`MSAccess`

Syntax

```
TMSChangeNotification = class(TComponent);
```

Remarks

The TMSChangeNotification 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.
You should assign a TMSChangeNotification object to the TCustomMSDataSet.ChangeNotification property of the dataset you want to be notified about changes. One TMSChangeNotification object can be associated with multiple datasets.
Client is notified only about changes made in the actually selected data. For example, if you select records that match a condition from a table, notification about the changes in records that do not match provided condition will not be received.
A notification subscription is removed after the notification event occurs. You can reopen/refresh your dataset to get the newest data and renew the notification subscription.
The Query Notification does not support the DBPROP_UNIQUEROWS option that is required for editable datasets. Therefore TMSChangeNotification executes an additional query immediately after the main query has been executed, and before records have been fetched. As the main connection is busy, OLE DB creates an additional connection to the server to execute this query. This can slow down your application. Setting the TMSConnection.Options option of TMSConnection to True helps to prevent creating additional connections to server.
Requirements:

1. The Query Notifications mechanism was implemented in SQL Server 2005, therefore this component can be used only with SQL Server 2005 and SQL Native Client.
2. Provided statement should meet restrictions described in MSDN .

Inheritance Hierarchy

TObject
  TMSChangeNotification

See Also

- Change Notification demo
- TCustomMSDataSet.ChangeNotification
- TCustomMSDataSet.Options
- MSDN: Creating a Query for Notification
- MSDN: Working with Query Notifications
- MSDN: Using Query Notifications

TMSChangeNotification class overview.

Properties

Name	Description
Enabled	Used to enable or disable using change notification.
Service	Used to assign a service manually.
TimeOut	Indicates the interval for a notification to remain active.

Events

Name	Description
OnChange	Occurs when data in one of the associated datasets was changed on the server.

Properties of the **TMSChangeNotification** class.
For a complete list of the **TMSChangeNotification** class members, see the TMSChangeNotification Members topic.

Published

Name	Description
Enabled	Used to enable or disable using change notification.
Service	Used to assign a service manually.
TimeOut	Indicates the interval for a notification to remain active.

See Also

- TMSChangeNotification Class
- TMSChangeNotification Class Members

Used to enable or disable using change notification.

Class

`TMSChangeNotification`

Syntax

```
property Enabled: boolean default True;
```

Remarks

Set the Enabled property to False to disable change notification for all datasets connected to the TMSChangeNotification component. Setting this property to True allows datasets, connected to the TMSChangeNotification component, to use change notification.

Used to assign a service manually.

Class

TMSChangeNotification

Syntax

```
property Service: string;
```

Remarks

If this property is not assigned, TMSChangeNotification automatically creates a service and associates it with a queue in order to receive change notifications from this queue. The name of the automatically created service consists of the 'SDAC NS ' prefix and the current session identifier (SPID). The queue name consists of the service name and the ' QUEUE' postfix. Such service and queue are created for each connection.
If several DataSet components work through the same connection associated with the TMSChangeNotification component, only one service and one queue will be used. After all DataSets of a connection are closed, and notifications are not necessary, the service and the queue are dropped. Also if there are invalid services and queues at the server, they will be dropped. A server or a queue is considered invalid if there is no connection with the corresponding SPID. This should be done in order to prevent clogging the server with unused services and queues, and to remove all unused notifications.
If a service name is assigned via this property, it is necessary for you to create the service manually. The service should be created according to the rules of such object creation for Query Notification. Manually assigned service will not be deleted by SDAC after all datasets using it are closed. It means that the notification subscription will stay active, and when the query is opened next time, it will be able to receive notifications.
You should remember that several applications, or several instances of the same application using the same service name, may work incorrectly, as they will obtain notifications from the same queue. To avoid possible problems, it is necessary to use a separate service for each connection (if Service is not assigned, this is done automatically).

Indicates the interval for a notification to remain active.

Class

TMSChangeNotification

Syntax

```
property TimeOut: integer default 432000;
```

Remarks

Set the TimeOut property to determine time interval in seconds, after which the notification registration will expire.
The default value is 432000, which equals to 5 days. The minimum value is 1 second, maximum is 2^31-1 seconds.

Events of the **TMSChangeNotification** class.
For a complete list of the **TMSChangeNotification** class members, see the TMSChangeNotification Members topic.

Published

Name	Description
OnChange	Occurs when data in one of the associated datasets was changed on the server.

See Also

- TMSChangeNotification Class
- TMSChangeNotification Class Members

Occurs when data in one of the associated datasets was changed on the server.

Class

`TMSChangeNotification`

Syntax

```
property OnChange: TMSChangeNotificationEvent;
```

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 DataSet parameter points to the dataset affected by this change. Other parameters provide detailed information about the change.

See Also

- Enabled

17.14.1.7 MSAccess.TMSConnection Class

A component for establishing connection to the database server, providing customized login support and performing transaction control.
For a list of all members of this type, see TMSConnection members.

Unit

`MSAccess`

Syntax

```
TMSConnection = class(TCustomMSConnection);
```

Remarks

TMSConnection publishes connection-related properties derived from its ancestor class TCustomDAConnection and introduces OLE DB specific properties, which give more control over transactions.
Note: if you would like to use SDAC in a service, console or just in a separate thread, you need to call CoInitialize for each thread. Also call CoUnInitialize when the thread is finished.

Inheritance Hierarchy

TObject
  TCustomDAConnection
    TCustomMSConnection
      TMSConnection

See Also

- TCustomMSDataSet.Connection
- TMSSQL.Connection

TMSConnection class overview.

Properties

Name	Description

Authentication	Used to specify the authentication service used by the database server to identify a user.
ClientVersion (inherited from **TCustomMSConnection**)	Contains the version of Microsoft OLE DB Provider for SQL Server.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectionTimeout	Used to specify the amount of time before an attempt to make a connection is considered unsuccessful.
ConnectString (inherited from **TCustomMSConnection**)	Used to specify the connection information for the data store.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Database (inherited from **TCustomMSConnection**)	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
IsolationLevel (inherited from **TCustomMSConnection**)	Used to specify the extent to which all outside transactions interfere with subsequent transactions of the current connection.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Used to specify the behaviour of a TMSConnection object.
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.
Server (inherited from **TCustomDAConnection**)	Serves to supply the server name for login.
ServerVersion (inherited from **TCustomMSConnection**)	Contains the exact number of the SQL Server version.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

Methods

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect (inherited from **TCustomMSConnection**)	Shares database connection between the TCustomMSConnection components.
ChangePassword	Assignes a new password instead of an expired 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 **TCustomMSConnection**)	Returns a new instance of the TMSSQL class and associates it with this connection object.

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.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
OpenDatasets (inherited from **TCustomMSConnection**)	Opens several datasets as one batch.
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.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.

Events

Name	Description
OnConnectionLost (inherited from **TCustomDAConnection**)	This event occurs when connection was lost.
OnError (inherited from **TCustomDAConnection**)	This event occurs when an error has arisen in the connection.
OnInfoMessage	Occurs if a SQL Server info message was generated.

Properties of the **TMSConnection** class.
For a complete list of the **TMSConnection** class members, see the TMSConnection Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect (inherited from **TCustomMSConnection**)	Shares database connection between the TCustomMSConnection components.
ClientVersion (inherited from **TCustomMSConnection**)	Contains the version of Microsoft OLE DB Provider for SQL Server.
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.
ConnectString (inherited from **TCustomMSConnection**)	Used to specify the connection information for the data store.

ConvertEOL (inherited from **TCustomDAConnection**)	Allows customi ing line breaks in string fields and parameters.
CreateDataSet (inherited from **TCustomDAConnection**)	Creates a dataset component.
CreateSQL (inherited from **TCustomMSConnection**)	Returns a new instance of the TMSSQL class and associates it with this connection object.
Database (inherited from **TCustomMSConnection**)	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
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.
IsolationLevel (inherited from **TCustomMSConnection**)	Used to specify the extent to which all outside transactions interfere with subsequent transactions 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.
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.
OpenDatasets (inherited from **TCustomMSConnection**)	Opens several datasets as one batch.
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.
ServerVersion (inherited from **TCustomMSConnection**)	Contains the exact number of the SQL Server version.

StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

Published

Name	Description
Authentication	Used to specify the authentication service used by the database server to identify a user.
ConnectionTimeout	Used to specify the amount of time before an attempt to make a connection is considered unsuccessful.
Options	Used to specify the behaviour of a TMSConnection object.

See Also

- TMSConnection Class
- TMSConnection Class Members

Used to specify the authentication service used by the database server to identify a user.

Class

`TMSConnection`

Syntax

```
property Authentication: TMSAuthentication default auServer;
```

Remarks

Use the Authentication property to specify the authentication service used by the database server to identify a user.
If you need to use this property at run-time, you must use the OLEDBAccess unit.

See Also

- TCustomDAConnection.Username
- TCustomDAConnection.Password

Used to specify the amount of time before an attempt to make a connection is considered unsuccessful.

Class

`TMSConnection`

Syntax

```
property ConnectionTimeout: integer default 15;
```

Remarks

Use the ConnectionTimeout property to specify the amount of time in seconds before an attempt to make a connection is considered unsuccessful.
The default value is 15 seconds.

See Also

- TCustomMSDataSet.CommandTimeout
- TMSSQL.CommandTimeout

Used to specify the behaviour of a TMSConnection object.

Class

TMSConnection

Syntax

```
property Options: TMSConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMSConnection object.
Descriptions of all options are in the table below.

Option Name	Description
ApplicationName	The name of a client application. The default value is the name of the executable file of your application.
AutoTranslate	Used to translate character strings sent between the client and server by converting through Unicode.
DefaultLockTimeout	Specifies how much time in milliseconds a transaction will wait for a lock.
Encrypt	Specifies if data should be encrypted before sending it over the network.
FailoverPartner	Specifies the SQL Server name to which SQL Native Client will reconnect when a failover of the principal SQL Server occurs.
InitialFileName	Specifies the name of the main database file.
Language	Specifies the SQL Server language name.
MultipleActiveResultSets	Enables support for the SQL Server 2005 Multiple Active Result Sets (MARS) technology.
NetworkLibrary	Specifies the name of the Net-Library (DLL) used to communicate with an instance of SQL Server.
PacketSize	Network packet size in bytes.
PersistSecurityInfo	Used to allow the data source object to persist sensitive authentication information such as a password along with other authentication information.
Provider	Used to specify a provider from the list of supported providers.
TrustServerCertificate	Used to enable traffic encryption without validation.
WorkstationID	A string identifying the workstation.

Methods of the **TMSConnection** class.
For a complete list of the **TMSConnection** class members, see the TMSConnection Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.

AssignConnect (inherited from **TCustomMSConnection**)	Shares database connection between the TCustomMSConnection components.
ChangePassword	Assignes a new password instead of an expired one..
ClientVersion (inherited from **TCustomMSConnection**)	Contains the version of Microsoft OLE DB Provider for SQL Server.
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.
ConnectString (inherited from **TCustomMSConnection**)	Used to specify the connection information for the data store.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customi ing line breaks in string fields and parameters.
CreateDataSet (inherited from **TCustomDAConnection**)	Creates a dataset component.
CreateSQL (inherited from **TCustomMSConnection**)	Returns a new instance of the TMSSQL class and associates it with this connection object.
Database (inherited from **TCustomMSConnection**)	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
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.
IsolationLevel (inherited from **TCustomMSConnection**)	Used to specify the extent to which all outside transactions interfere with subsequent transactions 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.
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.
OpenDatasets (inherited from **TCustomMSConnection**)	Opens several datasets as one batch.

Options (inherited from **TCustomMSConnection**)	Used to specify the behaviour of a TCustomMSConnection object.
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.
ServerVersion (inherited from **TCustomMSConnection**)	Contains the exact number of the SQL Server version.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

See Also

- TMSConnection Class
- TMSConnection Class Members

Assignes a new password instead of an expired one..

Class

TMSConnection

Syntax

```
procedure ChangePassword(NewPassword: string);
```

Parameters

NewPassword
Holds the new password assigned.

Remarks

Use the ChangePassword method to change an expired user's password. In SQL Server versions prior to SQL Server 2005 only a database administrator has permissions to change an expired user's password. With SQL Server 2005 and SQL Native Client you can change it using the ChangePassword method.
Note: Only an expired user's password can be changed using this method.

See Also

- MSDN: Changing Passwords Programmatically

Events of the **TMSConnection** class.
For a complete list of the **TMSConnection** class members, see the TMSConnection Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.

AssignConnect (inherited from **TCustomMSConnection**)	Shares database connection between the TCustomMSConnection components.
ClientVersion (inherited from **TCustomMSConnection**)	Contains the version of Microsoft OLE DB Provider for SQL Server.
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.
ConnectString (inherited from **TCustomMSConnection**)	Used to specify the connection information for the data store.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
CreateDataSet (inherited from **TCustomDAConnection**)	Creates a dataset component.
CreateSQL (inherited from **TCustomMSConnection**)	Returns a new instance of the TMSSQL class and associates it with this connection object.
Database (inherited from **TCustomMSConnection**)	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
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.
IsolationLevel (inherited from **TCustomMSConnection**)	Used to specify the extent to which all outside transactions interfere with subsequent transactions 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.
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.
OpenDatasets (inherited from **TCustomMSConnection**)	Opens several datasets as one batch.
Options (inherited from **TCustomMSConnection**)	Used to specify the behaviour of a TCustomMSConnection object.

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.
ServerVersion (inherited from **TCustomMSConnection**)	Contains the exact number of the SQL Server version.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

Published

Name	Description
OnInfoMessage	Occurs if a SQL Server info message was generated.

See Also

- TMSConnection Class
- TMSConnection Class Members

Occurs if a SQL Server info message was generated.

Class

`TMSConnection`

Syntax

```
property OnInfoMessage: TMSConnectionInfoMessageEvent;
```

Remarks

The OnInfoMessage event occurs in case of generation of a SQL Server info message. The event occurs only if the command is executed through a dataset descendant (TMSQuery, TMSStoredProc). To make this event occur for TMSScript, TMSScript.DataSet should be set. It does not work for TMSSQL. The following is the list of Transact-SQL commands that generate info messages:
PRINT
RAISERROR with a severity of 10 or lower
DBCC
SET SHOWPLAN
SET STATISTICS.

See Also

- EMSError

17.14.1.8 MSAccess.TMSConnectionOptions Class

This class allows setting up the behaviour of the TMSConnection class.
For a list of all members of this type, see TMSConnectionOptions members.

Unit

MSAccess

Syntax

```
TMSConnectionOptions = class(TCustomMSConnectionOptions);
```

Inheritance Hierarchy

TObject
  TDAConnectionOptions
    TCustomMSConnectionOptions
      TMSConnectionOptions

TMSConnectionOptions class overview.

Properties

Name	Description
ApplicationName	The name of a client application. The default value is the name of the executable file of your application.
AutoTranslate	Used to translate character strings sent between the client and server by converting through Unicode.
DefaultLockTimeout	Specifies how much time in milliseconds a transaction will wait for a lock.
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet. IndexFieldNames property of a dataset.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
Encrypt	Specifies if data should be encrypted before sending it over the network.
FailoverPartner	Specifies the SQL Server name to which SQL Native Client will reconnect when a failover of the principal SQL Server occurs.
InitialFileName	Specifies the name of the main database file.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
Language	Specifies the SQL Server language name.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection. OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
MultipleActiveResultSets	Enables support for the SQL Server 2005 Multiple Active Result Sets (MARS) technology.
NetworkLibrary	Specifies the name of the Net-Library (DLL) used to communicate with an instance of SQL Server.

NumericType (inherited from **TCustomMSConnectionOptions**)	Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.
PacketSize	Network packet size in bytes.
PersistSecurityInfo	Used to allow the data source object to persist sensitive authentication information such as a password along with other authentication information.
Provider	Used to specify a provider from the list of supported providers.
QuotedIdentifier (inherited from **TCustomMSConnectionOptions**)	Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.
TrustServerCertificate	Used to enable traffic encryption without validation.
UseWideMemos (inherited from **TCustomMSConnectionOptions**)	Used to manage field type creation for the NTEXT data type.
WorkstationID	A string identifying the workstation.

Properties of the **TMSConnectionOptions** class.
For a complete list of the **TMSConnectionOptions** class members, see the TMSConnectionOptions Members topic.

Public

Name	Description
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet. IndexFieldNames property of a dataset.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection. OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
NumericType (inherited from **TCustomMSConnectionOptions**)	Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.
QuotedIdentifier (inherited from **TCustomMSConnectionOptions**)	Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.
UseWideMemos (inherited from **TCustomMSConnectionOptions**)	Used to manage field type creation for the NTEXT data type.

Published

Name	Description
ApplicationName	The name of a client application. The default value is the name of the executable file of your application.
AutoTranslate	Used to translate character strings sent between the client and server by converting through Unicode.
DefaultLockTimeout	Specifies how much time in milliseconds a transaction will wait for a lock.
Encrypt	Specifies if data should be encrypted before sending it over the network.
FailoverPartner	Specifies the SQL Server name to which SQL Native Client will reconnect when a failover of the principal SQL Server occurs.
InitialFileName	Specifies the name of the main database file.
Language	Specifies the SQL Server language name.
MultipleActiveResultSets	Enables support for the SQL Server 2005 Multiple Active Result Sets (MARS) technology.
NetworkLibrary	Specifies the name of the Net-Library (DLL) used to communicate with an instance of SQL Server.
PacketSize	Network packet size in bytes.
PersistSecurityInfo	Used to allow the data source object to persist sensitive authentication information such as a password along with other authentication information.
Provider	Used to specify a provider from the list of supported providers.
TrustServerCertificate	Used to enable traffic encryption without validation.
WorkstationID	A string identifying the workstation.

See Also

- TMSConnectionOptions Class
- TMSConnectionOptions Class Members

The name of a client application. The default value is the name of the executable file of your application.

Class

`TMSConnectionOptions`

Syntax

```
property ApplicationName: string;
```

Remarks

Use the ApplicationName property to specify the name of a client application. The default value is the name of the executable file of your application.

Used to translate character strings sent between the client and server by converting through Unicode.

Class

TMSConnectionOptions

Syntax

```
property AutoTranslate: boolean default True;
```

Remarks

When set to True, character strings sent between the client and server are translated by converting through Unicode to minimize problems in matching extended characters between the code pages on the client and server.

Specifies how much time in milliseconds a transaction will wait for a lock.

Class

TMSConnectionOptions

Syntax

```
property DefaultLockTimeout: integer;
```

Remarks

Use the DefaultLockTimeout property to specify how much time in milliseconds a transaction will wait for a lock. The default value is 2000 ms.

Specifies if data should be encrypted before sending it over the network.

Class

TMSConnectionOptions

Syntax

```
property Encrypt: boolean;
```

Remarks

Use the Encrypt property to specify if data should be encrypted before sending it over the network. The default value is False.

Specifies the SQL Server name to which SQL Native Client will reconnect when a failover of the principal SQL Server occurs.

Class

TMSConnectionOptions

Syntax

```
property FailoverPartner: string;
```

Remarks

Use the FailoverPartner property to specify the SQL Server name to which SQL Native Client will reconnect when a failover of the principal SQL Server occurs. This option is supported only for SQL Server 2005 using SQL Native Client as OLE DB provider.

Specifies the name of the main database file.

Class

TMSConnectionOptions

Syntax

```
property InitialFileName: string;
```

Remarks

Use the InitialFileName property to specify the name of the main database file. This database will be the default database for the connection. SQL Server attaches the database to the server if it has not been attached to the server yet. So, this property can be used to connect to the database that has not been attached to the server yet.

Specifies the SQL Server language name.

Class

TMSConnectionOptions

Syntax

```
property Language: string;
```

Remarks

Use the Language property to specify the SQL Server language name. Identifies the language used for system message selection and formatting. The language must be installed on the computer running an instance of SQL Server otherwise the connection will fail.

Enables support for the SQL Server 2005 Multiple Active Result Sets (MARS) technology.

Class

TMSConnectionOptions

Syntax

```
property MultipleActiveResultSets: boolean default False;
```

Remarks

Use the MultipleActiveResultSets property to enable support for the SQL Server 2005 Multiple Active Result Sets (MARS) technology. It allows applications to have more than one pending request per connection, and, in particular, to have more than one active default result set per connection. Current session is not blocked when using FetchAll = False, and it is not necessary for OLE DB to create additional sessions for any query executing. MARS is only supported by SQL Server 2005 with using SQL Native Client as OLE DB provider.

Specifies the name of the Net-Library (DLL) used to communicate with an instance of SQL Server.

Class

TMSConnectionOptions

Syntax

```
property NetworkLibrary: string;
```

Remarks

The name of the Net-Library (DLL) used to communicate with an instance of SQL Server. The name should not include the path or the .dll file name extension. The default name is provided by SQL Server

Client Network Utility.

Network packet size in bytes.

Class

TMSConnectionOptions

Syntax

```
property PacketSize: integer default 4096;
```

Remarks

Use the PacketSize property to specify the network packet size in bytes. The packet size property value must be between 512 and 32,767. The default network packet size is 4,096.

Used to allow the data source object to persist sensitive authentication information such as a password along with other authentication information.

Class

TMSConnectionOptions

Syntax

```
property PersistSecurityInfo: boolean default False;
```

Remarks

If True, the data source object is allowed to persist sensitive authentication information such as a password along with other authentication information.

Used to specify a provider from the list of supported providers.

Class

TMSConnectionOptions

Syntax

```
property Provider: TOLEDBProvider;
```

Remarks

Use the Provider property to specify a provider from the list of supported providers. The default value of this property is prAuto. In this case a provider of the most recent version is used. Some features in SQL Server require the SQL Native Client (prNativeClient) provider to be used. If chosen provider is not installed, an exception is raised. The prCompact value should be set for working with SQL Server Compact Edition.

Used to enable traffic encryption without validation.

Class

TMSConnectionOptions

Syntax

```
property TrustServerCertificate: boolean default False;
```

Remarks

Use the TrustServerCertificate property to enable traffic encryption without validation. The default value is False. This option is only supported by SQL Server 2005 with using SQL Native Client as OLE DB provider.

A string identifying the workstation.

Class

TMSConnectionOptions

Syntax

```
property WorkstationID: string;
```

Remarks

A string identifying the workstation. The default value is the name of your machine.

17.14.1.9 MSAccess.TMSDataSetOptions Class

This class allows setting up the behaviour of the TMSDataSet class.
For a list of all members of this type, see TMSDataSetOptions members.

Unit

MSAccess

Syntax

```
TMSDataSetOptions = class(TDADataSetOptions);
```

Inheritance Hierarchy

TObject
  TDADataSetOptions
    TMSDataSetOptions

TMSDataSetOptions class overview.

Properties

Name	Description
AllFieldsEditable	Not supported.
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on a query execution.
AutoRefresh	Used to enable automatic refresh of a dataset every AutoRefreshInterval seconds.
AutoRefreshInterval	Used to define in what time interval in seconds the Refresh or TCustomMSDataSet.RefreshQuick method of DataSet is called.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CheckRowVersion	Used to determine whether dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.

CursorUpdate	Used to specify the way data updates reflect on database when modifying dataset by using server cursors ctKeySet and ctDynamic.
DefaultValues	Used to enable TCustomMSDataSet to fill the DefaultExpression property of TField objects by an appropriate value.
DescribeParams	Used to specify whether to query TMSParam properties from the server when executing the TCustomDADataSet.Prepare method.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
DisableMultipleResults	Used to forbid multiple results usage by a command.
DMLRefresh	Used to refresh a record when insertion or update is performed.
EnableBCD	Used to specify whether to treat numeric fields as floating-point or BCD.
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 specify the fields to include in the automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method.
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	Represents string fields with the length that is greater than 255 as TStringField.
NonBlocking	Used to fetch rows in a separate thread.
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryIdentity	Used to specify whether to request the Identity field value on execution of the Insert or Append method.
QueryRecCount	Used to perform additional query to get record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomMSDataSet to quote all field names in autogenerated SQL statements.

ReflectChangeNotify	Indicates whether DataSet will be automatically refreshed when the underlying data on the server is changed.
RemoveOnRefresh	Used for dataset to locally remove record on refresh if it does not match filter condition (WHERE clause for refresh SQL) anymore.
RequiredFields	Used for TCustomMSDataSet to set the Required property of TField objects for the NOT NULL fields.
ReturnParams	Used to return the new values 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	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar	Used to specify whether to discard all trailing spaces in the fixed-length string fields of a dataset.
TrimVarChar	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
UniqueRecords	Used to specify whether to query additional keyfields from the server.
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.

Properties of the **TMSDataSetOptions** class.
For a complete list of the **TMSDataSetOptions** class members, see the TMSDataSetOptions Members topic.

Public

Name	Description
AllFieldsEditable	Not supported.
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.
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.
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
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on a query execution.
AutoRefresh	Used to enable automatic refresh of a dataset every AutoRefreshInterval seconds.
AutoRefreshInterval	Used to define in what time interval in seconds the Refresh or TCustomMSDataSet.RefreshQuick method of DataSet is called.
CheckRowVersion	Used to determine whether dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.
CursorUpdate	Used to specify the way data updates reflect on database when modifying dataset by using server cursors ctKeySet and ctDynamic.
DefaultValues	Used to enable TCustomMSDataSet to fill the DefaultExpression property of TField objects by an appropriate value.
DescribeParams	Used to specify whether to query TMSParam properties from the server when executing the TCustomDADataSet.Prepare method.
DisableMultipleResults	Used to forbid multiple results usage by a command.
DMLRefresh	Used to refresh a record when insertion or update is performed.
EnableBCD	Used to specify whether to treat numeric fields as floating-point or BCD.

FullRefresh	Used to specify the fields to include in the automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method.
LongStrings	Represents string fields with the length that is greater than 255 as TStringField.
NonBlocking	Used to fetch rows in a separate thread.
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryIdentity	Used to specify whether to request the Identity field value on execution of the Insert or Append method.
QueryRecCount	Used to perform additional query to get record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomMSDataSet to quote all field names in autogenerated SQL statements.
ReflectChangeNotify	Indicates whether DataSet will be automatically refreshed when the underlying data on the server is changed.
RemoveOnRefresh	Used for dataset to locally remove record on refresh if it does not match filter condition (WHERE clause for refresh SQL) anymore.
RequiredFields	Used for TCustomMSDataSet to set the Required property of TField objects for the NOT NULL fields.
ReturnParams	Used to return the new values of fields to dataset after insert or update.
StrictUpdate	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar	Used to specify whether to discard all trailing spaces in the fixed-length string fields of a dataset.
TrimVarChar	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
UniqueRecords	Used to specify whether to query additional keyfields from the server.

See Also

- TMSDataSetOptions Class
- TMSDataSetOptions Class Members

Not supported.

Class

`TMSDataSetOptions`

Syntax

```
property AllFieldsEditable: boolean;
```

Remarks

Refer to TCustomDADataSet.Options.

Used to execute automatic TCustomDADataSet.Prepare on a query execution.

Class

TMSDataSetOptions

Syntax

```
property AutoPrepare: boolean;
```

Remarks

Use the AutoPrepare property to execute automatic TCustomDADataSet.Prepare on a query execution. Makes sense for the cases when a query will be executed several times, for example, in Master/Detail relationships.

Used to enable automatic refresh of a dataset every AutoRefreshInterval seconds.

Class

TMSDataSetOptions

Syntax

```
property AutoRefresh: boolean default False;
```

Remarks

If True, dataset will be automatically refreshed every AutoRefreshInterval seconds. If dataset has at least one key field and a TIMESTAMP field, the TCustomMSDataSet.RefreshQuick method will be executed, otherwise the Refresh method will be executed.

Used to define in what time interval in seconds the Refresh or TCustomMSDataSet.RefreshQuick method of DataSet is called.

Class

TMSDataSetOptions

Syntax

```
property AutoRefreshInterval: integer default 60;
```

Remarks

Use the AutoRefreshInterval property to define in what time interval in seconds the Refresh or TCustomMSDataSet.RefreshQuick method of DataSet is called.

Used to determine whether dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.

Class

TMSDataSetOptions

Syntax

```
property CheckRowVersion: boolean default False;
```

Remarks

Use the CheckRowVersion property to determine whether dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data. If the CheckRowVersion property is False and DataSet has keyfields, the WHERE clause of SQL statement is generated basing on these keyfields. If there is no primary key and no Identity field, then all non-BLOB fields will take part in generating SQL statements. If CheckRowVersion is True and DataSet has TIMESTAMP field, only this field is included into the WHERE clause of the generated SQL statement. Otherwise, all non-BLOB fields are included. All mentioned fields refer to the current TMSQuery.UpdatingTable. The default value is False.
The CheckRowVersion option requires enabled TCustomMSDataSet.Options.

Used to specify the way data updates reflect on database when modifying dataset by using server cursors ctKeySet and ctDynamic.

Class

TMSDataSetOptions

Syntax

```
property CursorUpdate: boolean default True;
```

Remarks

Use the CursorUpdate property to specify the way data updates reflect on database when modifying dataset by using server cursors ctKeySet and ctDynamic. If the CursorUpdate property is True, all dataset modifications are passed to the database by server cursors. If the CursorUpdate property is False, all dataset updates are passed to the server by the generated automatically SQL statements or specified in TCustomDADataSet.SQLUpdate, TCustomDADataSet.SQLInsert or TCustomDADataSet.SQLDelete. The default value is True.

Used to enable TCustomMSDataSet to fill the DefaultExpression property of TField objects by an appropriate value.

Class

TMSDataSetOptions

Syntax

```
property DefaultValues: boolean;
```

Remarks

If True, TCustomMSDataSet fills the DefaultExpression property of TField objects by an appropriate value.

Used to specify whether to query TMSParam properties from the server when executing the TCustomDADataSet.Prepare method.

Class

TMSDataSetOptions

Syntax

```
property DescribeParams: boolean default False;
```

Remarks

Specifies whether to query TMSParam properties (Name, ParamType, DataType, Size, TableTypeName) from the server when executing the TCustomDADataSet.Prepare method. The default value is False.

Used to forbid multiple results usage by a command.

Class

TMSDataSetOptions

Syntax

```
property DisableMultipleResults: boolean default False;
```

Remarks

Use the DisableMultipleResults property to forbid using multiple results by a command.
Set this property to True to disable the multiple results usage. In this case, if you open a query with a big amount of data and you have to break the execution of this query (by calling TCustomMSDataSet.BreakExec or TDataSet.Close), the execution will be broken quickly. The default value is False.

Used to refresh a record when insertion or update is performed.

Class

TMSDataSetOptions

Syntax

```
property DMLRefresh: boolean default False;
```

Remarks

Use the DMLRefresh property to refresh a record when insertion or update is performed. This feature doesn't support SQL Server Compact Edition. The default value is False.

Used to specify whether to treat numeric fields as floating-point or BCD.

Class

TMSDataSetOptions

Syntax

```
property EnableBCD: boolean;
```

Remarks

Use the Enable BCD property to specify whether to treat numeric fields as floating-point or BCD. Use the EnableBCD property to specify how fields are mapped to field classes. If EnableBCD is True, decimal and numeric fields are mapped to the TBCDField class when field objects are created. If EnableBCD is False, the fields are mapped to the TFloatField class. EnableBCD determines whether numeric and decimal fields are translated as floating-point values or currency values. Currency values eliminate the rounding errors associated with the floating point math (such as a 3 * (2/3) resulting in 2.00000000001). The default value is False.

Used to specify the fields to include in the automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method.

Class

TMSDataSetOptions

Syntax

```
property FullRefresh: boolean default False;
```

Remarks

Use the FullRefresh property to specify what fields to include in the automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. If the FullRefresh property is True, all fields from a query are included into SQL statement to refresh a single record. If FullRefresh is False, only fields from TMSQuery.UpdatingTable are included.
Note: If FullRefresh is True, the refresh of SQL statement for complex queries and views may be generated with errors. The default value is False.

Represents string fields with the length that is greater than 255 as TStringField.

Class

TMSDataSetOptions

Syntax

```
property LongStrings: boolean;
```

Remarks

Represents string fields with the length that is greater than 255 as TStringField, not as TMemoField. The default value is True.

Used to fetch rows in a separate thread.

Class

TMSDataSetOptions

Syntax

```
property NonBlocking: boolean default False;
```

Remarks

Set the NonBlocking option to True to fetch rows in a separate thread. The BeforeFetch event is called in the additional thread context that performs data fetching. This event is called every time on the Fetch method call. The AfterFetch event is called in the main thread context only once after fetching is completely finished.
In the NonBlocking mode as well as in the FetchAll=False mode an extra connection is created. When setting TCustomMSDataSet.Options.NonBlocking to True, you should keep in mind that execution of such queries blocks the current session. In order to avoid blocking, OLE DB creates an additional session as in the TCustomMSDataSet.FetchAll = False mode. It causes the same problems as in the TCustomMSDataSet.FetchAll = False mode. This problem can be solved by using MARS (TMSConnection.Options = True). The current session is not blocked and OLE DB is not required to create addition session to run a query. MARS is supported only by SQL Server 2005 if SQL Native Client is used as OLE DB provider.

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

TMSDataSetOptions

Syntax

```
property NumberRange: boolean;
```

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values. The default value is False.

Used to specify whether to request the Identity field value on execution of the Insert or Append method.

Class

TMSDataSetOptions

Syntax

```
property QueryIdentity: boolean default True;
```

Remarks

Specifies whether to request the Identity field value, if such exists, on execution the Insert or Append method. If you don't request Identity, you can have an impact on performance of Insert or Append for about 20%. Affects only the TCustomMSDataSet.CursorType cursor. If you insert a value into the SQL VARIANT field and QueryIdentity is True, then EOLEDBError raised. The default value is True.

Used to perform additional query to get record count for this SELECT, so the RecordCount property reflects the actual number of records.

Class

TMSDataSetOptions

Syntax

```
property QueryRecCount: boolean;
```

Remarks

If True, and the TCustomMSDataSet.FetchAll property is False or the NonBlocking option is True, TCustomDADataSet performs additional query to get 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 and the NonBlocking option is False.

Used for TCustomMSDataSet to quote all field names in autogenerated SQL statements.

Class

TMSDataSetOptions

Syntax

```
property QuoteNames: boolean;
```

Remarks

If True, TCustomMSDataSet quotes all field names in autogenerated SQL statements such as update SQL.

Indicates whether DataSet will be automatically refreshed when the underlying data on the server is changed.

Class

TMSDataSetOptions

Syntax

```
property ReflectChangeNotify: boolean default False;
```

Remarks

Indicates whether DataSet will be automatically refreshed when the underlying data on the server is changed. Automatic refresh happens when ReflectChangeNotify is True, the TCustomMSDataSet.ChangeNotification property is assigned, and the TMSChangeNotification.OnChange parameter is nsData. This option is available only for users of SDAC *Professional Edition* .

Used for dataset to locally remove record on refresh if it does not match filter condition (WHERE clause for refresh SQL) anymore.

Class

TMSDataSetOptions

Syntax

```
property RemoveOnRefresh: boolean;
```

Remarks

When the RemoveOnRefresh property is set to True, dataset locally removes record on refresh if it does not match filter condition (WHERE clause for refresh SQL) anymore. The default value is True.

Used for TCustomMSDataSet to set the Required property of TField objects for the NOT NULL fields.

Class

TMSDataSetOptions

Syntax

```
property RequiredFields: boolean default False;
```

Remarks

If True, TCustomMSDataSet sets the Required property of TField objects for the NOT NULL fields. It is useful when table has a trigger that updates the NOT NULL fields. The default value is False.

Used to return the new values of fields to dataset after insert or update.

Class

TMSDataSetOptions

Syntax

```
property ReturnParams: boolean;
```

Remarks

Use the ReturnParams property to return the new values of fields to dataset after insert or update. Actual value of a 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 corresponding fields. The default value is False.

Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.

Class

TMSDataSetOptions

Syntax

```
property StrictUpdate: boolean;
```

Remarks

TCustomDADataSet raises an exception when the number of updated or deleted records is not equal 1. Setting this option also causes an exception if the RefreshRecord procedure returns more than one record. Does not affect TCustomMSDataSet.CursorType if CursorUpdate is True. The default value is True. In order for this option to work correctly, the SQL Server NOCOUNT option should be OFF (this is the default value). If NOCOUNT is ON, SQL Server returns 0 instead of the actual affected rows count. SDAC does not care for this option itself in order to avoid additional round trips to server.
We do not recommend using the StrictUpdate option with tables on which a trigger is defined, because this will cause problems if there are commands that modify data in the trigger. But if you need to use this combination, you should call the SET NOCOUNT ON command at the very beginning of the trigger to suppress sending affected rows count for SQL statements executed within the trigger.

Used to specify whether to discard all trailing spaces in the fixed-length string fields of a dataset.

Class

TMSDataSetOptions

Syntax

```
property TrimFixedChar: boolean;
```

Remarks

Use the TrimFixedChar property to specify whether to discard all trailing spaces in the fixed-length string fields of a dataset. The default value is True.

Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

Class

TMSDataSetOptions

Syntax

```
property TrimVarChar: boolean;
```

Remarks

Use the TrimVarChar property to specify whether to discard all trailing spaces in the variable-length string fields of a dataset. The default value is False.

Used to specify whether to query additional keyfields from the server.

Class

TMSDataSetOptions

Syntax

```
property UniqueRecords: boolean default False;
```

Remarks

Use the UniqueRecords property to specify whether to query additional keyfields from the server. If UniqueRecords is False, keyfields aren't queried from the server when they are not included in the query explicitly. For example, the result of the query execution "SELECT ShipName FROM Orders" holds the only field - ShipName. When used with the TCustomDADataSet.ReadOnly property set to True, the

UniqueRecords option gives insignificant advantage of performance. But in this case SQLRefresh will be generated in simplified way. If UniqueRecord is True, keyfields needed for the complete automatic generation of SQLInsert, SQLUpdate, SQLDelete or SQLRefresh statements are queried from the server implicitly. For example, the result of query execution "SELECT ShipName FROM Orders" holds at least two fields - ShipName and OrderID. The default value is False. Has effect only for the TCustomMSDataSet.CursorType cursor.

17.14.1.10 MSAccess.TMSDataSource Class

TMSDataSource provides an interface between a SDAC dataset components and data-aware controls on a form.
For a list of all members of this type, see TMSDataSource members.

Unit

MSAccess

Syntax

```
TMSDataSource = class(TCRDataSource);
```

Remarks

TMSDataSource provides an interface between a SDAC dataset components and data-aware controls on a form.
TMSDataSource 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
    TMSDataSource

TMSDataSource class overview.

17.14.1.11 MSAccess.TMSEncryptor Class

The class that performs encrypting and decrypting of data.
For a list of all members of this type, see TMSEncryptor members.

Unit

MSAccess

Syntax

```
TMSEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

TObject
  TCREncryptor
    TMSEncryptor

TMSEncryptor 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.

17.14.1.12 MSAccess.TMSFileStream Class

A class for managing FILESTREAM data using Win32 API.
For a list of all members of this type, see TMSFileStream members.

Unit

MSAccess

Syntax

```
TMSFileStream = class(TStream);
```

Remarks

Use the TMSFileStream class to manage FILESTREAM data using Win32 API. This class inherits almost all its functionality from the TStream class, except the Close method. It's necessary to call the Close method before the transaction commits or rolls back FILESTREAM data.

Inheritance Hierarchy

TObject
  TMSFileStream

See Also

- TCustomMSDataSet.GetFileStreamForField

TMSFileStream class overview.

Methods

Name	Description
Close	Used to close an opened file handle associated with FILESTREAM data.
Flush	Used to write all buffered data to the file associated with FILESTREAM data.

Methods of the **TMSFileStream** class.
For a complete list of the **TMSFileStream** class members, see the TMSFileStream Members topic.

Public

Name	Description
Close	Used to close an opened file handle associated with FILESTREAM data.
Flush	Used to write all buffered data to the file associated with FILESTREAM data.

See Also

- TMSFileStream Class
- TMSFileStream Class Members

Used to close an opened file handle associated with FILESTREAM data.

Class

TMSFileStream

Syntax

```
procedure Close;
```

Remarks

Closes an opened file handle associated with FILESTREAM data. It's necessary to call this method before the transaction commits or rolls back FILESTREAM data. Failing to close the handle will cause server-side resource leaks.

Used to write all buffered data to the file associated with FILESTREAM data.

Class

TMSFileStream

Syntax

```
procedure Flush;
```

Remarks

Writes all buffered data to the file associated with FILESTREAM data. To use this method, you should create TMSFileStream with access rights for writing.

17.14.1.13 MSAccess.TMSMetadata Class

A component for obtaining metainformation about database objects from the server.
For a list of all members of this type, see TMSMetadata members.

Unit

MSAccess

Syntax

```
TMSMetadata = class(TCustomMSDataSet);
```

Remarks

The TMSMetaData component is used to obtain metainformation from the server about objects in the database, such as tables, table columns, stored procedures, etc in the form of a table. TMSMetaData publishes properties of TDAMetaData.
To get the information you are interested in, you should initially select the proper object type in the

TMSMetadata.ObjectType property. After that you may open TMSMetadata and view the result like in usual dataset (in the DB-aware controls or from code). This dataset may be too big for viewing because information about all objects of the specified type is shown. To get the information only about objects you are interested in, you should specify appropriate filters in properties like DatabaseName, SchemaName, TableName, etc. To ascertain which properties are applicable to the selected object type, refer to the table given in the description of the ObjectType property.

Example

Here is a small example demonstrating obtaining information about default column values of a table.

```
procedure TForm.ButtonClick(Sender: TObject);
var
 FieldNameCol, FieldDefCol: TField;
  DefValue: string;
begin
  MSMetadata.ObjectType := otColumns;
  MSMetadata.DatabaseName := EditDatabaseName.Text;
  MSMetadata.TableName := EditTableName.Text;
  MSMetadata.Open;
  FieldNameCol := MSMetadata.FieldByName('COLUMN_NAME');
  FieldDefCol := MSMetadata.FieldByName('COLUMN_DEFAULT');
  Memo.Lines.Clear;
  if MSMetadata.RecordCount = 0 then
    Memo.Lines.Add('Specified object not found')
  else
    while not MSMetadata.Eof do begin
      if FieldDefCol.IsNull then
        DefValue := '> Not defined <'
      else
        DefValue := FieldDefCol.AsString;
      Memo.Lines.Add(Format('Field Name: %s;    Default Value: %s', [Fiel
      MSMetadata.Next;
    end;
end;
```

Inheritance Hierarchy

TObject
  TMemDataSet
    TCustomDADataSet
      TCustomMSDataSet
        TMSMetadata

See Also

- TCustomDADataSet.Debug
- TCustomDASQL.Debug
- DBMonitor
- TCustomMSDataSet
- TDAMetaData

TMSMetadata class overview.

Properties

Name	Description
AssemblyID	Used to specify the assembly ID that constitutes object type descriptor used to retrieve metadata information from the server.
AssemblyName	Used to specify the assembly name that constitutes object type descriptor used to retrieve metadata information from the server.
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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
ColumnName	Used to specify the column name that constitutes object type descriptor used to retrieve metadata information from the server.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used to connect to a data store.
ConstraintName	Used to specify the constraint name that constitutes object type descriptor used to retrieve metadata information from the server.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
DatabaseName	Used to specify the database name that constitutes object type descriptor used to retrieve metadata information from the server.
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.
Encryption (inherited from **TCustomDADataSet**)	Used to specify the options of the data encryption in a dataset.
FetchAll (inherited from **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.

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.
IndexName	Used to specify the index name that constitutes object type descriptor used to retrieve metadata information from the server.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
LinkedServer	Used to specify the server name that constitutes object type descriptor used to retrieve metadata information from the server.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
ObjectType	Used to specify the object type metadata information will be requested from the server.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
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 **TCustomMSDataSet**)	Contains 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.
ReferencedAssemblyID	Used to specify the referenced assembly ID that constitutes object type descriptor used to retrieve metadata information from the server.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SchemaCollectionName	Used to specify the XML schema collection name that constitutes object type descriptor used to retrieve metadata information from the server.
SchemaName	Used to specify the schema name that constitutes object type descriptor used to retrieve metadata information from the server.
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.
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 stored procedure name that constitutes object type descriptor used to retrieve metadata information from the server.
TableName	Used to specify the table name that constitutes object type descriptor used to retrieve metadata information from the server.
TargetNamespaceURI	Used to specify the XML schema collection name that constitutes object type descriptor used to retrieve metadata information from the server.

UDTName	Used to specify the User-Defined Type name that constitutes object type descriptor used to retrieve metadata information from the server.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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 **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures 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.
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 **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.
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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
MacroByName (inherited from **TCustomDADataSet**)	Finds a Macro with the name passed in Name.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
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.

Properties of the **TMSMetadata** class.
For a complete list of the **TMSMetadata** class members, see the TMSMetadata 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
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 **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.

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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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
AssemblyID	Used to specify the assembly ID that constitutes object type descriptor used to retrieve metadata information from the server.
AssemblyName	Used to specify the assembly name that constitutes object type descriptor used to retrieve metadata information from the server.
ColumnName	Used to specify the column name that constitutes object type descriptor used to retrieve metadata information from the server.
ConstraintName	Used to specify the constraint name that constitutes object type descriptor used to retrieve metadata information from the server.
DatabaseName	Used to specify the database name that constitutes object type descriptor used to retrieve metadata information from the server.

IndexName	Used to specify the index name that constitutes object type descriptor used to retrieve metadata information from the server.
LinkedServer	Used to specify the server name that constitutes object type descriptor used to retrieve metadata information from the server.
ObjectType	Used to specify the object type metadata information will be requested from the server.
ReferencedAssemblyID	Used to specify the referenced assembly ID that constitutes object type descriptor used to retrieve metadata information from the server.
SchemaCollectionName	Used to specify the XML schema collection name that constitutes object type descriptor used to retrieve metadata information from the server.
SchemaName	Used to specify the schema name that constitutes object type descriptor used to retrieve metadata information from the server.
StoredProcName	Used to specify the stored procedure name that constitutes object type descriptor used to retrieve metadata information from the server.
TableName	Used to specify the table name that constitutes object type descriptor used to retrieve metadata information from the server.
TargetNamespaceURI	Used to specify the XML schema collection name that constitutes object type descriptor used to retrieve metadata information from the server.
UDTName	Used to specify the User-Defined Type name that constitutes object type descriptor used to retrieve metadata information from the server.

See Also

- TMSMetadata Class
- TMSMetadata Class Members

Used to specify the assembly ID that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property AssemblyID: integer default 0;
```

Remarks

Use the AssemblyID property to specify the ID of the assembly which together with AssemblyName, schema, database and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- AssemblyName
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the assembly name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property AssemblyName: string;
```

Remarks

Use the AssemblyName property to specify the name of the assembly which together with AssemblyID, schema, database and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- AssemblyID
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the column name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property ColumnName: string;
```

Remarks

Use the ColumnName property to specify the column name which together with table, schema and database names constitutes object type descriptor that is used to retrieve metadata information from

the server.
Refer to ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- TableName

Used to specify the constraint name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property ConstraintName: string;
```

Remarks

Use the ConstraintName property to specify the constraint name which together with table, schema and database names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- TableName

Used to specify the database name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property DatabaseName: string;
```

Remarks

Use the DatabaseName property to specify the database name which together with table, schema and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- SchemaName

- TableName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the index name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

`TMSMetadata`

Syntax

```
property IndexName: string;
```

Remarks

Use the IndexName property to specify the index name which together with table, schema and database names constitutes object type descriptor that is used to retrieve metadata information from the server. Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- TableName

Used to specify the server name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

`TMSMetadata`

Syntax

```
property LinkedServer: string;
```

Remarks

Use the LinkedServer property to specify the name of the server which together with other optional names constitutes object type descriptor that is used to retrieve metadata information from the server. Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the object type metadata information will be requested from the server.

Class

`TMSMetadata`

Syntax

```
property ObjectType: TMSObjectType default otDatabases;
```

Remarks

Use the ObjectType property to specify the object type which metadata will be requested from the server.
The following table lists the names of applicable restriction properties for each object type and also equivalent schema rowset name as it's described in Microsoft's MSDN OLE DB Library (see oledb.chm file for the in-depth information on each object type).

See Also

- DatabaseName
- SchemaName
- TableName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the referenced assembly ID that constitutes object type descriptor used to retrieve metadata information from the server.

Class

`TMSMetadata`

Syntax

```
property ReferencedAssemblyID: integer default 0;
```

Remarks

Use the ReferencedAssemblyID property to specify the ID of the referenced assembly which together with AssemblyName, AssemblyID, schema, database and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- AssemblyName
- AssemblyID
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the XML schema collection name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property SchemaCollectionName: string;
```

Remarks

Use the SchemaCollectionName property to specify the name of the XML schema collection which together with schema and database names and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName
- TargetNamespaceURI

Used to specify the schema name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property SchemaName: string;
```

Remarks

Use the SchemaName property to specify the schema name which together with table, database and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- TableName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the stored procedure name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property StoredProcName: string;
```

Remarks

Use the StoredProcName property to specify the stored procedure name which together with table, schema and database names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- TableName

Used to specify the table name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to specify the table name which together with schema, database and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

Used to specify the XML schema collection name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property TargetNamespaceURI: string;
```

Remarks

Use the TargetNamespaceURI property to specify the name of the XML schema collection which together with schema and database names and/or other optional names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName
- SchemaCollectionName

Used to specify the User-Defined Type name that constitutes object type descriptor used to retrieve metadata information from the server.

Class

TMSMetadata

Syntax

```
property UDTName: string;
```

Remarks

Use the UDTName property to specify the name of the User-Defined Type which together with schema and database names constitutes object type descriptor that is used to retrieve metadata information from the server.
Refer to the ObjectType property to get the complete listing of all object types to which this property is applicable. In all other cases this property is merely ignored.

See Also

- ObjectType
- DatabaseName
- SchemaName
- StoredProcName
- ColumnName
- IndexName
- ConstraintName

17.14.1.14 MSAccess.TMSParam 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 TMSParam members.

Unit

MSAccess

Syntax

```
TMSParam = class(TDAParam);
```

Remarks

Use the properties of TMSParam to set the value of a parameter. Objects that use parameters create TMSParam objects to represent these parameters. For example, TMSParam objects are used by TMSSQL, TCustomMSDataSet.
TMSParam 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 TMSParam object. Conversely, TMSParam includes properties that indicate how the field value is passed as a parameter.

Inheritance Hierarchy

TObject
  TDAParam
    TMSParam

TMSParam class overview.

Properties

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.
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 assign a recordset to the Table-Valued Parameter.
AsWideString (inherited from **TDAParam**)	Used to assign the Unicode string value to the parameter.
DataType (inherited from **TDAParam**)	Indicates the data type of the parameter.
IsNull (inherited from **TDAParam**)	Used to indicate whether the value assigned to a parameter is NULL.
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.
TableTypeName	Used to indicate the table type name of a Table-Valued Parameter.
Value (inherited from **TDAParam**)	Used to represent the value of the parameter as Variant.

Methods

Name	Description
AssignField (inherited from **TDAParam**)	Assigns field name and field value to a param.
AssignFieldValue (inherited from **TDAParam**)	Assigns the specified field properties and value to a parameter.
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.

Properties of the **TMSParam** class.
For a complete list of the **TMSParam** class members, see the TMSParam 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.
AsTable	Used to assign a recordset to the Table-Valued 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.
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.

TableTypeName	Used to indicate the table type name of a Table-Valued Parameter.

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.
Si e (inherited from **TDAParam**)	Specifies the si e of a string type parameter.
Value (inherited from **TDAParam**)	Used to represent the value of the parameter as Variant.

See Also

- TMSParam Class
- TMSParam Class Members

Used to assign a recordset to the Table-Valued Parameter.

Class

`TMSParam`

Syntax

```
property AsTable: TMSTableObject;
```

Remarks

Use the AsTable property to assign a recordset to the Table-Valued Parameter. Setting AsTable will set the DataType property to ftDataSet.

Used to indicate the table type name of a Table-Valued Parameter.

Class

`TMSParam`

Syntax

```
property TableTypeName: string;
```

Remarks

Use the TableTypeName property to determine the table type name of a Table-Valued Parameter.

17.14.1.15 MSAccess.TMSParams Class

Used to control TMSParam objects.
For a list of all members of this type, see TMSParams members.

Unit

`MSAccess`

Syntax

```
TMSParams = class(TDAParams);
```

Remarks

Use TMSParams to manage a list of TMSParam objects for an object that uses field parameters. For example, TMSStoredProc objects and TMSQuery objects use TMSParams objects to create and access their parameters.

Inheritance Hierarchy

TObject
  TDAParams
    TMSParams

See Also

- TMSParam
- TCustomDASQL.Params
- TCustomDADataSet.Params

TMSParams class overview.

Properties

Name	Description
Items (inherited from **TDAParams**)	Used to interate through all 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.

17.14.1.16 MSAccess.TMSQuery 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 TMSQuery members.

Unit

MSAccess

Syntax

```
TMSQuery = class(TCustomMSDataSet);
```

Remarks

TMSQuery is a direct descendant of the TCustomMSDataSet component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.
Use TMSQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TMSQuery 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, you can specify KeyFields. If they are not specified, TMSQuery will retrieve primary keys for UpdatingTable from metadata. TMSQuery 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.

Inheritance Hierarchy

TObject
  TMemDataSet
    TCustomDADataSet
      TCustomMSDataSet
        TMSQuery

See Also

- Query demo project
- Updating Data with SDAC Dataset Components
- Master/Detail Relationships
- TMSStoredProc
- TMSTable

TMSQuery 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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used to connect to a data store.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
Encryption (inherited from **TCustomDADataSet**)	Used to specify the options of the data encryption in a dataset.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.

FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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.
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.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
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.
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.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.
UpdatingTable	Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures 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.

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 **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.
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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
MacroByName (inherited from **TCustomDADataSet**)	Finds a Macro with the name passed in Name.

OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.

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.

Properties of the **TMSQuery** class.
For a complete list of the **TMSQuery** class members, see the TMSQuery 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.

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.
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 **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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.

OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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

- TMSQuery Class
- TMSQuery Class Members

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TMSQuery

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.

Used to specify what kind of lock will be performed when editing a record.

Class

TMSQuery

Syntax

```
property LockMode: TLockMode;
```

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

- TMSStoredProc.LockMode
- TMSTable.LockMode

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

TMSQuery

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 TCustomMSDataSet.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.
All fields from other than target table have their ReadOnly properties set to True (if TCustomMSDataSet.Options)
With TCustomMSDataSet.CursorType UpdatingTable can be used only if TCustomMSDataSet.Options = False.

17.14.1.17 MSAccess.TMSSQL 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 TMSSQL members.

Unit

MSAccess

Syntax

```
TMSSQL = class(TCustomDASQL);
```

Remarks

The TMSSQL component is a direct descendant of the TCustomDASQL class.
Use The TMSSQL 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
    TMSSQL

See Also

- TMSQuery

TMSSQL class overview.

Properties

Name	Description
ChangeCursor (inherited from **TCustomDASQL**)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
CommandTimeout	Used to specify the amount of time before an attempt to execute a command is considered unsuccessful.
Connection	Used to specify a connection object that will be used to connect to a data store.
Debug (inherited from **TCustomDASQL**)	Used to display executing statement, all its parameters' values, and the type of parameters.
DescribeParams	Used to specify whether to query TMSParam properties from the server when executing the TCustomDASQL.Prepare method.
FinalSQL (inherited from **TCustomDASQL**)	Used to return a SQL statement with expanded macros.
MacroCount (inherited from **TCustomDASQL**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDASQL**)	Makes it possible to change SQL queries easily.
NonBlocking	Used to execute a SQL statement in a separate thread.

ParamCheck (inherited from **TCustomDASQL**)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount (inherited from **TCustomDASQL**)	Indicates the number of parameters in the Params property.
Params	Contains parameters for a query's SQL statement.
ParamValues (inherited from **TCustomDASQL**)	Used to get or set the values of individual field parameters that are identified by name.
PermitPrepare	This option is not supported any more.
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.

Methods

Name	Description
BreakExec	Breaks execution of a SQL satatement on the server.
Execute (inherited from **TCustomDASQL**)	Overloaded. Executes SQL commands.
ExecuteForXML	Overloaded. Provides data in readable view for the SELECT statements written using the FOR XML clause.
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	Determines if a parameter with the specified name exists in a dataset.
MacroByName (inherited from **TCustomDASQL**)	Finds a Macro with the name passed in Name.
ParamByName	Sets or uses parameter information for a specific parameter based on its 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.

Properties of the **TMSSQL** class.
For a complete list of the **TMSSQL** class members, see the TMSSQL Members topic.

Public

Name	Description
AfterExecute (inherited from **TCustomDASQL**)	Occurs after a SQL statement has been executed.
ChangeCursor (inherited from **TCustomDASQL**)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
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.
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.

Published

Name	Description
CommandTimeout	Used to specify the amount of time before an attempt to execute a command is considered unsuccessful.
Connection	Used to specify a connection object that will be used to connect to a data store.
DescribeParams	Used to specify whether to query TMSParam properties from the server when executing the TCustomDASQL.Prepare method.
NonBlocking	Used to execute a SQL statement in a separate thread.
Params	Contains parameters for a query's SQL statement.
PermitPrepare	This option is not supported any more.

See Also

- TMSSQL Class
- TMSSQL Class Members

Used to specify the amount of time before an attempt to execute a command is considered unsuccessful.

Class

TMSSQL

Syntax

```
property CommandTimeout: integer default 0;
```

Remarks

Use the CommandTimeout property to specify the amount of time before an attempt to execute a command is considered unsuccessful. Measured in seconds.
If a command is successfully executed before the specified time expires, CommandTimeout has no effect.
The default value is 0 (infinite).

See Also

- TMSConnection.ConnectionTimeout

Used to specify a connection object that will be used to connect to a data store.

Class

TMSSQL

Syntax

```
property Connection: TCustomMSConnection;
```

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 TCustomMSConnection descendant objects.
At runtime, set the Connection property to reference an existing object of a TCustomMSConnection descendant.

See Also

- TCustomMSConnection

Used to specify whether to query TMSParam properties from the server when executing the TCustomDASQL.Prepare method.

Class

TMSSQL

Syntax

```
property DescribeParams: boolean default False;
```

Remarks

Specifies whether to query TMSParam properties (Name, ParamType, DataType, Si e, TableTypeName) from the server when executing the TCustomDASQL.Prepare method. The default value is False.

Used to execute a SQL statement in a separate thread.

Class

TMSSQL

Syntax

```
property NonBlocking: boolean default False;
```

Remarks

Set the NonBlocking option to True to execute a SQL statement in a separate thread.

Contains parameters for a query's SQL statement.

Class

TMSSQL

Syntax

```
property Params: TMSParams stored False;
```

Remarks

Contains 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 ero-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

- TMSParam
- ParamByName

This option is not supported any more.

Class

TMSSQL

Syntax

```
property PermitPrepare: boolean stored False;
```

Remarks

This option is out of date and isn't supported any more. The default behavior is the same to that when PermitPrepare is set to False. To get the same behaviour as when PermitPrepare is set to True, you need to execute the TCustomDASQL.Prepare method explicitly.

Methods of the **TMSSQL** class.
For a complete list of the **TMSSQL** class members, see the TMSSQL Members topic.

Public

Name	Description
AfterExecute (inherited from **TCustomDASQL**)	Occurs after a SQL statement has been executed.
BreakExec	Breaks execution of a SQL satatement 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.
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.
ExecuteForXML	Overloaded. Provides data in readable view for the SELECT statements written using the FOR XML clause.
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	Determines if a parameter with the specified name exists in a dataset.
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	Sets or uses parameter information for a specific parameter based on its 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

- TMSSQL Class
- TMSSQL Class Members

Breaks execution of a SQL satatement on the server.

Class

TMSSQL

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
- TMSConnection.OnInfoMessage
- TCustomMSDataSet.BreakExec

Provides data in readable view for the SELECT statements written using the FOR XML clause.

Class

TMSSQL

Overload List

Name	Description

ExecuteForXML(Stream: TStream; OutputEncoding: TOLEDBOutputEncoding)	Provides data in readable view for the SELECT statements written using the FOR XML clause.
ExecuteForXML(**out** XML: **string**)	Provides data in readable view for the SELECT statements written using the FOR XML clause.

Provides data in readable view for the SELECT statements written using the FOR XML clause.

Class

TMSSQL

Syntax

```
procedure ExecuteForXML(Stream: TStream; OutputEncoding:
 TOLEDBOutputEncoding); overload
```

Parameters

Stream
Is an output parameter where the result of the query execution is written.

OutputEncoding
Holds the encoding in which the result will be returned.

Determines if a parameter with the specified name exists in a dataset.

Class

TMSSQL

Syntax

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

Parameters

Value
holds the name of the param for which to search.

Return Value

the TMSParam object for the specified Name. If a TMSParam object with matching name was not found, returns nil.

Remarks

Call the FindParam method to determine if a 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 TMSParam object for the specified Name. Otherwise it returns nil.

See Also

- Params
- ParamByName

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

Class

TMSSQL

Syntax

```
function ParamByName(const Value: string): TMSParam;
```

Parameters

Value
Holds the Parameter value.

Return Value

the parameter, if a match was found. Otherwise, an exception is raised.

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 the parameter's value at runtime and returns a TMSParam object.

Example

For 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

- TMSParam
- Params
- FindParam

17.14.1.18 MSAccess.TMSStoredProc Class

A component for accessing and executing stored procedures and functions.
For a list of all members of this type, see TMSStoredProc members.

Unit

`MSAccess`

Syntax

```
TMSStoredProc = class(TCustomMSStoredProc);
```

Remarks

Use TMSStoredProc 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 return parameters in the Params property.

Inheritance Hierarchy

TObject
  TMemDataSet
    TCustomDADataSet
      TCustomMSDataSet
        TCustomMSStoredProc
          TMSStoredProc

See Also

- Stored proc update demo
- TMSQuery
- TMSSQL
- Updating Data with SDAC Dataset Components

TMSStoredProc 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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used to connect to a data store.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
Encryption (inherited from **TCustomDADataSet**)	Used to specify the options of the data encryption in a dataset.
FetchAll (inherited from **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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.
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.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
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 **TCustomMSDataSet**)	Contains 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.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
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.
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.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.
UpdatingTable (inherited from **TCustomMSStoredProc**)	Specifies 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 **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures 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.
ExecProc (inherited from **TCustomMSStoredProc**)	Executes SQL statements 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 **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.
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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
MacroByName (inherited from **TCustomDADataSet**)	Finds a Macro with the name passed in Name.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.

PrepareSQL (inherited from **TCustomMSStoredProc**)	Builds a query for TCustomMSStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.

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.

Properties of the **TMSStoredProc** class.
For a complete list of the **TMSStoredProc** class members, see the TMSStoredProc 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.

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.
ExecProc (inherited from **TCustomMSStoredProc**)	Executes SQL statements 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.
FetchAll (inherited from **TCustomMSDataSet**)	Used to decrease the time of retrieving additional records to the client side when calling TMemDataSet.Locate and TMemDataSet.LocateEx for the first time.
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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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 (inherited from **TCustomMSStoredProc**)	Builds a query for TCustomMSStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.

SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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.
UpdatingTable (inherited from **TCustomMSStoredProc**)	Specifies 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.

Published

Name	Description

LockMode	Used to specify what kind of lock will be performed when editing a record.
StoredProcName	Used to specify the name of the stored procedure to call on the server.

See Also

- TMSStoredProc Class
- TMSStoredProc Class Members

Used to specify what kind of lock will be performed when editing a record.

Class

TMSStoredProc

Syntax

```
property LockMode: TLockMode;
```

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

- TMSQuery.LockMode
- TMSTable.LockMode

Used to specify the name of the stored procedure to call on the server.

Class

TMSStoredProc

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.

17.14.1.19 MSAccess.TMSTable 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 TMSTable members.

Unit

MSAccess

Syntax

```
TMSTable = class(TCustomMSTable);
```

Remarks

The TMSTable component allows retrieving and updating data in a single table without writing SQL statements. Use TMSTable to access data in a table or view. Use the TableName property to specify table name. TMSTable 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.

Inheritance Hierarchy

TObject
- TMemDataSet
  - TCustomDADataSet
    - TCustomMSDataSet
      - TCustomMSTable
        - **TMSTable**

See Also

- Updating Data with SDAC Dataset Components
- Master/Detail Relationships
- Performance of Obtaining Data
- TCustomMSDataSet
- TMSQuery
- TCustomMSTable

TMSTable 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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used to connect to a data store.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
Encryption (inherited from **TCustomDADataSet**)	Used to specify the options of the data encryption in a dataset.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.

FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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.
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.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
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 **TCustomMSDataSet**)	Contains 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.

RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
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.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
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 **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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 **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures 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.
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 **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.
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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
MacroByName (inherited from **TCustomDADataSet**)	Finds a Macro with the name passed in Name.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL (inherited from **TCustomMSTable**)	Determines KeyFields and build query of TCustomMSTable.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.

AfterUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
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.

Properties of the **TMSTable** class.
For a complete list of the **TMSTable** class members, see the TMSTable 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 **TCustomMSDataSet**)	Occurs after executing insert, delete, update, lock and refresh operation.
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 **TCustomMSDataSet**)	Occurs before executing insert, delete, update, lock and refresh operation.
BreakExec (inherited from **TCustomMSDataSet**)	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 **TCustomMSDataSet**)	Points to a TMSChangeNotification component.
CommandTimeout (inherited from **TCustomMSDataSet**)	Used to specify the amount of time to attempt to execute a command.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
Connection (inherited from **TCustomMSDataSet**)	Used to specify a connection object that will be used 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 **TCustomMSDataSet**)	Serves for the creating of a stored procedures call.
CursorType (inherited from **TCustomMSDataSet**)	Cursor types supported by SQL Server.
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.
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 **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**)	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 **TCustomMSDataSet**)	Indicates whether 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 (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
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.
GetFileStreamForField (inherited from **TCustomMSDataSet**)	Used to create the TMSFileStream object for working with FILESTREAM data.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
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.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyFields (inherited from **TCustomDADataSet**)	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 **TCustomMSDataSet**)	Overloaded. Locks the current records to prevent multiple users' access to it.
LockTable (inherited from **TCustomMSDataSet**)	Locks a table to prevent multiple access to it.
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.
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.
OpenNext (inherited from **TCustomMSDataSet**)	Opens next rowset in the statement.
Options (inherited from **TCustomMSDataSet**)	Used to specify the behaviour of a TCustomMSDataSet object.
ParamByName (inherited from **TCustomMSDataSet**)	Provides access to a parameter by 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 **TCustomMSDataSet**)	Contains 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 (inherited from **TCustomMSTable**)	Determines KeyFields and build query of TCustomMSTable.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
RefreshQuick (inherited from **TCustomMSDataSet**)	An optimized procedure to retrieve the changes applied to the server by other clients to the particular client side.
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.
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.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.

SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
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.
UpdateObject (inherited from **TCustomMSDataSet**)	Used to point to an update object component which provides SQL statements that perform updates of 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.

OrderFields	Used to build ORDER BY clause of SQL statements.
TableName	Used to specify the name of the database table this component encapsulates.

See Also

- TMSTable Class
- TMSTable Class Members

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TMSTable

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.

Used to specify what kind of lock will be performed when editing a record.

Class

TMSTable

Syntax

```
property LockMode: TLockMode;
```

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

- TMSStoredProc.LockMode
- TMSQuery.LockMode

Used to build ORDER BY clause of SQL statements.

Class

TMSTable

Syntax

```
property OrderFields: string;
```

Remarks

TMSTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas.
TMSTable is reopened when OrderFields is being changed.

See Also

- TMSTable

Used to specify the name of the database table this component encapsulates.

Class

TMSTable

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 at design time,select a valid table name from the TableName drop-down list in Object Inspector.

17.14.1.20 MSAccess.TMSTableData Class

A component for working with user-defined table types in SQL Server 2008.
For a list of all members of this type, see TMSTableData members.

Unit

MSAccess

Syntax

```
TMSTableData = class(TMemDataSet);
```

Remarks

The TMSTableData allows working with table types in SQL Server 2008. Use the TMSTableData.TableTypeName property to specify the table type.
When adding, changing, and deleting operations in dataset, data are stored in an internal cache on the client side. Data are sent to the server only as data of Table-Valued Parameters when a stored procedure is executed. To assign dataset contents to a parameter use the Table property.

Inheritance Hierarchy

TObject
  TMemDataSet
    TMSTableData

See Also

- Using Table-Valued Parameters

TMSTableData class overview.

Properties

Name	Description

CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Connection	Specifies a connection object that will be used 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.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Table	Used for assigning data from TMSTableData to a stored procedure parameter.
TableTypeName	Specifies the name of user-define table type to work with.
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.

Properties of the **TMSTableData** class.
For a complete list of the **TMSTableData** class members, see the TMSTableData 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.
Table	Used for assigning data from TMSTableData to a stored procedure parameter.
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
Connection	Specifies a connection object that will be used to connect to a data store.
TableTypeName	Specifies the name of user-define table type to work with.

See Also

- TMSTableData Class
- TMSTableData Class Members

Specifies a connection object that will be used to connect to a data store.

Class

TMSTableData

Syntax

```
property Connection: TCustomMSConnection;
```

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 TCustomMSConnection or its descendant class objects.
At runtime, set the Connection property to reference an existing TCustomMSConnection object.

See Also

- TCustomMSConnection

Used for assigning data from TMSTableData to a stored procedure parameter.

Class

TMSTableData

Syntax

```
property Table: TMSTableObject;
```

Remarks

Use the Table property to assign data from TMSTableData to a stored procedure parameter.

Example

```

MSStoredProc.ParamByName('TVP').AsTable := MSTableData.Table;
```

Specifies the name of user-define table type to work with.

Class

TMSTableData

Syntax

```
property TableTypeName: string;
```

Remarks

Use the TableTypeName property to specify the name of user-define table type to work with.
If Connection is assigned at design time, you can select a valid table type name from the TableTypeName drop-down list in Object Inspector.

17.14.1.21 MSAccess.TMSUDTField Class

A field class providing native access to the CLR User-defined Types (UDT) fields of SQL Server 2005.
For a list of all members of this type, see TMSUDTField members.

Unit

MSAccess

Syntax

```
TMSUDTField = class(TBlobField);
```

Remarks

This field class is designed to provide native access to the CLR User-defined Types (UDT) fields of SQL Server 2005. UDT fields are mapped to TMSUDTField only if SQL Native Client is used as TMSConnection.Options. Otherwise, UDT fields are mapped to TVarBytesField.
You can get information about the underlying UDT using TMSUDTField.AssemblyTypename, TMSUDTField.UDTCatalogname, TMSUDTField.UDTName, TMSUDTField.UDTSchemaname. Extended abilities of UDT fields are accessible in Win32 applications through the TMSUDTField.AsUDT property.
Note: The CLR integration is disabled in SQL Server 2005 by default, so you should enable it to work with UDT. http://msdn2.microsoft.com/library/ms254506(VS.80).aspx of MSDN describes how to enable

CLR support for SQL Server.

Inheritance Hierarchy

TObject
TMSUDTField

See Also

- Working with User Defined Types (UDT)
- TMSUDTField.AsUDT

TMSUDTField class overview.

Properties

Name	Description
AssemblyTypename	Used to indicate the type name prefixed by namespace.
AsUDT	Used to access properties and methods of CLR User-defined Types (UDT) from the Win32 applications.
UDTCatalogname	Used to indicate the name of the catalog where UDT is defined.
UDTName	Used to indicate the name of the assembly containing the UDT class.
UDTSchemaname	Used to indicate the name of the schema where UDT is defined.

Properties of the **TMSUDTField** class.
For a complete list of the **TMSUDTField** class members, see the TMSUDTField Members topic.

Public

Name	Description
AssemblyTypename	Used to indicate the type name prefixed by namespace.
AsUDT	Used to access properties and methods of CLR User-defined Types (UDT) from the Win32 applications.
UDTCatalogname	Used to indicate the name of the catalog where UDT is defined.
UDTName	Used to indicate the name of the assembly containing the UDT class.
UDTSchemaname	Used to indicate the name of the schema where UDT is defined.

See Also

- TMSUDTField Class
- TMSUDTField Class Members

Used to indicate the type name prefixed by namespace.

Class

`TMSUDTField`

Syntax

```
property AssemblyTypename: string;
```

Remarks

Indicates the type name prefixed by namespace if applicable.
This property is read-only.

Used to access properties and methods of CLR User-defined Types (UDT) from the Win32 applications.

Class

TMSUDTField

Syntax

```
property AsUDT: Variant;
```

Remarks

The AsUDT property lets you access properties and methods of CLR User-defined Types (UDT) from your Win32 applications. In order to use this functionality, you should create a CLR assembly that implements your new UDT, register this assembly in SQL Server, and create a table containing a column based on your UDT. Also it is necessary to put the CLR assembly implementing your UDT into the directory with your application using SDAC, or register it in GAC.
After that you can open tables with UDT fields, cast these fields to TMSUDTField, and invoke members of your UDT through the AsUDT property.
For detailed information on what is UDT, how to create and use it, please refer to this topic of MSDN.
Note: if you use this functionality in your application, you will need to deploy the *Devart.Sdac.UDTProxy.dll* file along with it. This file should be present in the directory with your application, or registered in GAC. You will find this file in the Bin folder of your SDAC installation directory.
Note: This functionality has certain restrictions, like:

- .NET framework 2 or higher should be installed on the client computer;
- can be used only in Win32 applications;
- restrictions of User-defined Types itself, see here for more details.

Example

You can open tables with UDT fields, cast these fields to TMSUDTField, and invoke members of your UDT through the AsUDT property like it is shown in the code below.
This code is taken from the UDT demo of the SDAC General demo. Please refer to this demo for an example. Also the UDT demo includes an implementation of the test UDT named Square.

```
var
Square: variant;
begin
MSQuery.Edit;
Square := (MSQuery.FieldByName('c_square') as TMSUDTField).AsUDT;
Square.Move(StrToInt(edBaseX.Text), StrToInt(edBaseY.Text));
MSQuery.Post;
end;
```

Used to indicate the name of the catalog where UDT is defined.

Class

TMSUDTField

Syntax

```
property UDTCatalogname: string;
```

Remarks

Indicates the name of the catalog where UDT is defined.
This property is read-only.

Used to indicate the name of the assembly containing the UDT class.

Class

`TMSUDTField`

Syntax

```
property UDTName: string;
```

Remarks

Indicates the name of the assembly containing the UDT class.
This property is read-only.

Used to indicate the name of the schema where UDT is defined.

Class

`TMSUDTField`

Syntax

```
property UDTSchemaname: string;
```

Remarks

Indicates the name of the schema where UDT is defined.
This property is read-only.

17.14.1.22 MSAccess.TMSUpdateSQL Class

A component for tuning update operations for the DataSet component.
For a list of all members of this type, see TMSUpdateSQL members.

Unit

`MSAccess`

Syntax

```
TMSUpdateSQL = class(TCustomDAUpdateSQL);
```

Remarks

Use the TMSUpdateSQL 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
    TMSUpdateSQL

TMSUpdateSQL 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.

17.14.1.23 MSAccess.TMSXMLField Class

A class providing access to the SQL Server 2005 xml data type.
For a list of all members of this type, see TMSXMLField members.

Unit

MSAccess

Syntax

```
TMSXMLField = class(TWideMemoField);
```

Remarks

TMSXMLField provides access to the SQL Server 2005 xml data type.
The TMSXMLField.DataType property values equal to ftXML. You can access actual XML document using the AsString and TMSXMLField.XML properties.

Inheritance Hierarchy

TObject
  TMSXMLField

TMSXMLField class overview.

Properties

Name	Description
SchemaCollection	Contains information about typed XML column.
Typed	Indicates if an XML column is typed.
XML	Returns an XML document or a fragment of XML document as string.

Properties of the **TMSXMLField** class.
For a complete list of the **TMSXMLField** class members, see the TMSXMLField Members topic.

Public

Name	Description
SchemaCollection	Contains information about typed XML column.
Typed	Indicates if an XML column is typed.
XML	Returns an XML document or a fragment of XML document as string.

See Also

- TMSXMLField Class
- TMSXMLField Class Members

Contains information about typed XML column.

Class

TMSXMLField

Syntax

```
property SchemaCollection: TMSSchemaCollection;
```

Remarks

Contains the following XML schema information about a typed XML column.

Name	The name of a catalog in which an XML schema collection is defined. Empty for an untyped XML column.
CatalogName	The name of a schema in which an XML schema collection is defined. Empty for an untyped XML column.
SchemaName	The name of XML schema collection. Empty for an untyped XML column.

See Also

- Typed

Indicates if an XML column is typed.

Class

TMSXMLField

Syntax

```
property Typed: boolean;
```

Remarks

Indicates whether an XML column is typed. If XML column is typed, the SchemaCollection property is filled.

See Also

- SchemaCollection

Returns an XML document or a fragment of XML document as string.

Class

`TMSXMLField`

Syntax

```
property XML: string;
```

Remarks

Returns an XML document or a fragment of XML document as string.

17.14.2 Types

Types in the **MSAccess** unit.

Types

Name	Description
TMSChangeNotificationEvent	This type is used for the TMSChangeNotification.OnChange event.
TMSUpdateExecuteEvent	

17.14.2.1 MSAccess.TMSChangeNotificationEvent Procedure Reference

This type is used for the TMSChangeNotification.OnChange event.

Unit

MSAccess

Syntax

```
TMSChangeNotificationEvent = procedure (Sender: TObject; DataSet:
  TCustomMSDataSet; NotificationInfo: TMSNotificationInfo;
  NotificationSource: TMSNotificationSource; NotificationType:
  TMSNotificationType) of object;
```

Parameters

Sender
An object that raised the event.

DataSet

NotificationInfo

NotificationSource

NotificationType

17.14.2.2 MSAccess.TMSUpdateExecuteEvent Procedure Reference

Unit

MSAccess

Syntax

```
TMSUpdateExecuteEvent = procedure (Sender: TCustomMSDataSet;
  StatementTypes: TStatementTypes; Params: TMSParams) 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.

17.14.3 Enumerations

Enumerations in the **MSAccess** unit.

Enumerations

Name	Description
TIsolationLevel	Specifies the extent to which all outside transactions interfere with the subsequent transactions of the current connection.
TMSLockType	Specifies the parameters for locking the current record.
TMSNotificationInfo	Indicates the reason of the notification.
TMSNotificationSource	Indicates the source of notification.
TMSNotificationType	Indicates if this notification is generated because of change or by subscription.
TMSObjectType	Enumerates the object types supported by TMSMetadata.

17.14.3.1 MSAccess.TIsolationLevel Enumeration

Specifies the extent to which all outside transactions interfere with the subsequent transactions of the current connection.

Unit

MSAccess

Syntax

```
TIsolationLevel = (ilReadCommitted, ilReadUnCommitted,
 ilRepeatableRead, ilIsolated, ilSnapshot);
```

Values

Value	Meaning
ilIsolated	The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting additional processing on range locks. Used to put aside all undesired effects observed in the concurrent accesses to the same set of data, but may lead to a greater latency at times of a congested database environment.
ilReadCommitted	Sets isolation level at which transaction cannot see changes made by outside transactions until they are committed. Only dirty reads (changes made by uncommitted transactions) are eliminated by this state of the isolation level. The default value.
ilReadUnCommitted	The most unrestricted level of the transaction isolation. All types of data access interferences are possible. Mainly used for browsing database and to receive instant data with prospective changes.
ilRepeatableRead	Prevents concurrent transactions from modifying data in the current uncommitted transaction. This level eliminates dirty reads as well as nonrepeatable reads (repeatable reads of the same data in one transaction before and after outside transactions may have started and committed).
ilSnapshot	Uses row versioning. Provides transaction-level read consistency. A data snapshot is taken when the snapshot transaction starts, and remains consistent for the duration of a transaction.

17.14.3.2 MSAccess.TMSLockType Enumeration

Specifies the parameters for locking the current record.

Unit

MSAccess

Syntax

```
TMSLockType = (ltUpdate, ltExclusive);
```

Values

Value	Meaning
ltExclusive	The locked record can be neither read nor updated until the lock is released.
ltUpdate	The locked record can be read by others, but it cannot be updated until the lock is released.

17.14.3.3 MSAccess.TMSNotificationInfo Enumeration

Indicates the reason of the notification.

Unit

MSAccess

Syntax

```
TMSNotificationInfo = (niAlter, niDelete, niDrop, niError,
  niInsert, niInvalid, niIsolation, niOptions, niPreviousFire,
  niQuery, niResource, niRestart, niTemplateLimit, niTruncate,
  niUnknown, niUpdate);
```

Values

Value	Meaning
niAlter	One or more underlying server objects were modified.
niDelete	Data in one or more tables referenced in the underlying query was deleted by a DELETE statement.
niDrop	One or more underlying objects were dropped.
niError	An internal error occurred in SQL Server.
niInsert	Data in one or more tables referenced in the underlying query was changed by an INSERT statement.
niInvalid	A provided statement does not support notifications (INSERT, UPDATE, etc. statement). See this MSDN topic for the detailed information about supported statements.
niIsolation	The isolation mode is not valid for query notifications (for example, Snapshot).
niOptions	The connection options were not provided correctly.
niPreviousFire	A previous statement has caused query notifications to fire under the current transaction.
niQuery	A SELECT statement that does not correspond to restrictions was provided.
niResource	The notification subscription was removed as there may be not enough server resources.
niRestart	SQL Server was restarted.
niTemplateLimit	One or more tables used in a query reached the maximum number of allowed templates.
niTruncate	One or more tables used in the underlying query were truncated.
niUnknown	An option sent by the server was not recognized.
niUpdate	Data in one or more underlying tables was changed by an UPDATE statement.

17.14.3.4 MSAccess.TMSNotificationSource Enumeration

Indicates the source of notification.

Unit

MSAccess

Syntax

```
TMSNotificationSource = (nsClient, nsData, nsDatabase,
 nsEnvironment, nsExecution, nsObject, nsStatement, nsSystem,
 nsTimeout, nsUnknown);
```

Values

Value	Meaning
nsClient	Client is the reason of the notification.
nsData	Data in one or more tables referenced in the underlying query was changed.
nsDatabase	Database state was changed.
nsEnvironment	Environment changes that are incompatible with Change notification were applied.
nsExecution	An error occurred during execution.
nsObject	One of the underlying objects was changed (altered, dropped, etc.).
nsStatement	The provided query does not support notifications.
nsSystem	A system-related event has occurred (like server restart).
nsTimeout	The subscription timeout has been expired.
nsUnknown	Used when an option is not recognized.

17.14.3.5 MSAccess.TMSNotificationType Enumeration

Indicates if this notification is generated because of change or by subscription.

Unit

MSAccess

Syntax

```
TMSNotificationType = (ntChange, ntSubscribe, ntUnknown);
```

Values

Value	Meaning
ntChange	Data on the server was actually changed.
ntSubscribe	Notification subscription failed when creating.
ntUnknown	Used when an option is not recognized.

17.14.3.6 MSAccess.TMSObjectType Enumeration

Enumerates the object types supported by TMSMetadata.

Unit

MSAccess

Syntax

```
TMSObjectType = (otDatabases, otColumnPrivileges, otColumns,
 otForeignKeys, otPrimaryKeys, otIndexes, otServerTypes,
 otSchemata, otStatistics, otStoredProcs, otStoredProcParams,
 otAliases, otTables, otSynonyms, otSystemTables, otViews,
 otGlobalTempTables, otLocalTempTables, otSystemViews,
 otAliasesInfo, otTablesInfo, otSynonymsInfo, otSystemTablesInfo,
 otViewsInfo, otGlobalTempTablesInfo, otLocalTempTablesInfo,
 otExternalTablesInfo, otSystemViewsInfo, otTableConstraints,
 otTablePrivileges, otLinkedServers, otAssemblies,
 otAssemblyDependencies, otUserTypes, otXMLCollections,
 otCheckConstraints, otCheckConstraintsByTable,
 otTableStatistics);
```

Values

Value	Meaning
otAliases	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otAliasesInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otAssemblies	**Restrictions:** DatabaseName, SchemaName, AssemblyName, AssemblyID **Rowset name:** DBSCHEMA SQL ASSEMBLIES Rowset
otAssemblyDependencies	**Restrictions:** DatabaseName, SchemaName, AssemblyID, ReferencedAssemblyID **Rowset name:** DBSCHEMA SQL ASSEMBLY DEPENDENCIES Rowset
otCheckConstraints	**Restrictions:** DatabaseName, SchemaName, ConstraintName **Rowset name:** CHECK CONSTRAINTS Rowset
otCheckConstraintsByTable	**Restrictions:** DatabaseName, SchemaName, TableName, ConstraintName **Rowset name:** CHECK CONSTRAINTS BY TABLE Rowset
otColumnPrivileges	**Restrictions:** DatabaseName, SchemaName, TableName, ColumnName **Rowset name:** COLUMN PRIVILEGES Rowset
otColumns	**Restrictions:** DatabaseName, SchemaName, TableName, ColumnName **Rowset name:** COLUMNS Rowset
otDatabases	**Restrictions:** DatabaseName **Rowset name:** CATALOGS Rowset
otExternalTablesInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otForeignKeys	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** FOREIGN KEYS Rowset
otGlobalTempTables	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otGlobalTempTablesInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otIndexes	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** INDEXES Rowset
otLinkedServers	**Restrictions:** LinkedServer **Rowset name:** DBSCHEMA LINKEDSERVERS Rowset
otLocalTempTables	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otLocalTempTablesInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otPrimaryKeys	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** PRIMARY KEYS Rowset
otSchemata	**Restrictions:** DatabaseName, SchemaName **Rowset name:** SCHEMATA Rowset
otServerTypes	**Restrictions: Rowset name:** PROVIDER TYPES Rowset
otStatistics	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** STATISTICS Rowset
otStoredProcParams	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** STATISTICS Rowset

otStoredProcs	**Restrictions:** DatabaseName, SchemaName, StoredProcName **Rowset name:** PROCEDURES Rowset
otSynonyms	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otSynonymsInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otSystemTables	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otSystemTablesInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otSystemViews	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otSystemViewsInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otTableConstraints	**Restrictions:** DatabaseName, SchemaName, TableName, ConstraintName **Rowset name:** TABLE CONSTRAINTS Rowset
otTablePrivileges	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLE PRIVILEGES Rowset
otTables	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otTablesInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otTableStatistics	**Restrictions:** DatabaseName, SchemaName, TableName, StatisticsName **Rowset name:** TABLE STATISTICS Rowset
otUserTypes	**Restrictions:** DatabaseName, SchemaName, UDTName **Rowset name:** DBSCHEMA SQL USER TYPES Rowset
otViews	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES Rowset
otViewsInfo	**Restrictions:** DatabaseName, SchemaName, TableName **Rowset name:** TABLES INFO Rowset
otXMLCollections	**Restrictions:** DatabaseName, SchemaName, SchemaCollectionName, TargetNamespaceURI **Rowset name:** DBSCHEMA XML COLLECTIONS Rowset

Remarks

Every member of this enumeration specifies restrictions and has representation in the MSDN OLE DB schema rowset name.

17.14.4 Variables

Variables in the **MSAccess** unit.

Variables

Name	Description
__UseUpdateOptimization	In SDAC 4.00.0.4 update statements execution was optimized. This optimization changed the behaviour of affected rows count retrieval for the tables with triggers.

17.14.4.1 MSAccess.__UseUpdateOptimization Variable

In SDAC 4.00.0.4 update statements execution was optimized. This optimization changed the behaviour of affected rows count retrieval for the tables with triggers.

Unit

MSAccess

Syntax

```
__UseUpdateOptimization: boolean;
```

Remarks

In SDAC 4.00.0.4 update statements execution was optimized. This optimization changed the behaviour of affected rows count retrieval for the tables with triggers. If a trigger performs modifications of other records reacting on a modification in the underlying table, SQL Server sends several values of affected rows count (including for modifications made by a trigger). Prior to SDAC 4.00.0.4 the first value was considered as affected rows count, when in SDAC 4.00.0.4 and higher it's the last value. However neither of these two approaches can be considered correct, as there can be triggers that snap into action both before modification and after modification. There is no way to determine which of the values returned by SQL Server is the correct value of affected rows count. Therefore we do not recommend using the RowsAffected property when updating tables with triggers.
The StrictUpdate mode is based on RowsAffected, therefore we also do not recommend using StrictUpdate when updating tables with triggers.
If you want to disable this optimization, set the UseUpdateOptimization variable to False.

17.14.5 Constants

Constants in the **MSAccess** unit.

Constants

Name	Description
SdacVersion	Read this constant to get the current version number for SDAC.

17.14.5.1 MSAccess.SdacVersion Constant

Read this constant to get the current version number for SDAC.

Unit

MSAccess

Syntax

```
SDACVersion = '6.5.9';
```

17.15 MSCompactConnection

This unit contains implementation of the TMSCompactConnection class.

Classes

Name	Description
TMSCompactConnection	A component for establishing connection to Microsoft SQL Server Compact Edition, providing customized login support, and performing transaction control.
TMSCompactConnectionOptions	This class allows setting up the behaviour of the TMSCompactConnection class.

17.15.1 Classes

Classes in the **MSCompactConnection** unit.

Classes

Name	Description
TMSCompactConnection	A component for establishing connection to Microsoft SQL Server Compact Edition, providing customized login support, and performing transaction control.
TMSCompactConnectionOptions	This class allows setting up the behaviour of the TMSCompactConnection class.

17.15.1.1 MSCompactConnection.TMSCompactConnection Class

A component for establishing connection to Microsoft SQL Server Compact Edition , providing customized login support, and performing transaction control.
For a list of all members of this type, see TMSCompactConnection members.

Unit

`MSCompactConnection`

Syntax

```
TMSCompactConnection = class(TCustomMSConnection);
```

Remarks

The TMSCompactConnection component is used to establish connection to Microsoft SQL Server Compact Edition , provide customized login support, and perform transaction control.
TMSCompactConnection publishes connection-related properties derived from its ancestor class TCustomMSConnection.
Note: if you would like to use SDAC in a service, console or just in a separate thread, you need to call CoInitialize for each thread. Also call CoUnInitialize when the thread is finished.

Inheritance Hierarchy

TObject
  TCustomDAConnection
    TCustomMSConnection
      TMSCompactConnection

See Also

- TCustomMSDataSet.Connection
- TMSSQL.Connection
- MSDN: Microsoft SQL Server Compact Edition

TMSCompactConnection class overview.

Properties

Name	Description
ClientVersion (inherited from **TCustomMSConnection**)	Contains the version of Microsoft OLE DB Provider for SQL Server.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectString (inherited from **TCustomMSConnection**)	Used to specify the connection information for the data store.

ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Database (inherited from **TCustomMSConnection**)	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
InitMode	Used to specify the file mode to be used for opening a database file.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
IsolationLevel (inherited from **TCustomMSConnection**)	Used to specify the extent to which all outside transactions interfere with subsequent transactions of the current connection.
LockEscalation	Specifies the number of locks to perform before escalating from row to table or from page to table..
LockTimeout	Used to specify the time for which a transaction will wait for a lock.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Used to specify the behaviour of a TMSCompactConnection object.
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.
Server (inherited from **TCustomDAConnection**)	Serves to supply the server name for login.
ServerVersion (inherited from **TCustomMSConnection**)	Contains the exact number of the SQL Server version.
TransactionCommitMode	Used to specify the way the buffer pool will be flushed on transaction commit.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

Methods

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect (inherited from **TCustomMSConnection**)	Shares database connection between the TCustomMSConnection components.
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 **TCustomMSConnection**)	Returns a new instance of the TMSSQL class and associates it with this connection object.
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.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
OpenDatasets (inherited from **TCustomMSConnection**)	Opens several datasets as one batch.
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.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.

Events

Name	Description
OnConnectionLost (inherited from **TCustomDAConnection**)	This event occurs when connection was lost.
OnError (inherited from **TCustomDAConnection**)	This event occurs when an error has arisen in the connection.

Properties of the **TMSCompactConnection** class.
For a complete list of the **TMSCompactConnection** class members, see the TMSCompactConnection Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect (inherited from **TCustomMSConnection**)	Shares database connection between the TCustomMSConnection components.
ClientVersion (inherited from **TCustomMSConnection**)	Contains the version of Microsoft OLE DB Provider for SQL Server.
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.
ConnectString (inherited from **TCustomMSConnection**)	Used to specify the connection information for the data store.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
CreateDataSet (inherited from **TCustomDAConnection**)	Creates a dataset component.

CreateSQL (inherited from **TCustomMSConnection**)	Returns a new instance of the TMSSQL class and associates it with this connection object.
Database (inherited from **TCustomMSConnection**)	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
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.
IsolationLevel (inherited from **TCustomMSConnection**)	Used to specify the extent to which all outside transactions interfere with subsequent transactions 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.
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.
OpenDatasets (inherited from **TCustomMSConnection**)	Opens several datasets as one batch.
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.
ServerVersion (inherited from **TCustomMSConnection**)	Contains the exact number of the SQL Server version.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

Published

Name	Description
InitMode	Used to specify the file mode to be used for opening a database file.
LockEscalation	Specifies the number of locks to perform before escalating from row to table or from page to table..
LockTimeout	Used to specify the time for which a transaction will wait for a lock.
Options	Used to specify the behaviour of a TMSCompactConnection object.
TransactionCommitMode	Used to specify the way the buffer pool will be flushed on transaction commit.

See Also

- TMSCompactConnection Class
- TMSCompactConnection Class Members

Used to specify the file mode to be used for opening a database file.

Class

TMSCompactConnection

Syntax

```
property InitMode: TMSInitMode default imReadWrite;
```

Remarks

Use the InitMode property to specify the file mode that will be used to open the database file.

Specifies the number of locks to perform before escalating from row to table or from page to table..

Class

TMSCompactConnection

Syntax

```
property LockEscalation: integer default 100;
```

Remarks

Specifies how many locks should be performed before escalating from row to table or from page to table.
The default value is 100.

Used to specify the time for which a transaction will wait for a lock.

Class

TMSCompactConnection

Syntax

```
property LockTimeout: integer default 2000;
```

Remarks

Specifies how much time a transaction will wait for a lock.

Measured in milliseconds.
The default value is 2000.

Used to specify the behaviour of a TMSCompactConnection object.

Class

TMSCompactConnection

Syntax

```
property Options: TMSCompactConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMSCompactConnection object. Descriptions of all options are in the table below.

Option Name	Description
AutoShrinkThreshold	Specifies the amount of free space in the database file before automatic shrink starts.
CompactVersion	Specifies which version of SQL Server Compact will be used.
DefaultLockEscalation	Specifies how many locks should be performed before trying escalation from row to page or from page to table.
DefaultLockTimeout	Specifies how much time in milliseconds a transaction will wait for a lock.
FlushInterval	Specifies the interval at which committed transactions are flushed to disk.
MaxBufferSize	Specifies how much memory SQL Server Compact Edition can use before flushing changes to disk.
MaxDatabaseSize	Specified maximum size of the main database file.
TempFileDirectory	Specifies the temp file directory.
TempFileMaxSize	Specified maximum size of the temporary database file.

Used to specify the way the buffer pool will be flushed on transaction commit.

Class

TMSCompactConnection

Syntax

```
property TransactionCommitMode: TCompactCommitMode default
  cmAsynchCommit;
```

Remarks

Specifies in what way the buffer pool will be flushed on transaction commit.

17.15.1.2 MSCompactConnection.TMSCompactConnectionOptions Class

This class allows setting up the behaviour of the TMSCompactConnection class.
For a list of all members of this type, see TMSCompactConnectionOptions members.

Unit

MSCompactConnection

Syntax

```
TMSCompactConnectionOptions = class(TCustomMSConnectionOptions);
```

Inheritance Hierarchy

TObject
  TDAConnectionOptions
    TCustomMSConnectionOptions
      TMSCompactConnectionOptions

TMSCompactConnectionOptions class overview.

Properties

Name	Description
AutoShrinkThreshold	Specifies the amount of free space in the database file before automatic shrink starts.
CompactVersion	Specifies which version of SQL Server Compact will be used.
DefaultLockEscalation	Specifies how many locks should be performed before trying escalation from row to page or from page to table.
DefaultLockTimeout	Specifies how much time in milliseconds a transaction will wait for a lock.
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet. IndexFieldNames property of a dataset.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
Encrypt (inherited from **TCustomMSConnectionOptions**)	Specifies if data should be encrypted before sending it over the network.
FlushInterval	Specifies the interval at which committed transactions are flushed to disk.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection. OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
MaxBufferSi e	Specifies how much memory SQL Server Compact Edition can use before flushing changes to disk.

MaxDatabaseSize	Specified maximum size of the main database file.
NumericType (inherited from **TCustomMSConnectionOptions**)	Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.
Provider (inherited from **TCustomMSConnectionOptions**)	Used to specify a provider from the list of supported providers.
QuotedIdentifier (inherited from **TCustomMSConnectionOptions**)	Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.
TempFileDirectory	Specifies the temp file directory.
TempFileMaxSize	Specified maximum size of the temporary database file.
UseWideMemos (inherited from **TCustomMSConnectionOptions**)	Used to manage field type creation for the NTEXT data type.

Properties of the **TMSCompactConnectionOptions** class.
For a complete list of the **TMSCompactConnectionOptions** class members, see the TMSCompactConnectionOptions Members topic.

Public

Name	Description
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet. IndexFieldNames property of a dataset.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
Encrypt (inherited from **TCustomMSConnectionOptions**)	Specifies if data should be encrypted before sending it over the network.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection. OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
NumericType (inherited from **TCustomMSConnectionOptions**)	Specifies the format of storing and representing the NUMERIC (DECIMAL) fields for all TCustomMSDataSets associated with the given connection.
Provider (inherited from **TCustomMSConnectionOptions**)	Used to specify a provider from the list of supported providers.
QuotedIdentifier (inherited from **TCustomMSConnectionOptions**)	Causes Microsoft SQL Server to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings.
UseWideMemos (inherited from **TCustomMSConnectionOptions**)	Used to manage field type creation for the NTEXT data type.

Published

Name	Description
AutoShrinkThreshold	Specifies the amount of free space in the database file before automatic shrink starts.
CompactVersion	Specifies which version of SQL Server Compact will be used.
DefaultLockEscalation	Specifies how many locks should be performed before trying escalation from row to page or from page to table.
DefaultLockTimeout	Specifies how much time in milliseconds a transaction will wait for a lock.
FlushInterval	Specifies the interval at which committed transactions are flushed to disk.
MaxBufferSize	Specifies how much memory SQL Server Compact Edition can use before flushing changes to disk.
MaxDatabaseSize	Specified maximum size of the main database file.
TempFileDirectory	Specifies the temp file directory.
TempFileMaxSize	Specified maximum size of the temporary database file.

See Also

- TMSCompactConnectionOptions Class
- TMSCompactConnectionOptions Class Members

Specifies the amount of free space in the database file before automatic shrink starts.

Class

TMSCompactConnectionOptions

Syntax

```
property AutoShrinkThreshold: integer default 60;
```

Remarks

Use the AutoShrinkThreshold property to specify the amount of free space in the database file before automatic shrink starts. Measured in percents. The default value is 60.

Specifies which version of SQL Server Compact will be used.

Class

TMSCompactConnectionOptions

Syntax

```
property CompactVersion: TCompactVersion;
```

Remarks

Use the CompactVersion property to specify which version of SQL Server Compact will be used. Useful when there are SQL Server Compact 3.0 and 3.5 installed on the same computer. The default value is cvAuto. It means that if both server versions were installed simultaneously, SQL Server Compact 3.5 will be used.

Specifies how many locks should be performed before trying escalation from row to page or from page to table.

Class

TMSCompactConnectionOptions

Syntax

```
property DefaultLockEscalation: integer default 100;
```

Remarks

Use the DefaultLockEscalation property to specify how many locks should be performed before trying escalation from row to page or from page to table. The default value is 100.

Specifies how much time in milliseconds a transaction will wait for a lock.

Class

TMSCompactConnectionOptions

Syntax

```
property DefaultLockTimeout: integer;
```

Remarks

Use the DefaultLockTimeout property to specify how much time in milliseconds a transaction will wait for a lock. The default value is 2000 ms.

Specifies the interval at which committed transactions are flushed to disk.

Class

TMSCompactConnectionOptions

Syntax

```
property FlushInterval: integer default 10;
```

Remarks

Use the FlushInterval property to specify the interval at which committed transactions are flushed to disk. Measured in seconds. The default value is 10.

Specifies how much memory SQL Server Compact Edition can use before flushing changes to disk.

Class

TMSCompactConnectionOptions

Syntax

```
property MaxBufferSize: integer default 640;
```

Remarks

Use the MaxBufferSize property to specify how much memory SQL Server Compact Edition can use before flushing changes to disk. Measured in kilobytes. The default value is 640.

Specified maximum size of the main database file.

Class

TMSCompactConnectionOptions

Syntax

```
property MaxDatabaseSize: integer default 128;
```

Remarks

Specified maximum size of the main database file. Measured in megabytes. The default value is 128.

Specifies the temp file directory.

Class

TMSCompactConnectionOptions

Syntax

```
property TempFileDirectory: string;
```

Remarks

Use the TempFileDirectory property to specify the temp file directory. If this option is not assigned, the current database is used as a temporary database.

Specified maximum size of the temporary database file.

Class

TMSCompactConnectionOptions

Syntax

```
property TempFileMaxSize: integer default 128;
```

Remarks

Specified maximum size of the temporary database file. Measured in megabytes. The default value is 128.

17.16 MSConnectionPool

This unit contains the TMSConnectionPoolManager class for managing connection pool.

Classes

Name	Description
TMSConnectionPoolManager	A class of methods that are used for managing SDAC connection pool.

17.16.1 Classes

Classes in the **MSConnectionPool** unit.

Classes

Name	Description
TMSConnectionPoolManager	A class of methods that are used for managing SDAC connection pool.

17.16.1.1 MSConnectionPool.TMSConnectionPoolManager Class

A class of methods that are used for managing SDAC connection pool.
For a list of all members of this type, see TMSConnectionPoolManager members.

Unit

`MSConnectionPool`

Syntax

```
TMSConnectionPoolManager = class(TCRConnectionPoolManager);
```

Remarks

Use the TMSConnectionPoolManager methods to manage SDAC connection pool.

Inheritance Hierarchy

TObject
  TCRConnectionPoolManager
    TMSConnectionPoolManager

See Also

- Connection Pooling

TMSConnectionPoolManager class overview.

17.17 MSDump

This unit contains implementation of the TMSDump component.

Classes

Name	Description
TMSDump	The class that serves for storing data from tables or editable views as a script and for restoring data from a received script.
TMSDumpOptions	This class allows setting up the behaviour of the TMSDump class.

17.17.1 Classes

Classes in the **MSDump** unit.

Classes

Name	Description
TMSDump	The class that serves for storing data from tables or editable views as a script and for restoring data from a received script.
TMSDumpOptions	This class allows setting up the behaviour of the TMSDump class.

17.17.1.1 MSDump.TMSDump Class

The class that serves for storing data from tables or editable views as a script and for restoring data from a received script.
For a list of all members of this type, see TMSDump members.

Unit

`MSDump`

Syntax

```
TMSDump = class(TDADump);
```

Remarks

TMSDump serves to store data from tables or editable views as a script and to restore data from a received script.
Use the TDADump.TableNames property to specify the list of objects to be stored. To launch a generating script, call the TDADump.Backup method.
TMSDump also can generate scripts for a query. Just call the TDADump.BackupQuery method and pass a query statement into it. The object list assigned to the TableNames property is ignored if you call TDADump.BackupQuery. The generated script can be viewed in the TDADump.SQL property.
TMSDump works on the client side. It causes large network loading.

Inheritance Hierarchy

TObject
- TDADump
  - **TMSDump**

See Also

- TDADump.Backup
- TDADump.Restore

TMSDump class overview.

Properties

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Debug (inherited from **TDADump**)	Used to display executing statement, all its parameters' values, and the type of parameters.
Options	Used to specify the behaviour of a TMSDump object.

SQL (inherited from **TDADump**)	Used to set or get the dump script.
TableNames (inherited from **TDADump**)	Used to set the names of the tables to dump.

Methods

Name	Description
Backup (inherited from **TDADump**)	Dumps database objects to the TDADump.SQL property.
BackupQuery (inherited from **TDADump**)	Dumps the results of a particular query.
BackupToFile (inherited from **TDADump**)	Dumps database objects to the specified file.
BackupToStream (inherited from **TDADump**)	Dumps database objects to the stream.
Restore (inherited from **TDADump**)	Executes a script contained in the SQL property.
RestoreFromFile (inherited from **TDADump**)	Executes a script from a file.
RestoreFromStream (inherited from **TDADump**)	Executes a script received from the stream.

Events

Name	Description
OnBackupProgress (inherited from **TDADump**)	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 (inherited from **TDADump**)	Occurs when SQL Server raises some error on TDADump.Restore.
OnRestoreProgress (inherited from **TDADump**)	Occurs to indicate the TDADump. Restore, TDADump.RestoreFromFile , or TDADump.RestoreFromStream method execution progress.

Properties of the **TMSDump** class.
For a complete list of the **TMSDump** class members, see the TMSDump Members topic.

Public

Name	Description
Backup (inherited from **TDADump**)	Dumps database objects to the TDADump.SQL property.
BackupQuery (inherited from **TDADump**)	Dumps the results of a particular query.
BackupToFile (inherited from **TDADump**)	Dumps database objects to the specified file.
BackupToStream (inherited from **TDADump**)	Dumps database objects to the stream.
Restore (inherited from **TDADump**)	Executes a script contained in the SQL property.
RestoreFromFile (inherited from **TDADump**)	Executes a script from a file.
RestoreFromStream (inherited from **TDADump**)	Executes a script received from the stream.

Published

Name	Description

Connection	Used to specify a connection object that will be used to connect to a data store.
Debug (inherited from **TDADump**)	Used to display executing statement, all its parameters' values, and the type of parameters.
OnBackupProgress (inherited from **TDADump**)	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 (inherited from **TDADump**)	Occurs when SQL Server raises some error on TDADump.Restore.
OnRestoreProgress (inherited from **TDADump**)	Occurs to indicate the TDADump.Restore, TDADump.RestoreFromFile, or TDADump.RestoreFromStream method execution progress.
Options	Used to specify the behaviour of a TMSDump object.
SQL (inherited from **TDADump**)	Used to set or get the dump script.
TableNames (inherited from **TDADump**)	Used to set the names of the tables to dump.

See Also

- TMSDump Class
- TMSDump Class Members

Used to specify a connection object that will be used to connect to a data store.

Class

TMSDump

Syntax

```
property Connection: TMSConnection;
```

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 the provided TMSConnection objects.
At runtime, set the Connection property to reference an existing TMSConnection object.

See Also

- TMSConnection

Used to specify the behaviour of a TMSDump object.

Class

TMSDump

Syntax

```
property Options: TMSDumpOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMSDump object.

Descriptions of all options are in the table below.

Option Name	Description
DisableConstraints	Allows to disable foreign keys when dumping multiple tables.
IdentityInsert	Used to add SET IDENTITY INSERT TableName ON at the beginning of the script and SET IDENTITY INSERT TableName OFF at the end of the script.

17.17.1.2 MSDump.TMSDumpOptions Class

This class allows setting up the behaviour of the TMSDump class.
For a list of all members of this type, see TMSDumpOptions members.

Unit

MSDump

Syntax

```
TMSDumpOptions = class(TDADumpOptions);
```

Inheritance Hierarchy

TObject
  TDADumpOptions
    TMSDumpOptions

TMSDumpOptions class overview.

Properties

Name	Description
AddDrop (inherited from **TDADumpOptions**)	Used to add drop statements to a script before creating statements.
DisableConstraints	Allows to disable foreign keys when dumping multiple tables.
GenerateHeader (inherited from **TDADumpOptions**)	Used to add a comment header to a script.
IdentityInsert	Used to add SET IDENTITY INSERT TableName ON at the beginning of the script and SET IDENTITY INSERT TableName OFF at the end of the script.
QuoteNames (inherited from **TDADumpOptions**)	Used for TDADump to quote all database object names in generated SQL statements.

Properties of the **TMSDumpOptions** class.
For a complete list of the **TMSDumpOptions** class members, see the TMSDumpOptions Members topic.

Published

Name	Description
AddDrop (inherited from **TDADumpOptions**)	Used to add drop statements to a script before creating statements.

DisableConstraints	Allows to disable foreign keys when dumping multiple tables.
GenerateHeader (inherited from **TDADumpOptions**)	Used to add a comment header to a script.
IdentityInsert	Used to add SET IDENTITY INSERT TableName ON at the beginning of the script and SET IDENTITY INSERT TableName OFF at the end of the script.
QuoteNames (inherited from **TDADumpOptions**)	Used for TDADump to quote all database object names in generated SQL statements.

See Also

- TMSDumpOptions Class
- TMSDumpOptions Class Members

Allows to disable foreign keys when dumping multiple tables.

Class

TMSDumpOptions

Syntax

```
property DisableConstraints: boolean default False;
```

Remarks

Use the DisableConstraints property to disable foreign keys when dumping multiple tables. The default value is False.

Used to add SET IDENTITY INSERT TableName ON at the beginning of the script and SET IDENTITY INSERT TableName OFF at the end of the script.

Class

TMSDumpOptions

Syntax

```
property IdentityInsert: boolean default False;
```

Remarks

Use the IdentityInsert property to add SET IDENTITY INSERT TableName ON at the beginning of the script and SET IDENTITY INSERT TableName OFF at the end of the script. The first line allows explicit values to be inserted into the identity column of a table and INSERT statements are generated with IDENTITY field values. Otherwise the IDENTITY field will not be included to the INSERT statements. SET IDENTITY INSERT will not be added while the option is ON if the table does not have a field identified as IDENTITY or there are no records in the table.

17.18 MSLoader

This unit contains implementation of the TMSLoader component.

Classes

Name	Description
TMSColumn	A component representing the attributes for column loading.
TMSLoader	TMSLoader allows to load external data into the server database.
TMSLoaderOptions	This class allows setting up the behaviour of the TMSLoader class.

Types

Name	Description
TMSPutDataEvent	This type is used for the TMSLoader.OnPutData event.

17.18.1 Classes

Classes in the **MSLoader** unit.

Classes

Name	Description
TMSColumn	A component representing the attributes for column loading.
TMSLoader	TMSLoader allows to load external data into the server database.
TMSLoaderOptions	This class allows setting up the behaviour of the TMSLoader class.

17.18.1.1 MSLoader.TMSColumn Class

A component representing the attributes for column loading.
For a list of all members of this type, see TMSColumn members.

Unit

`MSLoader`

Syntax

```
TMSColumn = class(TDAColumn);
```

Remarks

Each TMSLoader uses TDAColumns to maintain a collection of TMSColumn objects. TMSColumn object represents the attributes for column loading. Every TMSColumn object corresponds to one of the table fields with the same name as its TDAColumn.Name property.
To create columns at design-time use column editor of the TMSLoader component.

Inheritance Hierarchy

TObject
  TDAColumn
    TMSColumn

See Also

- TMSLoader
- TDAColumns

TMSColumn class overview.

Properties

Name	Description
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	Defines the size used in the definition of the physical database field for the data types that support different precision.
Scale	Used to set the scale of numeric values.
Size	Used to set the size of numeric values.

Properties of the **TMSColumn** class.
For a complete list of the **TMSColumn** class members, see the TMSColumn Members topic.

Published

Name	Description
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	Defines the size used in the definition of the physical database field for the data types that support different precision.
Scale	Used to set the scale of numeric values.
Size	Used to set the size of numeric values.

See Also

- TMSColumn Class
- TMSColumn Class Members

Defines the size used in the definition of the physical database field for the data types that support different precision.

Class

TMSColumn

Syntax

```
property Precision: integer default 0;
```

Remarks

The Precision property can be filled automatically, when calling TDALoader.CreateColumns or when setting the TDALoader.TableName property. User can manually create fields by filling TMSColumn properties.

Used to set the scale of numeric values.

Class

TMSColumn

Syntax

```
property Scale: integer default 0;
```

Remarks

Use the Scale property to set the scale of numeric values.

Used to set the size of numeric values.

Class

TMSColumn

Syntax

```
property Size: integer;
```

Remarks

Use the Size property to set the size of numeric values.

17.18.1.2 MSLoader.TMSLoader Class

TMSLoader allows to load external data into the server database.
For a list of all members of this type, see TMSLoader members.

Unit

`MSLoader`

Syntax

```
TMSLoader = class(TDALoader);
```

Remarks

TMSLoader serves for fast loading of data to the server. TMSLoader functionality is based on the SQL Server memory-based bulk-copy operations using the IRowsetFastLoad interface. Simultaneous loading into multiple tables is not supported.

Inheritance Hierarchy

TObject
  TDALoader
    TMSLoader

See Also

- MSDN: IRowsetFastLoad Usage and Limitations

TMSLoader 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.
KeepIdentity	Used to specify the way IDENTITY column values must be handled.
KeepNulls	Used to specify the way NULL values for columns with a DEFAULT constraint must be handled.
Options	Used to specify the behaviour of a TMSLoader object.
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
OnGetColumnData	Occurs when putting 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.

Properties of the **TMSLoader** class.
For a complete list of the **TMSLoader** class members, see the TMSLoader 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
KeepIdentity	Used to specify the way IDENTITY column values must be handled.
KeepNulls	Used to specify the way NULL values for columns with a DEFAULT constraint must be handled.
Options	Used to specify the behaviour of a TMSLoader object.

See Also

- TMSLoader Class
- TMSLoader Class Members

Used to specify the way IDENTITY column values must be handled.

Class

TMSLoader

Syntax

```
property KeepIdentity: boolean default False;
```

Remarks

Use the KeepIdentity property to specify in what way IDENTITY column values must be handled. If KeepIdentity is set to False, IDENTITY columns will be initialized by the server. Any value assigned to such column in your application is ignored. If KeepIdentity is set to True, the IDENTITY property will not be available for all IDENTITY fields accepting NULL. So in this case unique values should be generated and assigned by the client application. The default value of the KeepIdentity property is False.

Used to specify the way NULL values for columns with a DEFAULT constraint must be handled.

Class

TMSLoader

Syntax

```
property KeepNulls: boolean default False;
```

Remarks

If this option is set to False, each NULL value inserted into a field with a DEFAULT constraint will be replaced with the default value. If KeepNulls is set to True, NULL values inserted into a field with a DEFAULT constraint will not be replaced with the default values. The default value of the KeepNulls property is False.

Used to specify the behaviour of a TMSLoader object.

Class

TMSLoader

Syntax

```
property Options: TMSLoaderOptions;
```

Remarks

Use the Options property to specify the behaviour of a TMSLoader object.
Descriptions of all options are in the table below.

Option Name	Description
CheckConstraints	Used to specify if the table constraints are checked during loading.
KilobytesPerBatch	Used to specify the size of data in kilobytes to load in a single batch.
LockTable	Used to specify if the table-level lock is performed while loading is in progress.
RowsPerBatch	Used to specify the number of rows to load in a single batch.

Events of the **TMSLoader** class.
For a complete list of the **TMSLoader** class members, see the TMSLoader 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
OnGetColumnData	Occurs when putting column values.
OnPutData	Occurs when putting loading data by rows.

See Also

- TMSLoader Class
- TMSLoader Class Members

Occurs when putting column values.

Class

`TMSLoader`

Syntax

```
property OnGetColumnData: TMSGetColumnDataEvent;
```

Remarks

Write the OnGetColumnData event handler to put column values. TMSLoader calls the OnGetColumnData event handler for each column in the loop. Column points to a TMSLoader 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. TMSLoader increments the Row parameter when all columns of the current record are loaded. The first row is 1. Set IsEOF to True to stop data loading. Fill the Value parameter by column values. To start loading call the TDALoader.Load method.
Another way to load data is using the OnPutData event.

See Also

- OnPutData
- TDALoader.Load
- TDALoader.OnGetColumnData

Occurs when putting loading data by rows.

Class

`TMSLoader`

Syntax

```
property OnPutData: TMSPutDataEvent;
```

Remarks

Write the OnPutData event handler to put loading data by rows. Note that rows should be loaded from the first one in ascending order. To start loading call the TDALoader.Load method.

See Also

- TDALoader.PutColumnData
- TDALoader.Load
- OnGetColumnData
- TDALoader.OnPutData

17.18.1.3 MSLoader.TMSLoaderOptions Class

This class allows setting up the behaviour of the TMSLoader class.
For a list of all members of this type, see TMSLoaderOptions members.

Unit

`MSLoader`

Syntax

```
TMSLoaderOptions = class(TPersistent);
```

Inheritance Hierarchy

TObject
  TMSLoaderOptions

TMSLoaderOptions class overview.

Properties

Name	Description
CheckConstraints	Used to specify if the table constraints are checked during loading.
KilobytesPerBatch	Used to specify the size of data in kilobytes to load in a single batch.
LockTable	Used to specify if the table-level lock is performed while loading is in progress.
RowsPerBatch	Used to specify the number of rows to load in a single batch.

Properties of the **TMSLoaderOptions** class.
For a complete list of the **TMSLoaderOptions** class members, see the TMSLoaderOptions Members topic.

Published

Name	Description
CheckConstraints	Used to specify if the table constraints are checked during loading.
KilobytesPerBatch	Used to specify the size of data in kilobytes to load in a single batch.
LockTable	Used to specify if the table-level lock is performed while loading is in progress.
RowsPerBatch	Used to specify the number of rows to load in a single batch.

See Also

- TMSLoaderOptions Class
- TMSLoaderOptions Class Members

Used to specify if the table constraints are checked during loading.

Class

TMSLoaderOptions

Syntax

```
property CheckConstraints: boolean default False;
```

Remarks

Use the CheckConstraints property to specify if the table constraints are checked during loading. If this option is set to False, the table constraints are not checked. The default value of the CheckConstraints option is False.

Used to specify the size of data in kilobytes to load in a single batch.

Class

TMSLoaderOptions

Syntax

```
property KilobytesPerBatch: integer default 0;
```

Remarks

Use the KilobytesPerBatch option to specify the size of data in kilobytes to load in a single batch. The default value of this option is Unknown.

Used to specify if the table-level lock is performed while loading is in progress.

Class

TMSLoaderOptions

Syntax

```
property LockTable: boolean default False;
```

Remarks

Use the LockTable property to specify if the table-level lock is performed while loading is in progress. Setting this option to True should improve the performance greatly. If this option is set to False, the locking behaviour is determined by the table option. The default value of the LockTable option is False.

Used to specify the number of rows to load in a single batch.

Class

TMSLoaderOptions

Syntax

```
property RowsPerBatch: integer default 0;
```

Remarks

Use the RowsPerBatch property to specify the number of rows to load in a single batch. Server optimizes loading according to this value. The default value of this option is Unknown.

17.18.2 Types

Types in the **MSLoader** unit.

Types

Name	Description
TMSPutDataEvent	This type is used for the TMSLoader.OnPutData event.

17.18.2.1 MSLoader.TMSPutDataEvent Procedure Reference

This type is used for the TMSLoader.OnPutData event.

Unit

`MSLoader`

Syntax

```
TMSPutDataEvent = procedure (Sender: TMSLoader) of object;
```

Parameters

Sender
An object that raised the event.

17.19 MSScript

This unit contains implementation of the TMSScript component.

Classes

Name	Description
TMSScript	A component for executing several SQL statements one by one.

17.19.1 Classes

Classes in the **MSScript** unit.

Classes

Name	Description
TMSScript	A component for executing several SQL statements one by one.

17.19.1.1 MSScript.TMSScript Class

A component for executing several SQL statements one by one.
For a list of all members of this type, see TMSScript members.

Unit

`MSScript`

Syntax

```
TMSScript = 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 TMSSQL. Usually it is not a good solution. With only one TMSScript component you can execute several SQL statements as one. This sequence of statements is named script. To separate single statements use semicolon (;), slash (/) or keyword 'GO' . 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 TMSScript shows exception and continues execution.
If you need to create several Stored Procedures (Functions) at a single script, use slash (/) to separate commands for creating stored procedures.

Inheritance Hierarchy

TObject
  TDAScript
    TMSScript

See Also

- TMSSQL

TMSScript class overview.

Properties

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
DataSet	Used to retrieve the results of the SELECT statements execution inside a script.
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.
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 (inherited from **TDAScript**)	Contains a list of statements obtained from the SQL property.
UseOptimization	Used to bind small queries in blocks for block executing.

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 SQL Server raises an error.

Properties of the **TMSScript** class.
For a complete list of the **TMSScript** class members, see the TMSScript Members topic.

Public

Name	**Description**
BreakExec (inherited from **TDAScript**)	Stops script execution.
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.
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.
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 (inherited from **TDAScript**)	Contains a list of statements obtained from the SQL property.

Published

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.
Connection	Used to specify a connection object that will be used to connect to a data store.
DataSet	Used to retrieve the results of the SELECT statements execution inside a script.
Debug (inherited from **TDAScript**)	Used to display the script execution and all its parameter values.
Delimiter (inherited from **TDAScript**)	Used to set the delimiter string that separates script statements.
Macros (inherited from **TDAScript**)	Used to change SQL script text in design- or run-time easily.
OnError (inherited from **TDAScript**)	Occurs when SQL Server raises an error.
SQL (inherited from **TDAScript**)	Used to get or set script text.
UseOptimization	Used to bind small queries in blocks for block executing.

See Also

- TMSScript Class
- TMSScript Class Members

Used to specify a connection object that will be used to connect to a data store.

Class

`TMSScript`

Syntax

```
property Connection: TCustomMSConnection;
```

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 TMSConnection objects.
At run-time, set the Connection property to reference an existing TMSConnection object.

See Also

- TMSConnection

Used to retrieve the results of the SELECT statements execution inside a script.

Class

`TMSScript`

Syntax

```
property DataSet: TCustomMSDataSet;
```

Remarks

Set the DataSet property if you need to retrieve the results of the SELECT statements execution inside a script.

See Also

- TDAScript.Execute

Used to bind small queries in blocks for block executing.

Class

`TMSScript`

Syntax

```
property UseOptimization: boolean;
```

Remarks

If the UseOptimization property is set, small queries will be united into blocks for block executing if possible. The UseOptimization option does not affect the TDAScript.ExecuteNext method performance. It works only for the TDAScript.Execute method.

17.20 MSServiceBroker

This unit contains implementation of the TMSServiceBroker component and auxiliary classes.

Classes

Name	Description
TMSConversation	A base class that describes the dialog process between two services.
TMSMessage	A class representing a Service Broker message of Microsoft SQL Server.
TMSServiceBroker	A component that provides sending and receiving messages within the SQL Server 2005 Service Broker system.

Enumerations

Name	Description
TMSMessageValidation	Defines the type of validation performed.

17.20.1 Classes

Classes in the **MSServiceBroker** unit.

Classes

Name	Description
TMSConversation	A base class that describes the dialog process between two services.
TMSMessage	A class representing a Service Broker message of Microsoft SQL Server.
TMSServiceBroker	A component that provides sending and receiving messages within the SQL Server 2005 Service Broker system.

17.20.1.1 MSServiceBroker.TMSConversation Class

A base class that describes the dialog process between two services.
For a list of all members of this type, see TMSConversation members.

Unit

`MSServiceBroker`

Syntax

```
TMSConversation = class(System.TObject);
```

Remarks

The TMSConversation class describes the dialog process between two services.
To start a new conversation, use the TMSServiceBroker.BeginDialog method. You can finish the dialog by calling the TMSConversation.EndConversation method.
Note: You should not create and delete TMSConversation objects by calling the Create and Free methods directly. Use the TMSServiceBroker.BeginDialog and TMSConversation.EndConversation methods instead.

Inheritance Hierarchy

TObject
  TMSConversation

See Also

- TMSMessage
- TMSServiceBroker
- TMSServiceBroker.Conversations

TMSConversation class overview.

Properties

Name	Description
ContractName	Stores the contact name the conversation conforms to.
FarService	Holds the service name of the second side taking part in the dialog.

GroupId	Holds the unique identifier of a conversation group.
Handle	Holds a unique identifier of the current conversation.
IsInitiator	Determines if the conversation was initiated by this side.
ServiceBroker	Determines the TMSServiceBroker object to which a conversation instance belongs.

Methods

Name	Description
BeginTimer	Provides a message of the SDialogTimerType message type after the time specified in the Timeout property has been expired.
EndConversation	Terminates the conversation.
EndConversationWithError	Terminates the conversation and provides the text and code of the error.
GetTransmissionStatus	Returns the status of the last sent message.
Send	Overloaded. Sends a message within a conversation.
SendEmpty	Sends an empty message within a conversation.

Properties of the **TMSConversation** class.
For a complete list of the **TMSConversation** class members, see the TMSConversation Members topic.

Public

Name	Description
ContractName	Stores the contact name the conversation conforms to.
FarService	Holds the service name of the second side taking part in the dialog.
GroupId	Holds the unique identifier of a conversation group.
Handle	Holds a unique identifier of the current conversation.
IsInitiator	Determines if the conversation was initiated by this side.
ServiceBroker	Determines the TMSServiceBroker object to which a conversation instance belongs.

See Also

- TMSConversation Class
- TMSConversation Class Members

Stores the contact name the conversation conforms to.

Class

`TMSConversation`

Syntax

```
property ContractName: string;
```

Remarks

Stores the name of the contract that the conversation conforms to.

See Also

- TMSServiceBroker.BeginDialog

Holds the service name of the second side taking part in the dialog.

Class

`TMSConversation`

Syntax

```
property FarService: string;
```

Remarks

The service name of the second side that is taking part in the dialog.

See Also

- TMSServiceBroker.Service

Holds the unique identifier of a conversation group.

Class

`TMSConversation`

Syntax

```
property GroupId: TGuid;
```

Remarks

Use the GroupId property to store a unique identifier of the conversation group. It is used for executing Transact-SQL queries.
The value of this property can be set when calling the TMSServiceBroker.BeginDialog method with the RelatedConversation or the GroupId parameter. If TMSServiceBroker.BeginDialog was called with these parameters omitted, the GroupId will be assigned to the unique value.
If the conversation already exists, you can assign a new value for GroupId. See the description of the MOVE CONVERSATION statement in MSDN for details.

See Also

- Handle

Holds a unique identifier of the current conversation.

Class

`TMSConversation`

Syntax

```
property Handle: TGuid;
```

Remarks

The Handle property stores a unique identifier of the current conversation. It is used for executing Transact-SQL queries. Handle is a read-only property.

See Also

- GroupId

Determines if the conversation was initiated by this side.

Class

`TMSConversation`

Syntax

```
property IsInitiator: boolean;
```

Remarks

Indicates whether the conversation was initiated by this side.

See Also

- TMSServiceBroker.Service
- TMSServiceBroker.BeginDialog
- FarService

Determines the TMSServiceBroker object to which a conversation instance belongs.

Class

`TMSConversation`

Syntax

```
property ServiceBroker: TMSServiceBroker;
```

Remarks

Use the ServiceBroker property to identify the TMSServiceBroker object to which a conversation instance belongs.

See Also

- TMSServiceBroker.Conversations

Methods of the **TMSConversation** class.
For a complete list of the **TMSConversation** class members, see the TMSConversation Members topic.

Public

Name	Description

BeginTimer	Provides a message of the SDialogTimerType message type after the time specified in the Timeout property has been expired.
EndConversation	Terminates the conversation.
EndConversationWithError	Terminates the conversation and provides the text and code of the error.
GetTransmissionStatus	Returns the status of the last sent message.
Send	Overloaded. Sends a message within a conversation.
SendEmpty	Sends an empty message within a conversation.

See Also

- TMSConversation Class
- TMSConversation Class Members

Provides a message of the SDialogTimerType message type after the time specified in the Timeout property has been expired.

Class

`TMSConversation`

Syntax

```
procedure BeginTimer(const Timeout: integer);
```

Parameters

Timeout
Specifies the amount of time to wait before displaying the message.

Remarks

Call the BeginTimer method to receive a message of the SDialogTimerType message type after the time specified in the Timeout property (in seconds) is expired.
See the description of the BEGIN CONVERSATION TIMER statement in MSDN for details.

See Also

- TMSMessage
- TMSServiceBroker.CurrentMessage

Terminates the conversation.

Class

`TMSConversation`

Syntax

```
procedure EndConversation(const Cleanup: boolean = False);
```

Parameters

Cleanup
True, if undelivered messages should be deleted. False otherwise.

Remarks

Call the EndConversation method to terminate a conversation. The Cleanup parameter determines

whether the undelivered massages will be deleted.
See the description of the END CONVERSATION statement in MSDN for details.

See Also

- EndConversationWithError
- TMSServiceBroker.BeginDialog

Terminates the conversation and provides the text and code of the error.

Class

TMSConversation

Syntax

```
procedure EndConversationWithError(const ErrorMessage: string;
 const ErrorCode: integer; const Cleanup: boolean = False);
```

Parameters

ErrorMessage
Holds the text of the error.

ErrorCode
Holds the code of the error.

Cleanup
True, if undelivered messages should be deleted. False otherwise.

Remarks

Use the EndConversationWithError method to terminate a conversation and handle the text (ErrorMessage) and code (ErrorCode) of the error. The Cleanup parameter determines whether the undelivered massages will be deleted.
See the description of the END CONVERSATION statement in MSDN for details.

See Also

- EndConversationWithError
- TMSServiceBroker.BeginDialog

Returns the status of the last sent message.

Class

TMSConversation

Syntax

```
function GetTransmissionStatus: string;
```

Return Value

the status of the last sent message.

Remarks

Call the GetTransmissionStatus method to return the status of the last sent message.
See the description of the GET_TRANSMISSION_STATUS statement in MSDN for details.

See Also

- TMSConversation.Send

Sends a message within a conversation.

Class

TMSConversation

Overload List

Name	Description
Send(**const** MessageBody: TBytes; **const** MessageType: string)	Sends a message within a conversation.
Send(**const** MessageBody: string; **const** MessageType: string)	Sends a message within a conversation.

Sends a message within a conversation.

Class

TMSConversation

Syntax

```
procedure Send(const MessageBody: TBytes; const MessageType:
  string = ''); overload
```

Parameters

MessageBody
Holds the message to be sent.

MessageType
Determines the type of the message being sent.

Remarks

The Send method sends a message within a conversation. The target service is determined when creating a conversation.
The MessageType parameter determines the type of the message being sent. For the detailed message types description see the description of the CREATE MESSAGE TYPE statement in MSDN.
You can check the status of the last sent message by the TMSConversation.GetTransmissionStatus method.
Note: The method overload with the WideString MessageBody parameter type is not supported under Delphi 5.

See Also

- TMSConversation.GetTransmissionStatus
- TMSConversation.SendEmpty
- TMSMessage.AsString
- TMSMessage.AsWideString
- TMSMessage.AsBytes

Sends a message within a conversation.

Class

TMSConversation

Syntax

```
procedure Send(const MessageBody: string; const MessageType:
```

```
string = ''); overload
```

Parameters

MessageBody
Holds the message to be sent.

MessageType
Determines the type of the message being sent.

Sends an empty message within a conversation.

Class

TMSConversation

Syntax

```
procedure SendEmpty(const MessageType: string = '');
```

Parameters

MessageType
Determines the type of the message being sent.

Remarks

Sends an empty message within a conversation. The target service is determined when creating a conversation.
This method must be used to send an empty message. This is the only method that can be used to send a message if MessageType was created with the Validation property set to mvEmpty.
The MessageType parameter determines the type of the message being sent. For the detailed description of the message types see the description of the CREATE MESSAGE TYPE statement in MSDN.

See Also

- TMSConversation.Send
- TMSMessage.IsEmpty

17.20.1.2 MSServiceBroker.TMSMessage Class

A class representing a Service Broker message of Microsoft SQL Server.
For a list of all members of this type, see TMSMessage members.

Unit

MSServiceBroker

Syntax

```
TMSMessage = class(System.TObject);
```

Remarks

The TMSMessage class represents a Service Broker message of Microsoft SQL Server. Use the TMSMessage class to retrieve the message body and parameters from the message obtained from a queue.
All properties of TMSMessage are read-only. Use the TMSConversation.Send method to send a message.
Each message belongs to a conversation.

Inheritance Hierarchy

TObject
  TMSMessage

See Also

- TMSServiceBroker
- TMSServiceBroker.CurrentMessage

TMSMessage class overview.

Properties

Name	Description
AsBytes	Used to read message body as an array of bytes.
AsString	Used to read message body as an ANSI string.
AsWideString	Used to read message body as a Unicode string.
Conversation	Used to get a link to the TMSConversation object that belongs to the message.
IsEmpty	Used to ascertain whether a message contains a value.
MessageId	Used to store a unique identifier for a message.
MessageSequenceNumber	Holds the sequence number of a message within a conversation.
MessageType	Used to store the name of the message type that describes the message.
QueuingOrder	Used to store the order number of a message within a queue.
Validation	Validation for the message to be performed by the server.

Properties of the **TMSMessage** class.
For a complete list of the **TMSMessage** class members, see the TMSMessage Members topic.

Public

Name	Description
AsBytes	Used to read message body as an array of bytes.
AsString	Used to read message body as an ANSI string.
AsWideString	Used to read message body as a Unicode string.
Conversation	Used to get a link to the TMSConversation object that belongs to the message.
IsEmpty	Used to ascertain whether a message contains a value.
MessageId	Used to store a unique identifier for a message.
MessageSequenceNumber	Holds the sequence number of a message within a conversation.
MessageType	Used to store the name of the message type that describes the message.
QueuingOrder	Used to store the order number of a message within a queue.

Validation	Validation for the message to be performed by the server.

See Also

- TMSMessage Class
- TMSMessage Class Members

Used to read message body as an array of bytes.

Class

TMSMessage

Syntax

```
property AsBytes: TBytes;
```

Remarks

Use the AsBytes property to read message body as an array of bytes.

See Also

- AsString
- AsWideString
- TMSConversation.Send
- IsEmpty

Used to read message body as an ANSI string.

Class

TMSMessage

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to read message body as an ANSI string.
Note: If the body of the received message stores a Unicode string value or a TBytes value, the result of AsString will be wrong.

See Also

- AsWideString
- AsBytes
- TMSConversation.Send
- IsEmpty

Used to read message body as a Unicode string.

Class

TMSMessage

Syntax

```
property AsWideString: string;
```

Remarks

Use the AsWideString property to read message body as a Unicode string.
Note: If the body of the received message stores an ANSI string value or a TBytes value, the result of AsWideString will be wrong.

See Also

- AsString
- AsBytes
- TMSConversation.Send
- IsEmpty

Used to get a link to the TMSConversation object that belongs to the message.

Class

TMSMessage

Syntax

```
property Conversation: TMSConversation;
```

Remarks

Use the Conversation property to get a link to the TMSConversation object that belongs to the message.

See Also

- TMSConversation

Used to ascertain whether a message contains a value.

Class

TMSMessage

Syntax

```
property IsEmpty: boolean;
```

Remarks

Check the IsEmpty property to ascertain whether a message contains a value. If IsEmpty is set to True, the message is blank, otherwise the message has a value.

See Also

- TMSConversation.SendEmpty
- AsString
- AsWideString
- AsBytes

Used to store a unique identifier for a message.

Class

TMSMessage

Syntax

```
property MessageId: TGuid;
```

Remarks

Use the MessageId property to store a unique identifier for a message.

See Also

- QueuingOrder
- MessageSequenceNumber

Holds the sequence number of a message within a conversation.

Class

TMSMessage

Syntax

```
property MessageSequenceNumber: Int64;
```

Remarks

Holds the sequence number of a message within a conversation.

See Also

- QueuingOrder
- MessageId

Used to store the name of the message type that describes the message.

Class

TMSMessage

Syntax

```
property MessageType: string;
```

Remarks

Use the MessageType property to store the name of the message type that describes the message.
If the MessageType property was not set when sending, it has the DEFAULT value.

See Also

- TMSConversation.Send

Used to store the order number of a message within a queue.

Class

TMSMessage

Syntax

```
property QueuingOrder: Int64;
```

Remarks

Use the QueuingOrder property to store the order number of a message within a queue.

See Also

- MessageSequenceNumber
- MessageId

Validation for the message to be performed by the server.

Class

TMSMessage

Syntax

```
property Validation: TMSMessageValidation;
```

Remarks

Use the Validation property to validate the message to be performed by the server.

See Also

- QueuingOrder
- MessageSequenceNumber

17.20.1.3 MSServiceBroker.TMSServiceBroker Class

A component that provides sending and receiving messages within the SQL Server 2005 Service Broker system.
For a list of all members of this type, see TMSServiceBroker members.

Unit

MSServiceBroker

Syntax

```
TMSServiceBroker = class(TComponent);
```

Remarks

The TMSServiceBroker component lets you send and receive messages within the SQL Server 2005 Service Broker system.
TMSServiceBroker supports synchronous and asynchronous messages receiving. Each message belongs to a conversation.
Before using the Service Broker system with a database, you should activate Message Delivery .

Inheritance Hierarchy

TObject
   TMSServiceBroker

See Also

- TMSConversation
- TMSMessage
- TCustomMSConnection

TMSServiceBroker class overview.

Properties

Name	Description
AsyncNotification	Used to receive messages asynchronously in a separate thread.
Connection	Used to specify a connection object that will be used to connect to the server.
ConversationCount	Conversations count in the Conversations list.
Conversations	Used for storing the list of available conversations.
CurrentMessage	The current message received by the Receive method.
FetchRows	Used to determine the amount of rows that will be received from the server at a time.
Queue	This property determines queue for the specified service.
Service	Used to set the service name that will be used for sending and receiving messages.
WaitTimeout	Used to specify the time to wait until a message arrives to the server.

Methods

Name	Description
BeginDialog	Overloaded. Initiates a dialog from one TMSServiceBroker.Service to another service.
CreateServerObjects	Creates a service and a queue on the server.
DropServerObjects	Removes both the service with the name assigned to the TMSServiceBroker.Service property and the queue with the name TMSServiceBroker.Service name + ' Queue'.
GetContractNames	Delivers contract name from the server.
GetMessageTypeNames	Delivers the names of message types from the server.
GetQueueNames	Delivers queue names from the server.
GetServiceNames	Delivers queue names from the server.
Receive	Designed for receiving messages from a queue on the server from the specified service.

Events

Name	Description
OnBeginConversation	Occurs when a new object of the TMSConversation class is being created.
OnEndConversation	Occurs when an existing conversation is being closed.

OnMessage	Occurs when a new message is received in the asynchronous mode.

Properties of the **TMSServiceBroker** class.
For a complete list of the **TMSServiceBroker** class members, see the TMSServiceBroker Members topic.

Public

Name	Description
ConversationCount	Conversations count in the Conversations list.
Conversations	Used for storing the list of available conversations.
CurrentMessage	The current message received by the Receive method.
Queue	This property determines queue for the specified service.

Published

Name	Description
AsyncNotification	Used to receive messages asynchronously in a separate thread.
Connection	Used to specify a connection object that will be used to connect to the server.
FetchRows	Used to determine the amount of rows that will be received from the server at a time.
Service	Used to set the service name that will be used for sending and receiving messages.
WaitTimeout	Used to specify the time to wait until a message arrives to the server.

See Also

- TMSServiceBroker Class
- TMSServiceBroker Class Members

Used to receive messages asynchronously in a separate thread.

Class

TMSServiceBroker

Syntax

```
property AsyncNotification: boolean default False;
```

Remarks

If the AsyncNotification property is set to True, messages will be received asynchronously in a separate thread. Each message receiving in asynchronous mode triggers the OnMessage event. For working in asynchronous mode, an addition connection to the server is automatically created.
The default value is False.

See Also

- OnMessage
- Receive

Used to specify a connection object that will be used to connect to the server.

Class

TMSServiceBroker

Syntax

```
property Connection: TMSConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to the server.
Set at design time by selecting from the list of available connection objects.
At runtime, set the Connection property to an instance of a TCustomMSConnection object.

See Also

- TCustomMSConnection

Conversations count in the Conversations list.

Class

TMSServiceBroker

Syntax

```
property ConversationCount: integer;
```

Remarks

Conversations count in the Conversations list.

See Also

- Conversations
- BeginDialog
- CurrentMessage
- Receive
- OnBeginConversation
- OnEndConversation

Used for storing the list of available conversations.

Class

TMSServiceBroker

Syntax

```
property Conversations[Index: Integer]: TMSConversation; default;
```

Parameters

Index

Indicates the index of the conversation object, where 0 is the index of the first object, 1 is the index of the second object, and so on.

Remarks

This property stores the list of available conversations. A new conversation object can be added to this list in two ways:

- when calling the BeginDialog method;
- when receiving an incoming message.

Note: This list is not synchronized with the list on the server. For example, if an initiator has created a conversation instance (by calling the BeginDialog method), a conversation instance on the target side will be created only after receiving the first message.
Analogously, when receiving a message with the SEndDialogType or SErrorType message type, the conversation on the server is already completed. But this conversation will be removed from the list after the next call to the Receive method.

See Also

- ConversationCount
- BeginDialog
- CurrentMessage
- Receive
- OnBeginConversation
- OnEndConversation

The current message received by the Receive method.

Class

`TMSServiceBroker`

Syntax

```
property CurrentMessage: TMSMessage;
```

Remarks

The current message received by the Receive method.

See Also

- Conversations
- BeginDialog
- Receive
- OnBeginConversation
- OnEndConversation

Used to determine the amount of rows that will be received from the server at a time.

Class

`TMSServiceBroker`

Syntax

```
property FetchRows: integer default 0;
```

Remarks

Use the FetchRows property to determine the amount of rows that will be received from the server at a time.

See Also

- Receive
- TMSMessage

This property determines queue for the specified service.

Class

TMSServiceBroker

Syntax

```
property Queue: string;
```

Remarks

This property determines queue for the specified service.

See Also

- Service
- GetQueueNames
- MSDN: CREATE QUEUE

Used to set the service name that will be used for sending and receiving messages.

Class

TMSServiceBroker

Syntax

```
property Service: string;
```

Remarks

Use the Service property to set the service name that will be used for sending and receiving messages.

See Also

- Queue
- GetServiceNames
- TMSConversation.FarService
- MSDN: CREATE SERVICE

Used to specify the time to wait until a message arrives to the server.

Class

TMSServiceBroker

Syntax

```
property WaitTimeout: integer default - 1;
```

Remarks

If the Receive method was called and there are no messages on the server, it will wait until at least one message arrives or WaitTimeout expires. The WaitTimeout is measured in milliseconds.
The possible values of WaitTimeout are the following:

Value	Meaning
-1	Do not wait (the default value).

0	Wait for an infinite interval while at least one message arrives.
1 and more	Wait for the specified interval or until a message arrives.

See Also

- Receive

Methods of the **TMSServiceBroker** class.
For a complete list of the **TMSServiceBroker** class members, see the TMSServiceBroker Members topic.

Public

Name	Description
BeginDialog	Overloaded. Initiates a dialog from one TMSServiceBroker.Service to another service.
CreateServerObjects	Creates a service and a queue on the server.
DropServerObjects	Removes both the service with the name assigned to the TMSServiceBroker.Service property and the queue with the name TMSServiceBroker.Service name + ' Queue'.
GetContractNames	Delivers contract name from the server.
GetMessageTypeNames	Delivers the names of message types from the server.
GetQueueNames	Delivers queue names from the server.
GetServiceNames	Delivers queue names from the server.
Receive	Designed for receiving messages from a queue on the server from the specified service.

See Also

- TMSServiceBroker Class
- TMSServiceBroker Class Members

Initiates a dialog from one Service to another service.

Class

`TMSServiceBroker`

Overload List

Name	Description
BeginDialog(**const** TargetService: string; **const** TargetDatabase: string; **const** UseEncryption: boolean; RelatedConversation: TMSConversation; **const** LifeTime: integer; **const** Contract: string)	Initiates a dialog from one Service to another service.
BeginDialog(**const** TargetService: string; **const** TargetDatabase: string; **const** UseEncryption: boolean; **const** GroupId: TGuid; **const** LifeTime: integer; **const** Contract: string)	Initiates a dialog from one Service to another service.

Initiates a dialog from one TMSServiceBroker.Service to another service.

Class

TMSServiceBroker

Syntax

```
function BeginDialog(const TargetService: string; const
  TargetDatabase: string = ''; const UseEncryption: boolean =
  True; RelatedConversation: TMSConversation = nil; const
  LifeTime: integer = 0; const Contract: string = ''):
  TMSConversation; overload
```

Parameters

TargetService
The target service name that the conversation initiates with.

TargetDatabase
Specifies the database name that the target service hosts.

UseEncryption
Specifies whether messages within the conversation must be encrypted. When set to True (the default value), it may require applying additional server settings.

RelatedConversation
Already existing conversation to whose group a new conversation will be added. If this parameter is not set, a new group will be created.

LifeTime
Specifies the maximum time interval (in seconds) while the dialog remains open. After this interval expires, the dialog automatically closes. A zero LifeTime value represents an infinity interval. This is the default value.

Contract
Specifies the name of the contract that the conversation conforms to. If the parameter is not set, the DEFAULT contract is used.

Return Value

a new instance of the TMSConversation class.

Remarks

These overloaded methods initiate a dialog from one TMSServiceBroker.Service to another service. A dialog is a conversation that provides messaging between two services.
Use BeginDialog to create an instance of the TMSConversation class. The new instance will be created with the TMSConversation.IsInitiator property assigned to True.

See Also

- TMSConversation
- TMSConversation.EndConversation
- TMSConversation.IsInitiator
- TMSServiceBroker.Conversations
- MSDN: BEGIN DIALOG CONVERSATION

Initiates a dialog from one TMSServiceBroker.Service to another service.

Class

TMSServiceBroker

Syntax

```
function BeginDialog(const TargetService: string; const
  TargetDatabase: string; const UseEncryption: boolean; const
  GroupId: TGuid; const LifeTime: integer = 0; const Contract:
  string = ''): TMSConversation; overload
```

Parameters

TargetService
The target service name that the conversation initiates with.

TargetDatabase
Specifies the database name that the target service hosts.

UseEncryption
Specifies whether messages within the conversation must be encrypted. When set to True (the default value), it may require applying additional server settings.

GroupId
Unique identifier of an existent group, which should join in the new conversation.

LifeTime
Specifies the maximum time interval (in seconds) while the dialog remains open. After this interval expires, the dialog automatically closes. A ero LifeTime value represents an infinity interval. This is the default value.

Contract
Specifies the name of the contract that the conversation conforms to. If the parameter is not set, the DEFAULT contract is used.

Return Value

a new instance of the TMSConversation class.

See Also

- TMSConversation
- TMSConversation.EndConversation
- TMSConversation.IsInitiator
- TMSServiceBroker.Conversations
- MSDN: BEGIN DIALOG CONVERSATION

Creates a service and a queue on the server.

Class

TMSServiceBroker

Syntax

```
procedure CreateServerObjects(Contract: string = 'DEFAULT');
```

Parameters

Contract
Specifies the name of the contract that the conversation conforms to. If the parameter is not set, the DEFAULT contract is used.

Remarks

If there are no such objects on the server, the service and the queue will be created on the server. These objects are created with the default settings (any messages can be transferred in any direction). The queue name are generated by concatenating Service and the ' Queue' prefix.

See Also

- DropServerObjects
- Service

- Queue

Removes both the service with the name assigned to the Service property and the queue with the name Service name + ' Queue'.

Class

`TMSServiceBroker`

Syntax

```
procedure DropServerObjects;
```

Remarks

This method removes both the service with the name assigned to the Service property and the queue with the name Service name + ' Queue'.

See Also

- CreateServerObjects
- Service
- Queue

Delivers contract name from the server.

Class

`TMSServiceBroker`

Syntax

```
procedure GetContractNames(List: _TStrings);
```

Parameters

List
Holds the list of contract names.

Remarks

Call the GetContractNames method to get contract names from the server.

See Also

- GetServiceNames
- GetQueueNames
- GetMessageTypeNames
- TMSConversation.ContractName

Delivers the names of message types from the server.

Class

`TMSServiceBroker`

Syntax

```
procedure GetMessageTypeNames(List: _TStrings);
```

Parameters

List

Holds the names of the message types.

Remarks

Call the GetMessageTypeNames method to get the names of message types from the server.

See Also

- GetServiceNames
- GetContractNames
- GetQueueNames
- TMSConversation.Send

Delivers queue names from the server.

Class

TMSServiceBroker

Syntax

```
procedure GetQueueNames(List: _TStrings);
```

Parameters

List
Holds the queue names.

Remarks

Call the GetQueueNames method to get queue names from the server.

See Also

- GetServiceNames
- GetContractNames
- GetMessageTypeNames

Delivers queue names from the server.

Class

TMSServiceBroker

Syntax

```
procedure GetServiceNames(List: _TStrings);
```

Parameters

List
Holds the service names.

Remarks

Call the GetServiceNames method to get service names from the server.

See Also

- GetQueueNames
- GetContractNames
- GetMessageTypeNames
- Service
- TMSConversation.FarService

Designed for receiving messages from a queue on the server from the specified service.

Class

TMSServiceBroker

Syntax

```
function Receive(Conversation: TMSConversation = nil): boolean;
```

Parameters

Conversation
Holds the name of the conversation to recieve messages of.

Return Value

False, if there are no more messages on the server.

Remarks

The Receive method is designed for receiving messages from a queue on the server from the specified service. You can get access to the current message using the CurrentMessage property.
If the Receive method returns False, there are no more messages on the server. So, the CurrentMessage will be nil.

The synchronous mode (AsyncNotification is False)

In this mode, after calling the Receive method, up to FetchRows messages are received from the server. The CurrentMessage property will point to the first message of the received ones. The subsequent calls to Receive will not lead to the server round trips, while there are messages in cache. This mode is enabled by default.
If the Conversation parameter is assigned, only messages of this conversation will be received.

The synchronous mode (AsyncNotification is True)

In this mode messages from the server are received in a separate thread and are put into a local queue. Calls to Receive itself do not lead to the server round trips.
The Conversation parameter can not be used in the asynchronous mode.

Example

Example for using Receive in synchronous mode:

```
while MSServiceBroker.Receive do
Process(MSServiceBroker.CurrentMessage);
```

See Also

- Service
- FetchRows
- CurrentMessage
- AsyncNotification
- TMSMessage
- TMSConversation

Events of the **TMSServiceBroker** class.
For a complete list of the **TMSServiceBroker** class members, see the TMSServiceBroker Members topic.

Published

Name	Description

OnBeginConversation	Occurs when a new object of the TMSConversation class is being created.
OnEndConversation	Occurs when an existing conversation is being closed.
OnMessage	Occurs when a new message is received in the asynchronous mode.

See Also

- TMSServiceBroker Class
- TMSServiceBroker Class Members

Occurs when a new object of the TMSConversation class is being created.

Class

`TMSServiceBroker`

Syntax

```
property OnBeginConversation: TMSConversationBeginEvent;
```

Remarks

The OnBeginConversation event occurs when creating a new object of the TMSConversation class.

See Also

- TMSConversation

Occurs when an existing conversation is being closed.

Class

`TMSServiceBroker`

Syntax

```
property OnEndConversation: TMSConversationEndEvent;
```

Remarks

The OnEndConversation event occurs when closing an existent conversation. The conversation can be closed due to the following reasons:

- The TMSConversation.EndConversation method is called.
- A message with the SEndDialogType or SErrorType message type is received.

See Also

- TMSConversation

Occurs when a new message is received in the asynchronous mode.

Class

`TMSServiceBroker`

Syntax

```
property OnMessage: TMSMessageEvent;
```

Remarks

The OnMessage event occurs when a new message is received in the asynchronous mode (AsyncNotification is set to True).
This event is called in the context of the main thread.

Example

```
procedure TForm1.MSServiceBrokerMessage(Sender: TObject);
  begin
    while MSServiceBroker.Receive do
      Process(MSServiceBroker.CurrentMessage);
  end;
```

See Also

- AsyncNotification
- Receive

17.20.2 Enumerations

Enumerations in the **MSServiceBroker** unit.

Enumerations

Name	Description
TMSMessageValidation	Defines the type of validation performed.

17.20.2.1 MSServiceBroker.TMSMessageValidation Enumeration

Defines the type of validation performed.

Unit

`MSServiceBroker`

Syntax

```
TMSMessageValidation = (mvEmpty, mvNone, mvXML);
```

Values

Value	Meaning
mvEmpty	The message should be empty.
mvNone	Validation is not performed.
mvXML	The message should be a well-formed XML document.

17.21 MSSQLMonitor

This unit contains implementation of the TMSSQLMonitor component.

Classes

Name	Description
TMSSQLMonitor	This component serves for monitoring dynamic SQL execution in SDAC-based applications.

17.21.1 Classes

Classes in the **MSSQLMonitor** unit.

Classes

Name	Description
TMSSQLMonitor	This component serves for monitoring dynamic SQL execution in SDAC-based applications.

17.21.1.1 MSSQLMonitor.TMSSQLMonitor Class

This component serves for monitoring dynamic SQL execution in SDAC-based applications.
For a list of all members of this type, see TMSSQLMonitor members.

Unit

`MSSQLMonitor`

Syntax

```
TMSSQLMonitor = class(TCustomDASQLMonitor);
```

Remarks

Use TMSSQLMonitor to monitor dynamic SQL execution in SDAC-based applications. TMSSQLMonitor 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 TMSSQLMonitor object. If an application has no TMSSQLMonitor instance, the Debug window is available to display SQL statements to be sent.

Inheritance Hierarchy

TObject
  TCustomDASQLMonitor
    TMSSQLMonitor

See Also

- TCustomDADataSet.Debug
- TCustomDASQL.Debug
- DBMonitor

TMSSQLMonitor 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.

17.22 MSTransaction

This unit contains implementation of the TMSTransaction component.

Classes

Name	Description
TMSTransaction	A component for managing transactions in an application.

17.22.1 Classes

Classes in the **MSTransaction** unit.

Classes

Name	Description
TMSTransaction	A component for managing transactions in an application.

17.22.1.1 MSTransaction.TMSTransaction Class

A component for managing transactions in an application.
For a list of all members of this type, see TMSTransaction members.

Unit

`MSTransaction`

Syntax

```
TMSTransaction = class(TCustomMSTransaction);
```

Remarks

The TMSTransaction component is used to provide discrete transaction control over connection. It can be used for manipulating simple local and global transactions. It is based on the Microsoft Distributed Transaction Coordinator functionality.
You can add connections in TMSTransaction both before calling the StartTransaction method, and after that. It means that a transaction can be started before a connection is added. Connections can be added and removed later, when a transaction is active. A transaction is distributed regardless of connection count in it.

Inheritance Hierarchy

TObject
  TDATransaction
    TCustomMSTransaction
      TMSTransaction

See Also

- TMSTransaction Component

TMSTransaction class overview.

Properties

Name	Description
ConnectionsCount	Used to get the number of connections associated with the transaction component.
IsolationLevel	Used to specify how the transactions containing database modifications are handled.

Methods

Name	Description
AddConnection	Binds a TCustomDAConnection object with the transaction component.

RemoveConnection	Disassociates the specified connections from a transaction.

Properties of the **TMSTransaction** class.
For a complete list of the **TMSTransaction** class members, see the TMSTransaction Members topic.

Public

Name	Description
ConnectionsCount	Used to get the number of connections associated with the transaction component.

Published

Name	Description
IsolationLevel	Used to specify how the transactions containing database modifications are handled.

See Also

- TMSTransaction Class
- TMSTransaction Class Members

Used to get the number of connections associated with the transaction component.

Class

TMSTransaction

Syntax

```
property ConnectionsCount: integer;
```

Remarks

Use the ConnectionsCount property for getting the number of connections associated with the transaction component.

Used to specify how the transactions containing database modifications are handled.

Class

TMSTransaction

Syntax

```
property IsolationLevel: TIsolationLevel default ilReadCommitted;
```

Remarks

Use the IsolationLevel property to specify how the transactions containing database modifications are handled.

Methods of the **TMSTransaction** class.
For a complete list of the **TMSTransaction** class members, see the TMSTransaction Members topic.

Public

Name	Description

AddConnection	Binds a TCustomDAConnection object with the transaction component.
RemoveConnection	Disassociates the specified connections from a transaction.

See Also

- TMSTransaction Class
- TMSTransaction Class Members

Binds a TCustomDAConnection object with the transaction component.

Class

TMSTransaction

Syntax

```
function AddConnection(Connection: TCustomDAConnection): integer;
```

Parameters

Connection
Holds a TCustomDAConnection object to associate with the transaction component.

Return Value

the index of associated connection in the connection list.

Remarks

Use the AddConnection method to associate a TCustomDAConnection object with the transaction component.

See Also

- RemoveConnection

Disassociates the specified connections from a transaction.

Class

TMSTransaction

Syntax

```
procedure RemoveConnection(Connection: TCustomDAConnection);
```

Parameters

Connection
Holds the connections to disassociate.

Remarks

Use the RemoveConnection method to disassociate the specified connections from a transaction.

See Also

- AddConnection

17.23 OLEDBAccess

This unit contains classes for accessing SQL Server through OLE DB providers

Classes

Name	Description
EMSError	Raised when SQL Server returns error as a result.
EOLEDBError	Raised when a component receives an OLE DB error.
TMSTableObject	A class for transfering data from the TMSTableData component to a parameter of a stored procedure.

Enumerations

Name	Description
TCompactCommitMode	Specifies the way of buffer pool flushing on transaction commit.
TCompactVersion	Used in TMSCompactEdition to define the version of SQL Server Compact Edition.
TMSAuthentication	Specifies the authentication service used by database server to identify a user.
TMSCursorType	Cursor types supported by SQL Server.
TMSInitMode	Specifies file modes for opening a database file.
TOLEDBOutputEncoding	Used to create coding in TMSSQL. ExecuteForXML.
TOLEDBProvider	Allows to specify a provider from the list of supported providers.

Variables

Name	Description
ParamsInfoOldBehavior	Preparing and the first call of a stored procedure are combined for performance optimization.

17.23.1 Classes

Classes in the **OLEDBAccess** unit.

Classes

Name	Description
EMSError	Raised when SQL Server returns error as a result.
EOLEDBError	Raised when a component receives an OLE DB error.
TMSTableObject	A class for transfering data from the TMSTableData component to a parameter of a stored procedure.

17.23.1.1 OLEDBAccess.EMSError Class

Raised when SQL Server returns error as a result.
For a list of all members of this type, see EMSError members.

Unit

`OLEDBAccess`

Syntax

```
EMSError = class(EOLEDBError);
```

Remarks

EMSError is raised when SQL Server returns error as a result, for example, of an attempt to execute invalid SQL statement. Use EMSError in the exception-handling block.

Inheritance Hierarchy

TObject
  EDAError
    EOLEDBError
      EMSError

See Also

- EOLEDBError
- EDAError

EMSError 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.
ErrorCount (inherited from **EOLEDBError**)	Contains the number of errors returned by the server.
Errors (inherited from **EOLEDBError**)	Contains an array of errors returned by the server.
LastMessage	Contains SQL Server last error message.

LineNumber	Contains the line number of a stored procedure on which the error occurred.
MessageWide (inherited from **EOLEDBError**)	Used to represent the Unicode equivalent of Exception.
MSSQLErrorCode	Contains the code of a SQL Server error.
OLEDBErrorCode (inherited from **EOLEDBError**)	Contains the code of OLE DB Error.
ProcName	Contains the name of the stored procedure that generated the error.
ServerName	Contains the name of the server that generated the error.
SeverityClass	Contains severity of a SQL Server message.
State	Contains the state of a SQL Server error message.

Properties of the **EMSError** class.
For a complete list of the **EMSError** class members, see the EMSError Members topic.

Public

Name	Description
Component (inherited from **EDAError**)	Contains the component that caused the error.
ErrorCode (inherited from **EDAError**)	Determines the error code returned by the server.
ErrorCount (inherited from **EOLEDBError**)	Contains the number of errors returned by the server.
Errors (inherited from **EOLEDBError**)	Contains an array of errors returned by the server.
LastMessage	Contains SQL Server last error message.
LineNumber	Contains the line number of a stored procedure on which the error occurred.
MessageWide (inherited from **EOLEDBError**)	Used to represent the Unicode equivalent of Exception.
MSSQLErrorCode	Contains the code of a SQL Server error.
OLEDBErrorCode (inherited from **EOLEDBError**)	Contains the code of OLE DB Error.
ProcName	Contains the name of the stored procedure that generated the error.
ServerName	Contains the name of the server that generated the error.
SeverityClass	Contains severity of a SQL Server message.
State	Contains the state of a SQL Server error message.

See Also

- EMSError Class
- EMSError Class Members

Contains SQL Server last error message.

Class

EMSError

Syntax

```
property LastMessage: string;
```

Remarks

LastMessage contains SQL Server last error message.

Contains the line number of a stored procedure on which the error occurred.

Class

EMSError

Syntax

```
property LineNumber: WORD;
```

Remarks

When applicable, the LineNumber property contains the line number of a stored procedure on which the error occurred.

See Also

- ProcName

Contains the code of a SQL Server error.

Class

EMSError

Syntax

```
property MSSQLErrorCode: integer;
```

Remarks

Code of the SQL Server error. Refer to MSDN for detail description of errors code. Using MSSQLErrorCode is more preferable than using ErrorCode, as decoding the last one depends on the class of an error (EOLEDBError or EMSError).

See Also

- EMSError
- EDAError.ErrorCode

Contains the name of the stored procedure that generated the error.

Class

EMSError

Syntax

```
property ProcName: string;
```

Remarks

The ProcName property contains the name of the stored procedure that generated the error. This property may be empty if no stored procedure was called.

Contains the name of the server that generated the error.

Class

EMSError

Syntax

```
property ServerName: string;
```

Remarks

The ServerName property contains the name of the server that generated the error.

Contains severity of a SQL Server message.

Class

EMSError

Syntax

```
property SeverityClass: BYTE;
```

Remarks

The SeverityClass property contains severity of a SQL Server message.

Contains the state of a SQL Server error message.

Class

EMSError

Syntax

```
property State: BYTE;
```

Remarks

The State property contains the state of a SQL Server error message. See the SQL Server documentation for more details.

17.23.1.2 OLEDBAccess.EOLEDBError Class

Raised when a component receives an OLE DB error.
For a list of all members of this type, see EOLEDBError members.

Unit

OLEDBAccess

Syntax

```
EOLEDBError = class(EDAError);
```

Remarks

EOLEDBError is raised when a component receives an OLE DB error. Use EOLEDBError in the exception-

handling block.
If several errors happen during execution of the same SQL statement, all these errors are stored into the Errors property. For example, if the following query will be executed:

```
'SELECT WrongField1, WrongField2 FROM Northwind..Orders'
```

ErrorCount equals to 2 and the Errors property contains two errors ('Invalid column name 'WrongField1'.' and 'Invalid column name 'WrongField2'.').
Keep in mind, if MSConnection.Connect was called from another thread than this event, the text of the message can be incomplete.

Inheritance Hierarchy

TObject
  EDAError
    EOLEDBError

See Also

- EMSError
- EDAError

EOLEDBError 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.
ErrorCount	Contains the number of errors returned by the server.
Errors	Contains an array of errors returned by the server.
MessageWide	Used to represent the Unicode equivalent of Exception.
OLEDBErrorCode	Contains the code of OLE DB Error.

Properties of the **EOLEDBError** class.
For a complete list of the **EOLEDBError** class members, see the EOLEDBError Members topic.

Public

Name	Description
Component (inherited from **EDAError**)	Contains the component that caused the error.
ErrorCode (inherited from **EDAError**)	Determines the error code returned by the server.
ErrorCount	Contains the number of errors returned by the server.
Errors	Contains an array of errors returned by the server.
MessageWide	Used to represent the Unicode equivalent of Exception.
OLEDBErrorCode	Contains the code of OLE DB Error.

See Also

- EOLEDBError Class
- EOLEDBError Class Members

Contains the number of errors returned by the server.

Class

`EOLEDBError`

Syntax

```
property ErrorCount: integer;
```

Remarks

The number of errors returned by the server.

See Also

- EMSError

Contains an array of errors returned by the server.

Class

`EOLEDBError`

Syntax

```
property Errors[Index: Integer]: EOLEDBError; default;
```

Parameters

Index
Holds the number of the error to access.

Remarks

The Errors property contains the array of errors returned by the server.

See Also

- EOLEDBError
- EMSError

Used to represent the Unicode equivalent of Exception.

Class

`EOLEDBError`

Syntax

```
property MessageWide: string;
```

Remarks

This property represents the Unicode equivalent of Exception.Message. Useful for the client applications working on systems that have charset incompatible with the server charset.

See Also

- EMSError

Contains the code of OLE DB Error.

Class

EOLEDBError

Syntax

```
property OLEDBErrorCode: integer;
```

Remarks

The OLEDBErrorCode holds the code of OLE DB Error. Refer to MSDN for the detailed description of the error code. Using OLEDBErrorCode is more preferable than using ErrorCode, as decoding the last one depends on the class of an error (EOLEDBError or EMSError).

See Also

- EMSError
- EDAError.ErrorCode

17.23.1.3 OLEDBAccess.TMSTableObject Class

A class for transfering data from the TMSTableData component to a parameter of a stored procedure. For a list of all members of this type, see TMSTableObject members.

Unit

OLEDBAccess

Syntax

```
TMSTableObject = class(TSharedObject);
```

Remarks

Use the TMSTableObject class for transfering data from the TMSTableData component to a parameter of a stored procedure.

Inheritance Hierarchy

TObject
  TSharedObject
    TMSTableObject

See Also

- Using Table-Valued Parameters

TMSTableObject class overview.

Properties

Name	Description
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
TableTypeName	Indicates the table type name of a TMSTableObject 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.

Properties of the **TMSTableObject** class.
For a complete list of the **TMSTableObject** class members, see the TMSTableObject Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
TableTypeName	Indicates the table type name of a TMSTableObject object.

See Also

- TMSTableObject Class
- TMSTableObject Class Members

Indicates the table type name of a TMSTableObject object.

Class

TMSTableObject

Syntax

```
property TableTypeName: string;
```

Remarks

Returns the table type name of a TMSTableObject object.

17.23.2 Enumerations

Enumerations in the **OLEDBAccess** unit.

Enumerations

Name	Description
TCompactCommitMode	Specifies the way of buffer pool flushing on transaction commit.
TCompactVersion	Used in TMSCompactEdition to define the version of SQL Server Compact Edition.
TMSAuthentication	Specifies the authentication service used by database server to identify a user.
TMSCursorType	Cursor types supported by SQL Server.
TMSInitMode	Specifies file modes for opening a database file.
TOLEDBOutputEncoding	Used to create coding in TMSSQL. ExecuteForXML.
TOLEDBProvider	Allows to specify a provider from the list of supported providers.

17.23.2.1 OLEDBAccess.TCompactCommitMode Enumeration

Specifies the way of buffer pool flushing on transaction commit.

Unit

OLEDBAccess

Syntax

```
TCompactCommitMode = (cmAsynchCommit, cmSynchCommit);
```

Values

Value	Meaning
cmAsynchCommit	Asynchronous commit to disk. The default value.
cmSynchCommit	Synchronous commit to disk.

17.23.2.2 OLEDBAccess.TCompactVersion Enumeration

Used in TMSCompactEdition to define the version of SQL Server Compact Edition.

Unit

OLEDBAccess

Syntax

```
TCompactVersion = (cvAuto, cv30, cv35);
```

Values

Value	Meaning
cv30	Use SQL Server Compact Edition version 3.0 or 3.1.
cv35	Use SQL Server Compact Edition version 3.5.
cvAuto	Tries to define the server version using the version of database file. If it is impossible, tries to work as with cv35 value assigned, if this is also impossible - as with cv30.

17.23.2.3 OLEDBAccess.TMSAuthentication Enumeration

Specifies the authentication service used by database server to identify a user.

Unit

OLEDBAccess

Syntax

```
TMSAuthentication = (auWindows, auServer);
```

Values

Value	Meaning
auServer	An alternative way of identifying users by database server. To establish a connection valid TCustomDAConnection.Username and TCustomDAConnection.Password either hardcoded into application or provided in the server login prompt fields are required. The default value.
auWindows	Uses Windows NT/2000/XP integrated security, or "SSPI" (Security Support Provider Interface). The Username, Password, and LoginPrompt properties are ignored.

17.23.2.4 OLEDBAccess.TMSCursorType Enumeration

Cursor types supported by SQL Server.

Unit

OLEDBAccess

Syntax

```
TMSCursorType = (ctDefaultResultSet, ctStatic, ctKeyset,
 ctDynamic, ctBaseTable);
```

Values

Value	Meaning
ctBaseTable	Base table cursor. This cursor is used for working with Compact Edition. This cursor is the fastest of the SQL server cursors and the only cursor that interacts directly with the storage engine. This allows to increase the speed of data access several times. Data modifications, deletions, and insertions by other users are visible. If UniDirectional=False, the cursor is used only when fetching data, and Data updates are reflected on database by SQL statements execution. In order to use the cursor also for data modification it is necessary to set the UniDirectional property to True. But in this case the cursor does not support bookmarks and cannot be represented in multiline controls such as DBGrid.
ctDefaultResultSet	By the old SQL Server terminology is the Firehose cursor. It serves for the fastest data fetch from server to the client side. Allows to run batches. Data updates are reflected in the database only by SQL statements execution. The default value.
ctDynamic	Dynamic cursor. Used when data is not cached at the server and fetch is performed row by row as required. Doesn't support bookmarks and cannot be represented in multiline controls such as DBGrid. Data modifications, deletions, and insertions by other users are visible. Data updates are reflected on database both by SQL statements execution and server cursors means.
ctKeyset	Allows to cache only keyfields at the server. Fetching is performed row by row when a data-aware component or a program requests it. Records added by other users are not visible, and records deleted by other users are inaccessible. Data updates are reflected in the database both by SQL statements execution and server cursors means.

ctStatic	Static copying of records. Query execution results are cashed at the server. Fetch is performed row by row when a data-aware component or a program requests it. When a cursor is opened, all newly added updates are invisible. Used mostly for reporting.

Remarks

ctStatic, ctKeyset, and ctDynamic cursors are server cursors. So the TCustomDADataSet.FetchRows, TCustomMSDataSet.FetchAll, TMemDataSet.CachedUpdates properties don't have any influence on such cursors and only the Options.CursorUpdate option does.
The default value is ctDefaultResultSet.

17.23.2.5 OLEDBAccess.TMSInitMode Enumeration

Specifies file modes for opening a database file.

Unit

OLEDBAccess

Syntax

```
TMSInitMode = (imReadOnly, imReadWrite, imExclusive, imShareRead);
```

Values

Value	Meaning
imExclusive	Database file is opened for exclusive use. This mode prevents others from opening this database file.
imReadOnly	Database file is opened for reading. Any modification operations are not allowed.
imReadWrite	Both read and write operations are allowed. The default value.
imShareRead	The database file is opened for reading and writing only by one user. Other users can not read or write to the database file.

17.23.2.6 OLEDBAccess.TOLEDBOutputEncoding Enumeration

Used to create coding in TMSSQL.ExecuteForXML.

Unit

OLEDBAccess

Syntax

```
TOLEDBOutputEncoding = (oeANSI, oeUTF8, oeUnicode);
```

Values

Value	Meaning
oeANSI	1-byte ANSI-coding is used.
oeUnicode	2-byte Unicode-coding is used.
oeUTF8	VariableLength-coding UTF8 is used.

17.23.2.7 OLEDBAccess.TOLEDBProvider Enumeration

Allows to specify a provider from the list of supported providers.

Unit

OLEDBAccess

Syntax

```
TOLEDBProvider = (prAuto, prSQL, prNativeClient, prCompact);
```

Values

Value	Meaning
prAuto	The available provider with the most recent version is chosen. The default value.
prCompact	SQL Server Compact Edition provider.
prNativeClient	Uses SQL Native Client. It should be installed on the computer to use this Provider value. This provider offers maximum functionality set.
prSQL	Uses the provider preinstalled with Windows which has limited functionality.

Remarks

The default value of this property is prAuto. In this case the most recent version of a provider is used. Some features in SQL Server require the SQL Native Client (prNativeClient) provider to be used. If chosen provider is not installed, exception is raised. prCompact value should be set for working with SQL Server Compact Edition.

17.23.3 Variables

Variables in the **OLEDBAccess** unit.

Variables

Name	Description
ParamsInfoOldBehavior	Preparing and the first call of a stored procedure are combined for performance optimization.

17.23.3.1 OLEDBAccess.ParamsInfoOldBehavior Variable

Preparing and the first call of a stored procedure are combined for performance optimization.

Unit

OLEDBAccess

Syntax

```
ParamsInfoOldBehavior: boolean;
```

Remarks

Starting with SDAC 3.70.1.26 preparing and the first call of a stored procedure were combined for performance optimization. This requires the necessity of setting the parameter type and data type of all parameters before preparing. In order to revert the old behaviour with preparation and parameters, the OLEDBAccess unit should be added to the uses clause of a unit in an application, and the following line should be added to the initialization section of the unit:

```
ParamsInfoOldBehavior := True.
```

17.24 SdacVcl

This unit contains the visual constituent of SDAC.

Classes

Name	Description
TMSConnectDialog	A class that provides a dialog box for user to supply his login information.

17.24.1 Classes

Classes in the **SdacVcl** unit.

Classes

Name	Description
TMSConnectDialog	A class that provides a dialog box for user to supply his login information.

17.24.1.1 SdacVcl.TMSConnectDialog Class

A class that provides a dialog box for user to supply his login information.
For a list of all members of this type, see TMSConnectDialog members.

Unit

SdacVcl

Syntax

```
TMSConnectDialog = class(TCustomConnectDialog);
```

Remarks

The TMSConnectDialog component is a direct descendant of TCustomConnectDialog class. Use TMSConnectDialog to provide dialog box for user to supply server name, user name, and password. You may want to customi e appearance of dialog box using this class's properties.

Inheritance Hierarchy

TObject
  TCustomConnectDialog
    TMSConnectDialog

See Also

- TCustomDAConnection.ConnectDialog

TMSConnectDialog 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.
Connection	Contains the TCustomMSConnection that is used by TMSConnectDialog object.
DialogClass (inherited from **TCustomConnectDialog**)	Used to specify the class of the form that will be displayed to enter login information.
LabelSet (inherited from **TCustomConnectDialog**)	Used to set the language of buttons and labels captions.
PasswordLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for password edit.

Retries (inherited from **TCustomConnectDialog**)	Used to indicate the number of retries of failed connections.
SavePassword (inherited from **TCustomConnectDialog**)	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for the server name edit.
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	Retrieves the list of available database servers.

Properties of the **TMSConnectDialog** class.
For a complete list of the **TMSConnectDialog** class members, see the TMSConnectDialog 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.
Connection	Contains the TCustomMSConnection that is used by TMSConnectDialog object.
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.

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.

See Also

- TMSConnectDialog Class
- TMSConnectDialog Class Members

Contains the TCustomMSConnection that is used by TMSConnectDialog object.

Class

TMSConnectDialog

Syntax

```
property Connection: TCustomMSConnection;
```

Remarks

Read Connection property to find out what TCustomMSConnection uses the TMSConnectDialog object. This property is read-only.

See Also

- TCustomDAConnection.ConnectDialog

Methods of the **TMSConnectDialog** class.
For a complete list of the **TMSConnectDialog** class members, see the TMSConnectDialog 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	Retrieves the list of available database servers.
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.
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.

See Also

- TMSConnectDialog Class
- TMSConnectDialog Class Members

Retrieves the list of available database servers.

Class

`TMSConnectDialog`

Syntax

```
procedure GetServerList(List: _TStrings); override;
```

Parameters

List
A TStrings descendant that will be filled with database servers names.

Remarks

Call GetServerList method to retrieve the list of available database servers. It is useful for writing custom login forms.

17.25 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.

17.25.1 Classes

Classes in the **VirtualTable** unit.

Classes

Name	Description
TVirtualTable	A base class for storing data in memory.

17.25.1.1 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 SQL Server Access page.

Inheritance Hierarchy

TObject
  TMemDataSet
    TVirtualTable

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**)	Determines whether a query is prepared for execution or not.
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
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.

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

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.

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

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 specifiad 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

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

```
MSQuery1.SQL.Text := 'SELECT * FROM Orders';
MSQuery1.Active := True;
VirtualTable1.Assign(MSQuery1);
VirtualTable1.Active := True;
```

See Also

- TVirtualTable

Removes all records from TVirtualTable.

Class

TVirtualTable

Syntax

```
procedure Clear;
```

Remarks

Call the Clear method to remove all records from TVirtualTable.

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

Deletes all fields.

Class

`TVirtualTable`

Syntax

```
procedure DeleteFields;
```

Remarks

Call the DeleteFields method to delete all fields.

See Also

- DeleteField

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.

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.

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.

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.

17.25.2 Types

Types in the **VirtualTable** unit.

Types

Name	Description
TVirtualTableOptions	Represents the set of TVirtualTableOption.

17.25.2.1 VirtualTable.TVirtualTableOptions Set

Represents the set of TVirtualTableOption.

Unit

`VirtualTable`

Syntax

```
TVirtualTableOptions = set of TVirtualTableOption;
```

17.25.3 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.

17.25.3.1 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.

Index