Table of Contents

Index
0

1   What's New

New Features in MyDAC 13.0

- Added support for RAD Studio 13 Florence

- Added support for Lazarus 4.2

- Added support for MariaDB 12

- Added extended support for the Vector data type

- Added support for the UUID data type in MariaDB 10.7 and later

- Added support for the INET4 and INET6 data types in MariaDB 10.10 and later

- Added support for Bearer Token authentication when connecting through an HTTP tunnel

- Added the InsertBatchSize option for the TMyDump component to set the size of the SQLInsert query

- Added the AnsiQuotesMode option to set the ANSI_QUOTES value for the sql_mode variable

- Added support for system proxy settings to the TProxyOptions class for HTTP tunnelling

- Added the Pooled function to the TCustomDAConnection component

- Added support for ReportBuilder 23 in the demo project

- Improved trigger backup with TMyDump

New Features in MyDAC 12.4

- Added support for RAD Studio 12 Athens Release 3

- Added support for the RAD Studio 64-bit IDE

- Added support for Lazarus 3.8

- Added support for MySQL 9

- Added support for parameters in the Script component

- Added support for the TurboPack SynEdit package in component editors

- Added support for syntax highlighting in component editors in Lazarus on the Windows platform

- Added the IndentFormat argument to the SaveToXML method of the dataset to decrease file size

- Added the SaveToJSON method for serializing a dataset to the JSON format

- Added the ability to use readonly fields in the SQL Generator at design-time

- Added support for password-protected private key for SSL protocol

- Added the EMySqlNetException class for internal errors in the MySQL protocol

- Improved work with design-time editors in the High DPI mode

New Features in MyDAC 12.3

- Added support for RAD Studio 12 Athens Release 2

- Added support for C++Builder 64-bit Windows (Modern) Application Development

- Added support for Lazarus 3.6

- Added C++Builder demo components for working with SecureBridge

- Added demo with the new version of SecureBridge components

- Added the WriteBOM property for TMyDumpOptions

New Features in MyDAC 12.2

- Added support for RAD Studio 12 Athens Release 1

- Added support for Lazarus 3.2

- Now Release 1 for RAD Studio 12 Athens, Delphi 12 Athens, and C++Builder 12 Athens is required

New Features in MyDAC 12.1

- Lazarus 3.0 is supported

New Features in MyDAC 12.0

- Added support for RAD Studio 12

- Added support for MariaDB 11

- Added support for macOS Sonoma

- Added support for iOS 17

- Added support for Android 13

- Added support for the mysql_clear_password authentication mechanism

- Added support for nested Macros in SQL queries

- Added support Display Format for Aggregate fields

- Added SHA-2(SHA-256, SHA-512) in hash algorithm for encryption

New Features in MyDAC 11.3

- Added support for RAD Studio 11 Alexandria Release 3

- Added support for iOS Simulator ARM 64-bit target platform

- Added support for Lazarus 2.2.6

- Added support for the YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, GETDATE, DATE, TIME, TRIM, TRIMLEFT, TRIMRIGHT statements in TDADataSet.Filter

- Added support for the mathematical operations in TDADataSet.Filter

- Added support for Aggregate Fields and InternalCalc Fields

- Added ability to restore from file with TEncoding via the Dump component

- Now the SetRange will function according to the case sensitivity of keywords in IndexFieldNames

New Features in MyDAC 11.2

- Added support for RAD Studio 11 Alexandria Release 2

- Added support for Lazarus 2.2.2

- Added support for iOS 15

- Added support for Android 12

- Added the CloneCursor method for Query and Table components that allows sharing data between datasets

- Improved the performance of exporting to XML

- Fixed bug with "Record changed by another user" error when LockMode <> lmNone and using encryption

- Fixed bug with truncation of milliseconds of DateTime fields in the Dump component

- Fixed bug with posting calculated field

- Fixed bug when a connection string parameter value contains a single quote

New Features in MyDAC 11.1

- RAD Studio 11 Alexandria Release 1 is supported

- Lazarus 2.2.0 is supported

- Windows 11 is supported

- macOS Monterey is supported

New Features in MyDAC 11.0

- RAD Studio 11 Alexandria is supported

- macOS ARM is supported

- Added demo project for FastReport FMX

New Features in MyDAC 10.4

- RAD Studio 10.4.2 Sydney is supported

- macOS 11 Big Sur is supported

- iOS 14 is supported

- Android 11 is supported

- The LastWarningCount property in TMyConnection is added

- The OnWarning event in TMyConnection is added

- Performance of batch operations is improved

- Performance of the FindFirst, FindNext, FindLast, and FindPrior methods is improved

New Features in MyDAC 10.3

- Lazarus 2.0.10 and FPC 3.2.0 are supported

- Performance of Batch Insert, Update, and Delete operations is improved

New Features in MyDAC 10.2

- RAD Studio 10.4 Sydney is supported

- Lazarus 2.0.8 is supported

- macOS 64-bit in Lazarus is supported

New Features in MyDAC 10.1

- Android 64-bit is supported

- Lazarus 2.0.6 is supported

- OpenSSL 1.1 library is supported

- Now Trial edition for macOS and Linux is fully functional

- A package containing the TMySSHIOHandler and TMySSLIOHandler components for Lazarus is added

- The TMyConnection.GetStoredProcNames and TMyConnection.GetTriggerNames methods are added

New Features in MyDAC 10.0

- macOS 64-bit is supported

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

New Features in MyDAC 9.4

- Lazarus 2.0.2 is supported

- Performance of TMyDump for data dumping from a query is improved

- Performance of describing stored procedures is improved

- Multi-threading support is improved

- The DefaultSortType property for TVirtualTable is added

- Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

New Features in MyDAC 9.3

- RAD Studio 10.3 Rio is supported

- Support for PAM and Windows authentications is added

- Support of UPPER and LOWER functions for Unified SQL is added

- TMyConnection.OnDialogAuthPlugin event is added

New Features in MyDAC 9.2

- Lazarus 1.8.4 is supported

- MySQL 8 is supported

- Support for sha2_password, caching_sha2_password authentications is added

- Performance of batch operations is improved

- Demo projects for IntraWeb 14 are added

New Features in MyDAC 9.1

- Azure Database for MySQL is supported

- JSON data type is supported

New Features in MyDAC 9.0

- RAD Studio 10.2 Tokyo is supported

- Linux in RAD Studio 10.2 Tokyo is supported

- Lazarus 1.6.4 and Free Pascal 3.0.2 is supported

- Support for HTTPS protocol is added

New Features in MyDAC 8.7

- RAD Studio 10.1 Berlin is supported

- Lazarus 1.6 and FPC 3.0.0 is supported

- Support for the BETWEEN statement in TDADataSet.Filter is added

- Support for utf8mb4 charset is added

- SmartFetch mode in Disconnected mode is supported

- Data Type Mapping performance is improved

- Performance of TDALoader on loading data from TDataSet is improved

New Features in MyDAC 8.6

- RAD Studio 10 Seattle is supported

- Now Trial for Win64 is a fully functional Professional Edition

- INSERT, UPDATE and DELETE batch operations are supported

- MariaDB Embedded is supported

New Features in MyDAC 8.5

- RAD Studio XE8 is supported

- AppMethod is supported

New Features in MyDAC 8.4

- RAD Studio XE7 is supported

- Lazarus 1.2.4 is supported

- The TCustomDADataSet.GetKeyFieldNames method is added

- The ConstraintColumns metadata kind for the TMyMetaData component is added

- Demo projects for FastReport 5 are added

New Features in MyDAC 8.3

- RAD Studio XE6 is supported

- Android in C++Builder XE6 is supported

- Lazarus 1.2.2 and FPC 2.6.4 is supported

- SmartFetch mode for TDataSet descendants is added

- The TMyDataSetOptions.MasterFieldsNullable property is added

- Now update queries inside TDataSet descendants have correct owner

New Features in MyDAC 8.2

- iOS in C++Builder XE5 is supported

- RAD Studio XE5 Update 2 is now required

- Now .obj and .o files are supplied for C++Builder

- Compatibility of migrating floating-point fields from other components is improved

New Features in MyDAC 8.1

- RAD Studio XE5 is supported

- Application development for Android is supported

- Lazarus 1.0.12 is supported

- IPv6 protocol support is added

- MariaDB is supported

- Performance is improved

- Automatic checking for new versions is added

- Flexible management of conditions in the WHERE clause is added

- The possibility to use conditions is added

- Support of the IN keyword in the TDataSet.Filter property is added

- Like operator behaviour when used in the Filter property is now similar to TClientDataSet

- The possibility to use ranges is added

- The AllowImplicitConnect option for the TMyConnection component is added

- The SQLRecCount property for the TMyQuery and TMyStoredProc components is added

- The ScanParams property for the TMyScript component is added

- The RowsAffected property for the TMyScript component is added

- ConnectionTimeout is now used when disconnecting after connection loss

New Features in MyDAC 8.0

- Rad Studio XE4 is supported

- NEXTGEN compiler is supported

- Application development for iOS is supported

- FPC 2.6.2 and Lazarus 1.0.8 are supported

- Connection string support is added

- Possibility to encrypt entire tables and datasets is added

- Possibility to determine if data in a field is encrypted is added

- Support for TimeStamp, Single and Extended fields in VirtualTable is added

New Features in MyDAC 7.6

- Rad Studio XE3 Update 1 is now required

- C++Builder 64-bit for Windows is supported

- TMyDumpOptions.FieldsAsString property for dumping all data types as string is added

New Features in MyDAC 7.5

- Rad Studio XE3 is supported

- Windows 8 is supported

New Features in MyDAC 7.2

- 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 TMyEncryptor component for data encryption is added

- Calling of the TCustomDASQL.BeforeExecute event is added

New Features in MyDAC 7.1

- Update 4 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required

- Mac OS X and iOS in RAD Studio XE2 is supported

- FireMonkey support is improved

- Lazarus 0.9.30.4 and FPC 2.6.0 are supported

- Mac OS X in Lazarus is supported

- Linux x64 in Lazarus is supported

- FreeBSD in Lazarus is supported

- Performance of SQL query generation for stored procedure execution is improved

New Features in MySQL Data Access Components 7.00

- Embarcadero RAD Studio XE2 is supported

- Application development for 64-bit Windows is supported

- FireMonkey application development platform is supported

- Support of master/detail relationship for TVirtualTable is added

- OnProgress event in TVirtualTable is added

- TDADataSetOptions.SetEmptyStrToNull property that allows inserting NULL value instead of empty string is added

New Features in MySQL Data Access Components 6.10

- Lazarus 0.9.30 and FPC 2.4.2 is supported

- Now the BreakExec method also stops working when getting record count if QueryRecCount=True

New Features in MySQL Data Access Components 6.00

- Embarcadero RAD Studio XE suppored

New Features in MySQL Data Access Components 5.90

- Embarcadero RAD Studio 2010 supported

New Features in MySQL Data Access Components 5.80

- Free Pascal under Linux supported

- Added NoPreconnect property to TMyScript for executing CONNECT and CREATE DATABASE commands

New Features in MySQL Data Access Components 5.70

- 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

New Features in MySQL Data Access Components 5.55
- dbForge Fusion for MySQL support added

New Features in MySQL Data Access Components 5.50
- Added compatibility with UniDAC

- Improved support of default field values

- The new component for metadata receiving added

- Added ability to specify key fields for a dataset

- Added support of automatic records locking

New Features in MySQL Data Access Components 5.20
- CodeGear RAD Studio 2007 supported

- Added the OnProgress event in TMyLoader

New Features in MySQL Data Access Components 5.10
- C++Builder 2007 supported

New Features in MySQL Data Access Components 5.00

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

- Support for SSH protocol via SecureBridge component set added

- Added DataSet Manager to control project datasets

- Integration with MyDeveloper Tools 2.00 added

- New TCRBatchMove component for transferring data between all types of TDataSet
  descendants added

- Output parameters from stored procedures and functions retrieval supported

- Data export and import to/from XML supported

- WideMemo field type in Delphi 2006 supported

- AutoRefresh mode support added

- Option to break long-duration query execution added

- Ability to search for installed MySQL servers on the network added

- Support for sending messages to DBMonitor from any point in your program added

Support for more MySQL server functionality:

- HANDLER syntax support in TMyTable added

- Enumeration value retrieval for ENUM and SET fields added

Extensions and improvements to existing functionality:

- General performance improved

- Master/detail functionality extensions:

- Local master/detail relationship support added

- Support for master/detail relationships in CachedUpdates mode added

- TMyScript component improvements:

- Support for executing individual statements in scripts added

- Support for executing huge scripts stored in files with dynamic loading added

- Ability to use standard MySQL client tool syntax 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

- Option to [ignore or replace records](#) with duplicated key values in [TMyLoader](#) added

- Enhanced [TMyServerControl](#) functionality for working with [server values](#)

- Ability to customize update commands by attaching external components to [TMyUpdateSQL](#) objects added

- Ability to [include all fields](#) in automatically generated update SQLs added

Usability improvements:

- [Syntax highlighting](#) in design-time editors added

- Completely restructured and clearer [demo projects](#)

New Features in MySQL Data Access Components 4.40

- Optimized TDALoader.CreateColumns method

- Support for Professional editions of Turbo Delphi, Turbo Delphi for .NET, Turbo C++ added

- Added support for macros names in which first symbol is digit

- Added capability to use quoted field names in IndexFieldNames property

New Features in MySQL Data Access Components 4.30.1

- MyDAC is now compatible with InterBase Data Access Components

- Modifying FieldDefs in TVirtualTable component accelerated

- Performance of SaveToFile and LoadFromFile functions in TVirtualTable improved

New Features in MySQL Data Access Components 4.30

- Support for Delphi 2006 added

- BIT fields of MySQL 5.0 and above are now represented as TLargeintField

- FastReport 3.20 support added

- Added server version checking on Prepare method

- Added capability to close DataSet quicker when FetchAll property is False

- Improved performance of updating recordsets with multiple fields

- TCustomDADataSet.Locate now centers position in DBGrid

- Added support for MIDAS TDataSet.PSExecuteStatement ResultSet parameter

New Features in MySQL Data Access Components 4.00.2

- Added 'delimiter' keyword support in TMyScript

- TCustomDADataSet.FindKey, TCustomDADataSet.FindNearest methods added for BDE compatibility

- Added BIT and INTEGER types support in parameters of stored procedures

New Features in MySQL Data Access Components 4.00.1

- Deferred detail dataset refresh feature with TCustomDADataSet.Options.DetailDelay property added

- TCustomMyConnection.Ping behavior in case connection loss improved

- FieldDefs.Update behavior in case of temporary tables improved

- Added capability to prevent simultaneous access of several MyEmbConnection instances to single data folder

New Features in MySQL Data Access Components 4.00

- Unicode support added

- Enhanced support for Embedded MySQL Server with TMyEmbConnection component added

- Binary protocol support for MySQL 4.1 and above added

- Encrypted SSL connections support with OpenSSL added

- Enhanced support for national charsets added with TMyConnectionOptions.Charset property

- BLOB compression support added

- RefreshQuick for TCustomMyDataSet added

- Retrieve field's default value added with TCustomMyDataSet.Options.DefaultValues
  property

- Large amount of data support for TMyDump added

- Server start/stop ability for TMyServerControl added

- TMyBuilder component added for easy using SQL Builder for MySQL at run-time

- Optimized macros processing

- FAQ added

- Tested with MySQL server 5.0.9

New Features in MySQL Data Access Components 3.55

- MySQL 5.0.3 BIT type support added

- Optimized MySQLMonitor BLOB parameters processing

- Ability of automatic preparing query with TCustomDADataSet.Options.AutoPrepare property
  added

- Ability to synchronize position at different DataSets with TCustomDADataSet.GotoCurrent
  method added

21-Jan-05 MySQL Data Access Components 3.50:

- Support for Delphi 2005 added

- TMySQLMonitor.OnSQL can return statement encoded to an escaped SQL string

- Support for ConnectionTimeout in TMyConnection.ExecSQL added

- CommandTimeout default value set to 0 (infinite)

- TCustomDADataSet.UpdateObject support for MIDAS added

- Lock Demo added

- DECIMAL column type in MySQL 5.0.3 support added

- Update Pack 3 is required for Delphi 8

New Features in MySQL Data Access Components 3.30

- Full support for all current authentication protocols added

- Generating update SQL for tables from other database added

- TCustomMyDataSet.Options.EnableBoolean property added

- TMyConnection.ThreadId property added

- lxPartialCompare option for DataSet.LocateEx added

- FastReport3 engine and demo added

- Ability to store only a part of data in TMyDump.BackupQuery added

- Creating additional connection for TMyDump disabled

- TCustomMyDataSet.CommandTimeout property added

- "True" value for boolean fields and parameters stored as "1"

New Features in MySQL Data Access Components 3.10.2

- Common class DADataAdapter isolated to Devart.Dac.AdoNet.dll library

New Features in MySQL Data Access Components 3.10.1 new features

- Assembly Devart.MyDac.Data renamed to Devart.MyDac.AdoNet

- Fatal errors processing improved

- TINYINT(1) fields now represented as TBooleanField

New Features in MySQL Data Access Components 3.10

- Local sorting ability with TMemDataSet.IndexFieldNames added

- TCustomMyTable.IndexDefs property added

- TMyConnection.Options.NumericType property added

- TMyStoredProc component added

- MyDataAdapter component added

New Features in MySQL Data Access Components

3.00.1

- TCustomMyDataSet.Options.LongStrings property added

- TMyLoader.OnPutData event published again

- Trial version IDE warning disabled

- TCRColumn.TotalValue property added

New Features in MySQL Data Access Components 3.00

- Support for Delphi 8 added

- Connection pooling support

- Performance improved

- TMyLoader performance greatly improved

- TCRGrid sources in Standard edition

- .NET Windows Forms demo project added

- ASP.NET demo project added

- Global variable MySQLClientLibrary added

- New time trial limitation

New Features in MySQL Data Access Components 2.00.3

- SELECT '' support added

- Method TMyConnection.Ping added

- Method TMyConnection.GetExecuteInfo added

- Mouse wheel support added to CRDBGrid

- Embedded MySQL Server Demo added

- ConnectDialog Demo added

New Features in MySQL Data Access Components 2.00.2

- BDE Migration Wizard algorithm optimized

- Limited MySQL server 4.1.1 support added

- If libmysql.dll not found then raise EOSError (instead of Exception)

- Property TCustomMyDataSet.InsertId: int64 added

- timestamp support added for CheckRowVersion = True

New Features in MySQL Data Access Components 2.00.1

- Property MyConnection.Options.Direct is set to True by default

- TCustomMyDataSet.Lock method added

- AutoInc fields can be modified now

New Features in MySQL Data Access Components 2.00

- Access to MySQL without client library using DirectMySQLObjects by Cristian Nicola

- Prepare support and new parameter binding schema for MySQL 4.1 added

- Supports working with MySQL server and Embedded server at the same time

- BDE migration wizard

- TMyDump component to store a database or its parts as a script

- TMyBackup component for backup coping specified tables on the server

- TMyServerControl component to manage the server and standard service tasks execution

- TMyLoader component for fast loading data to the server

- New options of TMyConnection such as Compress, Protocol, Direct and Embedded added

- New properties ClientVersion, ServerVersion were added to TMyConnection

- Method ExecSQL in TMyConnection added

- Methods GetTableNames and GetDatabaseNames in TMyConnection added

- Property TMyConnection.Charset added

- Property TMyConnection.IsolationLevel added

- Methods LockTable and UnlockTable added to TCustomMyDataset

- Properties Limit and Offset added to TCustomMyTable

- Method TCustomMyTable.EmptyTable added

- FetchAll set to True by default

- Large SQL (INSERT/UPDATE BLOB's) executing performance greatly improved

New Features in MySQL Data Access Components 1.50

- Embedded MySQL Server support added

- MySQL Server 4.1 limited support added

- Properties Port and Database in ConnectForm added

- RefreshRecord performance improved

- InfoPower demos added

- 'Explain query...' added to design-time MyQuery menu

- 'Show CREATE...' added to design-time MyQuery and MyTable menus

- SQL Generator improved - support for complicated statements added

- SQL Generator improved - "Quote names" checkbox added

- Complex keys support added

- Design-time SQL Generator was simplified

- TParam -> TDAParam

- Embedded MySQL Server support added for Kylix

- Check for datadir present added

- Changed behavior on calculating affected rows count

New Features in MySQL Data Access Components 1.30.2

- Unit MySQLAccess renamed to MyClasses

- Property TMyDataSetOptions.LongStrings removed

- Parameters parsing improved. Symbol ':' in string literals is ignored

- Search algorithm for 'libmysqlclient.so' under Linux improved

New Features in MySQL Data Access Components 1.30.1

- Refresh improved - current record is restored after Refresh call

- Property MyConnection.Options.KeepDesignConnected added

- Property MyConnectDialog.StoreLogInfo published

- Property MyScript.DataSet was published

- Property TMyCommand.InsertId: int64 added

- TINYTEXT -> TMemoField, TINYBLOB -> TBlobField

- Support for TIMESTAMP (10), TIMESTAMP (4), TIMESTAMP (2) added

- Support for LIKE expressions in Filter property added (D2706)

New Features in MySQL Data Access Components 1.30

- MySQL v4.0 support added

- Dataset 'with many fields' update performance improved

- Improved performance for opening queries with lot of parameters

New Features in MySQL Data Access Components 1.20

- Kylix2 and Kylix3 support

- ReportBuilder demos added

- DBMonitor client implementation moved to COM server

- Fetch performance improved for DataSet.FetchAll = True

- 'Connection Lost' error processing improved

New Features in MySQL Data Access Components 1.10

- Delphi 7 support

- New memory management model for ftString and ftVarBytes types. Allows significantly decrease memory usage on large tables fetch. Controlled by FlatBuffers dataset option

- Support for blob fields in CachedUpdates mode

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Devart. All Rights
Reserved.

2 General Information

This section contains general information about MySQL Data Access Components

- [Overview](#)
- [Features](#)
- [Requirements](#)
- [Compatibility](#)
- [Using Several DAC Products in One IDE](#)
- [Component List](#)
- [Hierarchy Chart](#)
- [Editions](#)
- [Licensing and Subscriptions](#)
- [Getting Support](#)
- [Frequently Asked Questions](#)

© 1997-2025
Devart. All Rights   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)
Reserved.

2.1 Overview

MySQL Data Access Components (MyDAC) is a library of components that provides direct access to MySQL database servers from Delphi, C++Builder, and Lazarus (Free Pascal). MyDAC can connect directly to MySQL server or work through the MySQL client library. The MyDAC library is designed to help programmers develop faster and cleaner MySQL database applications. MyDAC is a complete replacement for standard MySQL connectivity solutions and presents an efficient alternative to the Borland Database Engine for access to MySQL.

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

Advantages of MyDAC Technology

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

Wide Coverage of MySQL Features

By providing access to the most advanced database functionality, MyDAC allows developers to harness the full capabilities of the MySQL server and optimize their database applications. MyDAC provides a complete support of MySQL Embedded Server, row-level locking, using HANDLER statements, MySQL administration tasks. View the full list of supported MySQL features in Features.

Native Connection Options

MyDAC offers two connection modes to the MySQL server: Direct connection and connection through the standard MySQL Client in Client mode. MyDAC-based database applications are easy to deploy, do not require installation of other data provider layers (such as BDE), and tend to be faster than those that use standard data connectivity solutions. See the How does MyDAC work section.

Optimized Code

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

Compatibility with other Connectivity Methods

The MyDAC interface retains compatibility with standard VCL data access components BDE. Existing BDE-based applications can be easily migrated to MyDAC and enhanced to take advantage of MySQL-specific features. Migration of a Delphi project can be automated with the BDE Migration Wizard. Find out more about Migration Wizard in Migration from BDE.

Development and Support

MyDAC is a MySQL connectivity solution that is actively developed and supported. MyDAC comes with full documentation, demo projects, and fast (usually within one business day) technical support by the MyDAC development team. Find out more about how to get help or submit feedback and suggestions to the MyDAC Development Team in Getting Support.

A description of the MyDAC components is provided in Component List.

Key Features

- Direct access to server data without using client library. Does not require installation of other data provider layers (such as BDE and ODBC)

- VCL, LCL and FMX versions of library available

- Full support of the latest versions of MySQL Server

- Support for all MySQL 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 TMyQuery, TMyTable, and TMyStoredProc components

- Unicode and national charset support

- Supports many MySQL-specific features, such as locking, SET and ENUM types

- Advanced script execution functionality with TMyScript component

- Support for using macros in SQL

- Easy migration from BDE with Migration Wizard

- Lets you use Professional Edition of Delphi and C++Builder to develop client/server applications

- Included annual MyDAC Subscription with Priority Support

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

The full list of MyDAC features is available in Features.

How does MyDAC work?

MyDAC allows you to connect to MySQL in two ways: in the Direct mode over TCP/IP or in

the Client mode using MySQL client software. MyDAC works directly through the native MySQL interface to avoid using BDE and ODBC. The Direct mode can be enabled or disabled using the Direct option.

In the Direct mode, MyDAC connects to MySQL directly, without using the MySQL client software.

In the Client mode, MyDAC connects to MySQL using the MySQL client library. The MySQL client library is supplied with MySQL server.

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

MyDAC Connection

BDE Connection

Request Support   DAC Forum   Provide Feedback
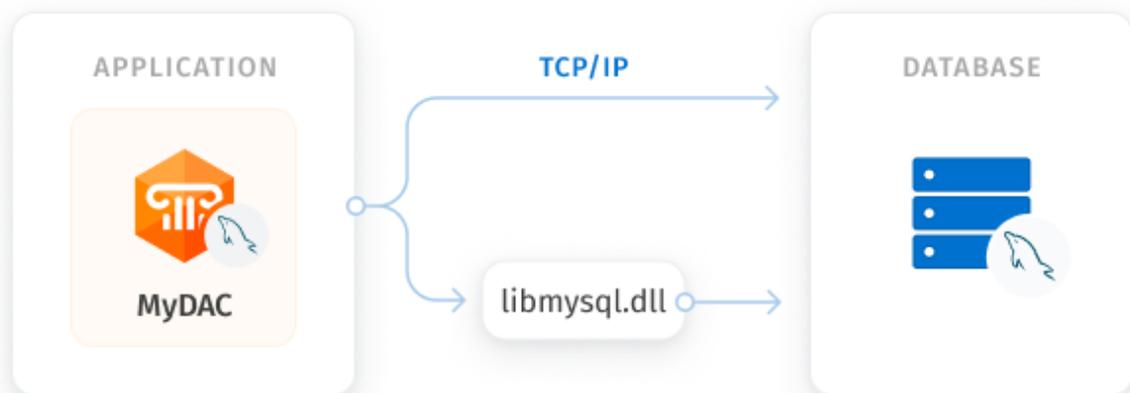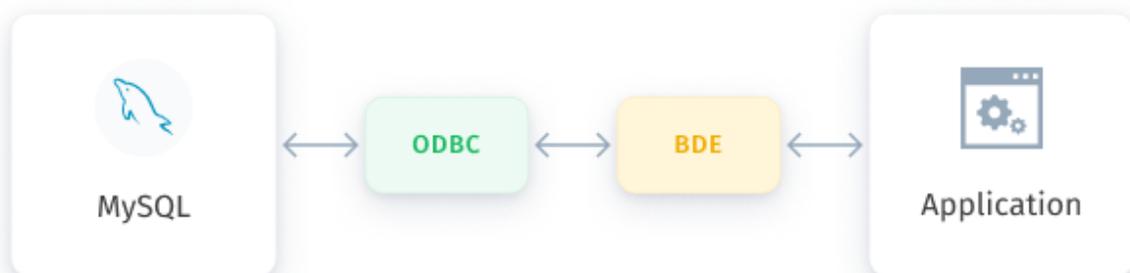
Reserved.

2.2   Features

Supported target platforms

- Windows 32-bit and 64-bit

- macOS 64-bit

- Mac ARM

- iOS 64-bit

- iOS Simulator ARM 64-bit

- Android 32-bit and 64-bit

- Linux 32-bit (only in Lazarus and Free Pascal) and 64-bit

General usability:

- Direct access to server data without using client library. Does not require installation of other data provider layers (such as BDE and ODBC)

- Interface compatible with standard data access methods, such as BDE and ADO

- VCL, LCL and FMX versions of library available

- Separated run-time and GUI specific parts allow you to create pure console applications such as CGI

- Unicode and national charset support

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

- Support for all existing MySQL protocols including the prepared statement (binary) protocol

- Secure connections with SSL, SSH, and HTTP/HTTPS tunneling (using SecureBridge)

- Full support for all current authentication protocols

- Ability to search for installed MySQL servers in a local network

- Connection timeout and command timeout management

Compatibility:

- Full support of the latest versions of MySQL

- Support for Embedded MySQL server

- Support for all MySQL Server data types

- Compatible with all IDE versions starting with Delphi 6, C++Builder 6 and Lazarus (Free Pascal)

- Includes provider for UniDAC Standard Edition

- Wide reporting component support, including support for InfoPower, ReportBuilder

- 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

MySQL Server technology support:

- Fast record insertion with TMyLoader component

- HANDLER syntax support

- Transaction isolation level support

- Possibility to retrieve last auto-incremented value

- Session identifer retrieval

- Server object information retrieval

- Row-level and table-level locking support

Performance:

- High overall performance

- Fast controlled fetch of large data blocks

- Optimized string data storing

- Advanced connection pooling

- High performance applying of cached updates with batches

- Caching of calculated and lookup fields

- 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

- Localmaster/detail relationship

- Master/detail relationship in CachedUpdates mode

Data access and data management automation:

- Automatic data updating with TMyQuery, TMyTable and TMyStoredProc components

- Automaticrecord refreshing

- Automatic query preparing

- Automatic checking for row modifications by another user

- 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 TMyTable component

- BLOB compression support

- Support for using macros in SQL

- FmtBCD fields support

- Ability to customize update commands by attaching external components to
  TMyUpdateSQL objects

- Ability to perform MySQL administration tasks with the TMyServerControl component

- Value range retrieval for ENUM and SET fields

- Retrieval of output parameters from stored procedures and functions

- Automatic retrieval of default field values

- [Deferred detail DataSet refresh](#) in master/detail relationships

- [MIDAS](#) technology support

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

- Extended data management with [TMyDump](#), [TMyBackup](#) components

Script execution:

- Advanced script execution features with [TMyScript](#) component

- Support for executing [individual statements](#) in scripts

- Support for [executing huge scripts stored in files](#) with dynamic loading

- [Optimized multi-statement script execution](#)

- Ability to use standard MySQL client tool syntax in scripts

- Ability to [break long-running query execution](#)

SQL execution monitoring:

- Extended SQL tracing capabilities provided by [TMySQLMonitor](#) component and [DBMonitor](#)

- Borland SQL Monitor support

- Ability to [send messages to DBMonitor](#) from any point in your program

- Ability to [retrieve information about the last query execution](#)

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](#) with [Migration Wizard](#)

- More convenient data source setup with the [TMyDataSource](#) component

- Syntax highlighting in design-time editors

Resources:
- Code documentation and guides in the CHM, PDF, and HXS formats

- Many helpful [demo](#) projects

Licensing and support:
- Included annual [MyDAC Subscription](#) with [Priority Support](#)

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

2.3   Requirements

Requirements for using MyDAC in Direct mode

If you use MyDAC to connect to MySQL in [Direct](#) mode, you do not need to have MySQL client library on your machine or deploy it with your MyDAC-based application.

Requirements for using MyDAC in Client mode

If you use MyDAC to connect to MySQL in Client mode, you need to have access to the MySQL client library. In particular, you will need to make sure that the MySQL client library is installed on the machines your MyDAC-based application is deployed to. MySQL client library is libmysql.dll file for Windows, or libmysqlclient.so (libmysqlclient.so.X) for Linux. Please refer to descriptions of LoadLibrary() and dlopen() functions accordingly for detailed information about MySQL client library file location. You may need to deploy the MySQL client library with your application or require that users have it installed.

Requirements for using Embedded MySQL server

If you are working with Embedded server, you should have access to Embedded MySQL server library (libmysqld.dll). For more information visit <u>Using Embedded server</u>.

Request Support   **DAC Forum**   **Provide Feedback**

2.4 Compatibility

MySQL Compatibility

MyDAC supports the following versions of MySQL:

- MySQL server: 9, 8.4, 8.3, 8.2, 8.1, 8.0, 6.0, 5.7, 5.6, 5.5, 5.1, 5.0, 4.1, 4.0, 3.23.

- MySQL embedded server: 9, 8.4, 8.3, 8.2, 8.1, 8.0, 6.0, 5.7, 5.6, 5.5, 5.1, 4.1, 4.0.

MariaDB Compatibility

MariaDB server: 5.x-12.x.

Microsoft Azure Database for MySQL Compatibility

MyDAC supports Microsoft Azure Database for MySQL.

Amazon RDS for MySQL Compatibility

MyDAC supports Amazon RDS for MySQL and Amazon Aurora.

Google Cloud for MySQL Compatibility

MyDAC supports Google Cloud for MySQL.

IDE Compatibility

MyDAC is compatible with the following IDEs:

Embarcadero RAD Studio 13 Florence
- Embarcadero Delphi 13 Florence for Windows

- Embarcadero Delphi 13 Florence for macOS

- Embarcadero Delphi 13 Florence for Linux

- Embarcadero Delphi 13 Florence for iOS

- Embarcadero Delphi 13 Florence for Android

- Embarcadero C++Builder 13 Florence for Windows

- Embarcadero C++Builder 13 Florence for iOS

- Embarcadero C++Builder 13 Florence for Android

Embarcadero RAD Studio 12 Athens (Requires Release 1, Release 2, or Release 3)
- Embarcadero Delphi 12 Athens for Windows

- Embarcadero Delphi 12 Athens for macOS

- Embarcadero Delphi 12 Athens for Linux

- Embarcadero Delphi 12 Athens for iOS

- Embarcadero Delphi 12 Athens for Android

- Embarcadero C++Builder 12 Athens for Windows

- Embarcadero C++Builder 12 Athens for iOS

- Embarcadero C++Builder 12 Athens for Android

Embarcadero RAD Studio 11.1 Alexandria
- Embarcadero Delphi 11.1 Alexandria for Windows

- Embarcadero Delphi 11.1 Alexandria for macOS

- Embarcadero Delphi 11.1 Alexandria for Linux

- Embarcadero Delphi 11.1 Alexandria for iOS

- Embarcadero Delphi 11.1 Alexandria for Android

- Embarcadero C++Builder 11.1 Alexandria for Windows

- Embarcadero C++Builder 11.1 Alexandria for iOS

- Embarcadero C++Builder 11.1 Alexandria for Android

Embarcadero RAD Studio 10.4 Sydney (Requires Release 1 or Release 2)
- Embarcadero Delphi 10.4 Sydney for Windows

- Embarcadero Delphi 10.4 Sydney for macOS

- Embarcadero Delphi 10.4 Sydney for Linux

- Embarcadero Delphi 10.4 Sydney for iOS

- Embarcadero Delphi 10.4 Sydney for Android

- Embarcadero C++Builder 10.4 Sydney for Windows

- Embarcadero C++Builder 10.4 Sydney for iOS

- Embarcadero C++Builder 10.4 Sydney for Android

Embarcadero RAD Studio 10.3 Rio (Requires Release 2 or Release 3)
- Embarcadero Delphi 10.3 Rio for Windows

- Embarcadero Delphi 10.3 Rio for macOS

- Embarcadero Delphi 10.3 Rio for Linux

- Embarcadero Delphi 10.3 Rio for iOS

- Embarcadero Delphi 10.3 Rio for Android

- Embarcadero C++Builder 10.3 Rio for Windows

- Embarcadero C++Builder 10.3 Rio for macOS

- Embarcadero C++Builder 10.3 Rio for iOS

- Embarcadero C++Builder 10.3 Rio for Android

Embarcadero RAD Studio 10.2 Tokyo (Incompatible with Release 1)
- Embarcadero Delphi 10.2 Tokyo for Windows

- Embarcadero Delphi 10.2 Tokyo for macOS

- Embarcadero Delphi 10.2 Tokyo for Linux

- Embarcadero Delphi 10.2 Tokyo for iOS

- Embarcadero Delphi 10.2 Tokyo for Android

- Embarcadero C++Builder 10.2 Tokyo for Windows

- Embarcadero C++Builder 10.2 Tokyo for macOS

- Embarcadero C++Builder 10.2 Tokyo for iOS

- Embarcadero C++Builder 10.2 Tokyo for Android

Embarcadero RAD Studio 10.1 Berlin
- Embarcadero Delphi 10.1 Berlin for Windows

- Embarcadero Delphi 10.1 Berlin for macOS

- Embarcadero Delphi 10.1 Berlin for iOS

- Embarcadero Delphi 10.1 Berlin for Android

- Embarcadero C++Builder 10.1 Berlin for Windows

- Embarcadero C++Builder 10.1 Berlin for macOS

- Embarcadero C++Builder 10.1 Berlin for iOS

- Embarcadero C++Builder 10.1 Berlin for Android

Embarcadero RAD Studio 10 Seattle
- Embarcadero Delphi 10 Seattle for Windows

- Embarcadero Delphi 10 Seattle for macOS

- Embarcadero Delphi 10 Seattle for iOS

- Embarcadero Delphi 10 Seattle for Android

- Embarcadero C++Builder 10 Seattle for Windows

- Embarcadero C++Builder 10 Seattle for macOS

- Embarcadero C++Builder 10 Seattle for iOS

- Embarcadero C++Builder 10 Seattle for Android

Embarcadero RAD Studio XE8
- Embarcadero Delphi XE8 for Windows

- Embarcadero Delphi XE8 for macOS

- Embarcadero Delphi XE8 for iOS

- Embarcadero Delphi XE8 for Android

- Embarcadero C++Builder XE8 for Windows

- Embarcadero C++Builder XE8 for macOS

- Embarcadero C++Builder XE8 for iOS

- Embarcadero C++Builder XE8 for Android

Embarcadero RAD Studio XE7
- Embarcadero Delphi XE7 for Windows

- Embarcadero Delphi XE7 for macOS

- Embarcadero Delphi XE7 for iOS

- Embarcadero Delphi XE7 for Android

- Embarcadero C++Builder XE7 for Windows

- Embarcadero C++Builder XE7 for macOS

- Embarcadero C++Builder XE7 for iOS

- Embarcadero C++Builder XE7 for Android

Embarcadero RAD Studio XE6
- Embarcadero Delphi XE6 for Windows

- Embarcadero Delphi XE6 for macOS

- Embarcadero Delphi XE6 for iOS

- Embarcadero Delphi XE6 for Android

- Embarcadero C++Builder XE6 for Windows

- Embarcadero C++Builder XE6 for macOS

- Embarcadero C++Builder XE6 for iOS

- Embarcadero C++Builder XE6 for Android

Embarcadero RAD Studio XE5 (Requires Update 2)
- Embarcadero Delphi XE5 for Windows

- Embarcadero Delphi XE5 for macOS

- Embarcadero Delphi XE5 for iOS

- Embarcadero Delphi XE5 for Android

- Embarcadero C++Builder XE5 for Windows

- Embarcadero C++Builder XE5 for macOS

- Embarcadero C++Builder XE5 for iOS

Embarcadero RAD Studio XE4
- Embarcadero Delphi XE4 for Windows

- Embarcadero Delphi XE4 for macOS

- Embarcadero Delphi XE4 for iOS

- Embarcadero C++Builder XE4 for Windows

- Embarcadero C++Builder XE4 for macOS

Embarcadero RAD Studio XE3 (Requires Update 2)

- Embarcadero Delphi XE3 for Windows

- Embarcadero Delphi XE3 for macOS

- Embarcadero C++Builder XE3 for Windows

- Embarcadero C++Builder XE3 for macOS

Embarcadero RAD Studio XE2 (Requires Update 4 Hotfix 1)
- Embarcadero Delphi XE2 for Windows

- Embarcadero Delphi XE2 for macOS

- Embarcadero C++Builder XE2 for Windows

- Embarcadero C++Builder XE2 for macOS

Embarcadero RAD Studio XE
- Embarcadero Delphi XE

- Embarcadero C++Builder XE

Embarcadero RAD Studio 2010
- Embarcadero Delphi 2010

- Embarcadero C++Builder 2010

CodeGear RAD Studio 2009 (Requires Update 3)
- CodeGear Delphi 2009

- CodeGear C++Builder 2009

CodeGear RAD Studio 2007
- CodeGear Delphi 2007

- CodeGear C++Builder 2007

Borland Developer Studio 2006
- Borland Delphi 2006

- Borland C++Builder 2006

Borland Delphi 7
Borland Delphi 6 (Requires Update Pack 2 – Delphi 6 Build 6.240)
Borland C++Builder 6 (Requires Update Pack 4 – C++Builder 6 Build 10.166)
Lazarus 4.2 and Free Pascal 3.2.2 for Windows, macOS, and Linux.

All the existing Delphi and C++Builder editions are supported: Architect, Enterprise,

Professional, Community, and Starter.

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

Supported Target Platforms

- Windows 32-bit and 64-bit

- macOS 64-bit and ARM (Apple Silicon M1)

- Linux 32-bit (only in Lazarus and Free Pascal) and 64-bit

- iOS 64-bit

- iOS Simulator ARM 64-bit

- Android 32-bit and 64-bit

Support for Windows 64-bit is available since RAD Studio XE2. Support for iOS 64-bit is available since RAD Studio XE8. Support for Android 32-bit is available since RAD Studio XE5. Support for Linux 64-bit is available since RAD Studio 10.2 Tokyo. Support for macOS 64-bit is available since RAD Studio 10.3 Rio. Support for Android 64-bit is available since RAD Studio 10.3.3 Rio.

Supported GUI Frameworks

- FireMonkey (FMX)

- Visual Component Library (VCL)

- Lazarus Component Library (LCL)

Devart Data Access Components Compatibility

All DAC products are compatible with each other.

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

     [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

2.5 Using Several DAC Products in One IDE

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

Packages:

- dacXX.bpl

- dacvclXX.bpl

- dcldacXX.bpl

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

Request Support   DAC Forum   Provide Feedback

2.6 Component List

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

Basic MyDAC components

	TMyConnection	Lets you set up and control connections to MySQL database server.
	TMyQuery	Uses SQL statements to retrieve data from MySQL table or tables and supply it to one or more data-aware components through a TDataSource component. Provides flexible data update functionality.

	[TMyCommand](#)	Executes SQL statements and stored procedures, which do not return rowsets.
	[TMyTable](#)	Lets you retrieve and update data in a single table without writing SQL statements.
	[TMyStoredProc](#)	Executes stored procedures and functions.
	[TMyUpdateSQL](#)	Lets you tune update operations for a DataSet component.
	[TMyDataSource](#)	Provides an interface between MyDAC dataset components and data-aware controls on a form.
	[TMyScript](#)	Executes sequences of SQL statements.
	[TMySQLMonitor](#)	Interface for monitoring dynamic SQL execution in MyDAC-based applications.
	[TMyConnectDialog](#)	Used to build custom prompts for username, password and server name.
	[TVirtualTable](#)	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.
	[TVirtualDataSet](#)	Dataset that processes arbitrary non-tabular data.

MyDAC Professional Edition components

	[TMyEncryptor](#)	Represents data encryption and decryption in client application.
	[TMyLoader](#)	Provides quick loading data to MySQL database.
	[TMyDump](#)	Serves to store a database or its parts as a script and also to restore database from received script.
	[TMyBackup](#)	Serves for backup copying specified tables on the server.
	[TMyServerControl](#)	Serves to control the server and execution of standard service tasks.

	TMyEmbConnection	Is used to establish connection to Embedded MySQL server.
	TMyMetaData	Retrieves metadata on specified SQL object.
	TCRBatchMove	Transfers data between all types of TDataSets descendants. This component is placed on the Data Access page of the Component palette.

See Also

- Hierarchy chart

Request Support   DAC Forum   Provide Feedback

2.7   Hierarchy Chart

Many MyDAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for MyDAC is shown below. The MyDAC 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

      |         |—TCustomMyConnection

      |              |—TMyConnection

      |              |—TMyEmbConnection

      |—TDataSet

      |    |—TMemDataSet

      |         |—TCustomDADataSet

      |         |    |—TCustomMyDataSet

      |         |         |—TMyQuery
```

```
|            |              |─TCustomMyTable
|            |       |   |─TMyTable
|            |       |─TCustomMyStoredProc
|            |       |   |─TMyStoredProc
|            |       |─TMyServerControl
|       |─TDAMetaData
|       |   |─TMyMetaData
|       |─TVirtualTable
|─TDataSource
|   |─TCRDataSource
|       |─TMyDataSource
|─DADataAdapter
|   |─MyDataAdapter
|─TCRBatchMove
|─TCustomConnectDialog
|   |─TMyConnectDialog
|─TCustomDASQL
|   |─TMyCommand
|─TCustomDASQLMonitor
|   |─TMySQLMonitor
|─TDADump
|   |─TMyDump
|─TDALoader
|   |─TMyLoader
|─TDAScript
|   |─TMyScript
|─TMyBackup
|─TMyIOHandler
|─TCREncryptor
    |─TMyEncryptor
```

Request Support   DAC Forum   Provide Feedback

2.8 Editions

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

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

The **Professional** edition shows off the full power of MyDAC, enhancing MyDAC Standard Edition with support for MySQL-specific functionality and advanced dataset management features.

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

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

MyDAC Edition Matrix

Feature	Standard	Professional
Direct connectivity		
Connection without MySQL client library	✓	✓
Desktop Application Development		
Windows	✓	✓
macOS	✕	✓
Linux	✕	✓
Mobile Application Development		
iOS	✕	✓
Android	✕	✓

Data Access Components		
Base Components: TMyConnection TMyQuery TMyCommand TMyTable TMyStoredProc TMyDataSource	✔	✔
Script executing TMyScript	✔	✔
Fast data loading into the server TMyLoader	✘	✔
MySQL Specific Components		
Obtaining metadata about database objects TMyMetaData	✔	✔
Connection with Embedded MySQL Server TMyEmbConnection	✘	✔
MySQL server control TMyServerControl	✘	✔
SQL Builder managing for MySQL Add-in TMyBuilder	✘	✔
Storing a database as a script TMyDump	✘	✔
Backup Service TMyBackup	✘	✔
DataBase Activity Monitoring		
Monitoring of per-component SQL execution TMySQLMonitor	✔	✔
Additional components		
Advanced connection dialog TMyConnectDialog	✔	✔
Data encryption and decryption TMyEncryptor	✘	✔
Data storing in memory table TVirtualTable	✔	✔
Dataset that wraps arbitrary non-tabular data TVirtualDataset	✔	✔
Advanced DBGrid with extended functionality TCRDBGrid	✔	✔

Records transferring between datasets TCRBatchMove	✕	✔	
Design-Time Features			
Enhanced component and property editors	✔	✔	
Migration Wizard	✔	✔	
DataSet Manager	✕	✔	
Cross IDE Support			
Lazarus and Free Pascal Support	✕	SRC[1]	

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

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

2.9   Licensing

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

INTRODUCTION

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

LICENSE

1. GRANT OF LICENSE

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

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

- install and use the Software on one or more computers, provided it is used by 1 (one) for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Single Developer License"); or

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

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

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

- make one copy of the Software for archival purposes only, or copy the Software onto the hard disk of your computer and retain the original for archival purposes;

- develop and test applications with the Software, subject to the Limitations below;

- create libraries, components, and frameworks derived from the Software for personal use only;

- deploy and register run-time libraries and packages of the Software, subject to the Redistribution policy defined below.

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

No other rights or privileges are granted in this Agreement.

2. LIMITATIONS

Only legally registered users are licensed to use the Software, subject to all of the conditions

of this Agreement. Usage of the Software is subject to the following restrictions.

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

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

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

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

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

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

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

3. REDISTRIBUTION

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

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

4. TRANSFER

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

5. TERMINATION

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

6. EVALUATION

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

7. WARRANTY

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

8. SUBSCRIPTION AND SUPPORT

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

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

9. COPYRIGHT

The Software is confidential and proprietary copyrighted work of Devart and is protected by

international copyright laws and treaty provisions. You may not remove the copyright notice from any copy of the Software or any copy of the written materials, accompanying the Software.

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

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

2.10   Getting Support

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

Support Options

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

- You can find out more about MyDAC installation or licensing by consulting the Licensing and FAQ sections.

- You can get community assistance and technical support on the MyDAC Community Forum.

- You can get advanced technical assistance by MyDAC developers through the **MyDAC Priority Support** program.

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

MyDAC Priority Support

MyDAC Priority Support is an advanced product support service for getting expedited individual assistance with MyDAC-related questions from the MyDAC 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 MyDAC Subscription.

To get help through the MyDAC Priority Support program, please send an email to support@devart.com describing the problem you are having. Make sure to include the following information in your message:

- The version of Delphi, C++Builder you are using.

- Your MyDAC Registration number.

- Full MyDAC edition name and version number. You can find both of these in the About sheet of TMyConnection Editor or from the MySQL | About menu.

- Versions of the MySQL server and client you are using.

- A detailed problem description.

- If possible, a small test project that reproduces the problem. Please include definitions for all database objects and avoid using third-party components.

Request Support   DAC Forum   Provide Feedback

2.11   Frequently Asked Questions

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

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

Installation and Deployment
1. I am having a problem installing MyDAC or compiling MyDAC-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 MyDAC 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 MyDAC, SDAC, ODAC or IBDAC versions. All these products use common base packages. The easiest way to avoid the problem is to uninstall all installed DAC products and then download from our site and install the last builds.

2. What software should be installed on a client computer for MyDAC-based applications to work?

Usually, you do not need any additional files. The only exceptions to this rule are listed below:

- If you are using MySQL Embedded server (if you are using TMyConnection with TMyConnection.Options.Embedded = True or TMyEmbConnection), you need the server itself (*libmysqld.dll*) and the service files for it, for example *errmsg.sys*.

- If you are connecting in Client mode, (TMyConnection.Options.Direct = False), you need *libmysql.dll*.

- If you are using SSL (TMyConnection.Options.Protocol = mpSSL), you need the OpenSSL library files - *ssleay32.dll* and *libeay32.dll*.

3. When I try to connect to the server, I get an error "MySQL client library couldn't be loaded."

You are using TMyConnection.Options.Direct := False mode and the client library is not available for your application.

Windows: You should copy client file libmysql.dll to a folder available to the executable unit of your program. For example, to the folder containing the executable or to the Windows system folder. For more details, see the description of LoadLibrary and the PATH environment variable.

Linux: You should copy the client file libmysqlclient.so.X to the folder available to the executable unit of your program. For more details, see the description of the dlopen function and the LD_LIBRARY_PATH environment variable.

Licensing and Subscriptions
1. Am I entitled to distribute applications written with MyDAC?

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

2. Can I create components using MyDAC?

You can create your own components that are inherited from MyDAC or that use the MyDAC

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 MyDAC 5.00?

The basic MyDAC license agreement will remain the same. With MyDAC 5.00, the MyDAC Edition Matrix will be reorganized and a new MyDAC Subscription Program will be introduced.

4. What do the MyDAC 5.00 Edition Levels correspond to?

MyDAC 5.00 will come in six editions: Trial, Standard, Professional, Professional with Sources, Developer, and Developer with Sources.

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

Edition Correspondence Table for Upgrading to MyDAC 5.00

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

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

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

After MyDAC 5.00, all upgrades to future versions are free to users with an active MyDAC Subscription.

Users that have a registration for versions of MyDAC prior to MyDAC 5.00 will have to first upgrade to MyDAC 5.00 to jump in on the Subscription program.

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

The **MyDAC Subscription Program** is an annual maintenance and support service for MyDAC users.

Users with a valid MyDAC Subscription get the following benefits:

- Access to new versions of MyDAC when they are released

- Access to all MyDAC updates and bug fixes

- Product support through the MyDAC Priority Support program

- Notification of new product versions

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

The MyDAC Subscription Program is available for registered users of MyDAC 5.00 and higher.

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

Yes, you can. MyDAC version licenses are perpetual.

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

An annual MyDAC Subscription is included when ordering or upgrading to any registered (non-Trial) edition of MyDAC 5.00 or higher.

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

9. Does this mean that if I upgrade to MyDAC 5 from MyDAC 4, I'll get an annual MyDAC Subscription for free?

Yes.

10.How do I upgrade to MyDAC 5.00?

To upgrade to MyDAC 5.00, you can get a Version Update from the MyDAC Ordering Page. For more information, please contact sales@devart.com.

Performance
1. How productive is MyDAC?

MyDAC uses a low-level protocol to access the database server. This allows MyDAC to achieve high performance. From time to time we compare MyDAC with other products, and MyDAC 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, MyDAC 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 MyDAC component editors at design time?

Download and install dbForge Fusion for MySQL. In addition to syntax highlighting, MySQL Developer Tools provides a lot of additional features.
Alternatively, you can download and install the freeware SynEdit component set.

2. How can I quickly convert a project from BDE to MyDAC?

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

3. How can I determine which version of MyDAC I am using?

You can determine your MyDAC version number in several ways:
- During installation of MyDAC, consult the MyDAC Installer screen.

o After installation, see the *history.html* file in your MyDAC installation directiory.

o At design-time, select MySQL | About MyDAC from the main menu of your IDE.

o At run-time, check the value of the MydacVersion and DACVersion constants.

4. 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.

5. How can I execute a query saved in the SQLInsert, SQLUpdate, SQLDelete, or SQLRefresh properties of a MyDAC 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 MyDAC during the execution of the Post, Delete, or RefreshRecord methods, and are used to construct the query to the server. Their values can contain parameters with names of fields in the underlying data source, which will be later replaced by appropriate data values.

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

o Fill the SQLInsert property with the parameterized query template you want to use.

o Call Insert.

o Initialize field values of the row to insert.

o Call Post.

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

6. How can I get a list of the databases on the server?

Use the TMyConnection.GetDatabaseNames method.

7. How can I get a list of the tables list in a database?

Use the TMyConnection.GetTableNames method.

8. Some questions about the visual part of MyDAC

The following situations usually arise from the same problem:

- o I set the Debug property to True but nothing happens!

- o While executing a query, the screen cursor does not change to an hour-glass.

- o Even if I have LoginPromp set to True, the connect dialog does not appear.

To fix this, you should add the MyDacVcl (for Windows) or MyDacClx (for Linux) unit to the uses clause of your project.

General Questions

1. I would like to develop an application that works with MySQL Server. Should I use MyDAC or DbxMda?

DbxMda is our dbExpress driver for MySQL. 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.

MyDAC is a specialized set of components for MySQL, which has advanced server-specific design-time and a component interface similar to that of BDE.

We tried to include maximal support of MySQL-specific features in both DbxMda and MyDAC. However, the nature of dbExpress technology has some insurmountable restrictions. For example, Unicode fields cannot be passed from a driver to dbExpress.

MyDAC and DbxMda 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 MySQL, it is probably better to use DbxMda. In other cases, especially when migrating from BDE or ADO, you should use MyDAC.

2. Are the MyDAC connection components thread-safe?

Yes, MyDAC is thread-safe but there is a restriction. The same TMyConnection object cannot be used in several threads. So if you have a multithreaded application, you should have a TMyConnection object for each thread that uses MyDAC.

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

We always try to keep MyDAC compatible with previous versions, but sometimes we have to change behaviour of MyDAC 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 MyDAC help. This topic describes such changes, and how to revert to the old MyDAC behaviour.

4. 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 TCustomMyDataSet.Options.StrictUpdate topic in MyDAC help for more information.

5. I have problems using BIGINT and INT UNSIGNED fields as key fields in master/ detail relationships, and accessing values of such fields through the Field.Value property.

Fields of this type are represented in Delphi by TLargeIntField objects. In some versions of Delphi, you cannot access these fields through the Value property (for more information see the SetVarValue protected method of TLargeintField in the DB unit). To avoid this problem, you can change the field type to INT, which is usually sufficient for key fields. Alternatively, you can avoid using Value.

For master/detail relationships the problem can be avoided only by changing type of the key field to INT, as Delphi's master/detail mechanism works through Field.Value.

6. On accessing server I get a 'MySQL server has gone away' or 'Lost connection to MySQL server during query' error.

First of all, you should find out what causes the problem. The list of most frequent reasons for this error to occur is below.

- o Client side: The value of TMyConnection.ConnectionTimeout or TCustomMyDataSet.CommandTimeout is too small. To check this hypothesis, try setting TCustomMyDataSet.CommandTimeout to 0 (infinitive) and TMyConnection.ConnectionTimeout to 300.

- o Server side: MySQL server has closed the connection. You can read a detailed description of all possible reasons for this to happen in the MySQL Reference Manual. Almost always it is because the value of wait_timeout variable is too small. Try increasing it. If this solution is not possible (for example, because you don't have enough rights), you should invoke MyConnection.Ping with an interval less than wait_timeout. Use TTimer in TMyConnection thread to accomplish this task.

- o Unstable connection (GPRS etc). In case of unstable connection you can adapt MyDAC to work in such conditions by changing some of its settings. For more information please see the "Working in Unstable Networks" article in the MyDAC help documentation.

If the connection is lost, MyDAC tries to reconnect to server. However, your last command will probably not be executed, and you should repeat it again. MyDAC does not try to reconnect if a transaction has started or if at least one of statements is prepared.

7. Some problems using TCustomDADataSet.FetchAll=False mode

The following problems may appear when using FetchAll=False mode:
- o I have problems working with temporary tables.

- o I have problems working with transactions.

- o Sometimes my application hangs on applying changes to the database.

Usage of FetchAll=False mode has many advantages; however, it also has some restrictions

since it requires an additional connection to server for data fetching to be created. The additional connection is created to prevent the main connection from blocking.

These problems can be avoided by setting the FetchAll property. Please see description of the FetchAll propery and the CreateConnection option in MyDAC help for more information.

Another alternative that prevents the application from hanging is to switch to the InnoDB storage engine from MyISAM (FetchAll stays False). An application may hang because MyISAM tables can get locked in a read/write collision. If you try to update a table that is not fetched out, MySQL blocks the thread and waits untill the table is completely fetched. For details please refer to the MySQL Reference Manual, the Locking Issues section.

8. I get an error when opening a Stored Procedure that returns a result set.

Probably this is a bug of the MySQL Server protocol with prepared stored procedures that return record sets. It occurs in the following cases:

o After a call to the Prepare method of MyStoredProc, if the latter had already prepared and opened. The following piece of code demonstrates the problem:

```
MyStoredProc.Prepare;
MyStoredProc.Open;
MyStoredProc.UnPrepare;
MyStoredProc.Prepare;
```

o After a call to the MyStoredProc.Execute method, if the stored procedure returns more than one record set.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

3   Getting Started

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

- Installation

- [Migration Wizard](#)

- [Migration from BDE](#)

- [Connecting to MySQL](#)

- [Connecting to MySQL Embedded](#)

- [Creating Database Objects](#)

- [Deleting Data From Tables](#)

- [Inserting Data Into Tables](#)

- [Retrieving Data](#)

- [Modifying Data](#)

- [Stored Procedures](#)

- [Working With Result Sets Using Stored Procedures](#)

Request Support   DAC Forum   Provide Feedback

3.1   Installation

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

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

Before installing MyDAC ...

Two versions of MyDAC 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 MyDAC may be incompatible with older versions of ODAC, IBDAC, and SDAC.

So before installing a new version of MyDAC, uninstall any previous version of MyDAC 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 support@devart.com

> **Note:** You can avoid performing MyDAC uninstallation manually when upgrading to a new version by directing the MyDAC 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 `mydacXX.exe /force,` specifying the full path to the appropriate version of the installation program).

Installed packages

> **Note:** `%MyDAC%` denotes the path to your MyDAC 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
mydacXX.bpl	MyDAC run-time package	Windows\System32
dclmydacXX.bpl	MyDAC design-time package	Delphi\Bin
dclmysqlmonXX.bpl	TMySQLMonitor component	Delphi\Bin
mydacvclXX.bpl*	VCL support package	Delphi\Bin
crcontrolsXX.bpl	TCRDBGrid component	Delphi\Bin

* Not included in Delphi 5. In this IDE this functionality is distributed among the other packages.

Additional packages for using MyDAC managers and wizards

Name	Description	Location
datasetmanagerXX.bpl	DataSet Manager package	Delphi\Bin
mymigwizardXX.dll	MyDAC BDE Migration wizard	%MyDAC%\Bin

Environment Changes

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

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

Delphi

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

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

C++Builder

C++Builder 6:

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

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

C++Builder 2006, 2007:

- $(BCB)\MyDAC\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)\MyDAC\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 MyDAC Installer performs C++Builder environment changes automatically for compiled versions of MyDAC.

Lazarus

The MyDAC installation program only copies MyDAC files. You need to install MyDAC packages to Lazarus IDE manually. Open %MyDAC%\Source\Lazarus1\dclmydac10.lpk (for Trial version %MyDAC%\Packages\dclmydac10.lpk) file in Lazarus and press the Install

button. After that Lazarus IDE will be rebuilded with MyDAC packages.

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

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

Installation of Additional Components and Add-ins

dbForge Fusion for MySQL

dbForge Fusion for MySQL is a powerful database development and administration tool for MySQL. dbForge Fusion for MySQL is available as an add-in for Delphi and C++Builder 2009, CodeGear RAD Studio 2007, or as a standalone application. For more information, visit the dbForge Fusion for MySQL page online.

DBMonitor

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

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

3.2   Migration Wizard

Note: Migration Wizard is only available for Delphi.

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

To convert a project, perform the following steps.

• Select **BDE Migration Wizard** from the **MyDAC** menu

• Select **Replace BDE components** to replace corresponding components with MyDAC and press the Next button.

- Select the location of the files to search - current open project or disc folder.

- If you have selected Disc folder on the previous step, specify the required folder and specify whether to process subfolders. Press the Next button.

- Select whether to make backup (it is highly recommended to make a backup), backup location, and log parameters, and press the Next button. Default backup location is RBackup folder in your project folder.

- Check your settings and press the Finish button to start the conversion operation.

- The project should be saved before conversion. You will be asked before saving it. Click Yes to continue project conversion.

After the project conversion it will be reopened.

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

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

- Select **BDE Migration Wizard** from the **MyDAC** 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](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

3.3   Migration from BDE

In MyDAC the interests of BDE application developers were taken into consideration. So starting to use MyDAC after working with BDE would be easy even for developing complex projects. Moreover, MyDAC does not have problems like ones with LiveQuery and compatibility of applications developed using different versions in BDE.

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 MySQL server. Also access to MySQL Embedded server is supported.

MyDAC 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 MyDAC. BDE-components that will be replaced:

• TDatabase -> TMyConnection

• TQuery -> TMyQuery

• TTable -> TMyTable

• TUpdateSQL -> TMyUpdateSQL

To run the Wizard, select BDE Migration Wizard item in MyDAC menu and follow the instructions. BDE Migration Wizard does not support C++Builder.

Note: The Wizard serves only to simplify routine operations and after the conversion project might be uncompiled.

Below is the list of properties and methods which cannot be converted automatically. Here you can find hints for users to simplify manual replacement.

TDatabase
• AliasName - specific BDE property. Not supported by MyDAC

• DatabaseName - has a different meaning in BDE and MyDAC. At MyDAC it means MySQL Server database. See TMyConnection.Database for details

• Locale - see TMyConnection.Options.CharSet

• KeepConnection - not supported by MyDAC

• Params - see TMyConnection properties

- Session, SessionAlias, SessionName - MyDAC does not need global management of a group of database connections in an application. So these properties are not supported

- Temporary - has no sense in MyDAC. Additional connections are created but not available for the user. See TCustomMyDataSet.FetchAll = False for details

- TraceFlags - see TCustomDASQLMonitor.TraceFlags

- TransIsolation - see IsolationLevel

- Execute - use ExecSQL instead of this method

- FlushSchemaCache - not supported by MyDAC

- GetFieldNames - not supported by MyDAC

- IsSQLBased - not supported by MyDAC. For MySQL must be always True

- ApplyUpdates - parameters are not supported. To update only specified DataSets use TMemDataset.ApplyUpdates. Update is performed within a transaction.

TBDEDataSet

- BlockReadSize - see TCustomDADataSet.FetchRows

- CacheBlobs - MySQL Server does not provide service of suspended BLOB loading

- KeySize - specific BDE property. Not supported by MyDAC.

TDBDataSet

- AutoRefresh - supported through TCustomDADataSet.RefreshOptions

- DBFlags, DBHandle, DBLocate, DBSession, Handle - BDE-specific property. Not supported by MyDAC

- SessionName - not supported by MyDAC

- UpdateMode - not supported by MyDAC. By default, the behaviour corresponds upWhereKeyOnly. To change this behaviour see TCustomDADataSet.SQLUpdate, TCustomDADataSet.SQLDelete, TCustomDADataSet.SQLRefresh, and TCustomMyDataSet.Options.CheckRowVersion.

TQuery

- Constrained - specific BDE property. Not supported by MyDAC

- DataSource - see TCustomDADataSet.MasterSource

- Local - specific BDE property. Not supported by MyDAC

- RequestLive - almost all query result sets are updatable. See TMyQuery.UpdatingTable,
  TCustomDADataSet.ReadOnly, CanModify, TCustomDADataSet.SQLInsert,
  TCustomDADataSet.SQLUpdate, TCustomDADataSet.SQLDelete.

- Text - specific BDE property. Not supported by MyDAC.

TTable

- DefaultIndex - not used in MyDAC. If you need to sort a table by any field see
  P:Devart.MyDac.TCustomMyTable.OrderFields, TMemDataSet.IndexFieldNames

- Exists, CreateTable, AddIndex, DeleteIndex, StoreDefs, Deletetable, TableType - MyDAC
  does not allow to create tables by using TTable. If you need to create a table execute
  'CREATE TABLE ...' query or use any special third-party tools.

- IndexDefs   - not used in MyDAC, but fills on first call

- IndexFieldNames   - a list of fields for local sorting. See TMemDataSet.IndexFieldNames

- IndexFieldCount, IndexFields, IndexFiles, IndexName, GetIndexNames, GetIndexInfo - Not
  supported by MyDAC

- KeyExclusive - not supported by MyDAC. Use SELECT ... FROM .. WHERE ... to get
  requested result

- KeyFieldCount - not supported by MyDAC, as key fields are not used for searching on the
  client side

- TableLevel - specific BDE property. Not supported by MyDAC

- ApplyRange, CancelRange, EditRangeStart, EditRangeEnd, SetRange - MyDAC does not
  support Range

- BatchMove - has no meaning in MySQL. Use INSERT ... INTO ... SELECT syntax to copy
  records onto 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 MySQL

- FlushBuffers - see TMyServerControl.Flush

- Translate - use AnsiToNative and similar functions.

TSession

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

TUpdateSQL

A complete analogue to TMyUpdateSQL.

Request Support   DAC Forum   Provide Feedback

3.4   Connecting to MySQL Server

This tutorial describes how to connect to MySQL using the TMyConnection component.

1. Requirements
2. General Information
3. Creating a Connection
    - 3.1 Connecting at Design-Time
        - 3.1.1 Using TMyConnection Editor
        - 3.1.2 Using Object Inspector
    - 3.2 Connecting at Runtime
4. Opening a Connection
5. Modifying a Connection
6. Closing a Connection

Requirements

This tutorial assumes that you have installed MyDAC and run the database server and the IDE. You need to know the server address, the port number (if you use a port other than the default port `3306`), the database name, and the username and password. To connect at runtime, add the MyAccess unit to the `uses` clause for Delphi or include the `MyAccess.hpp` header file for C++ Builder.

General Information

To establish a connection to the server, set up the properties of the TMyConnection component: Server, Port, Database, Username, and Password. You can also specify all connection parameters in the ConnectString property.

You can connect to MySQL in the `Direct` mode or using the MySQL client library. The connection mode is defined by the Direct property, which defaults to `True`. To access MySQL using the client library, set the property to `False`.

MyDAC can also work with the embedded MySQL server. For information about connecting to the embedded server, see Connecting to MySQL Embedded.

Creating a Connection

Connecting at Design-Time

The following assumes that you have already created or opened an existing form in the IDE. At design-time, you can set up a `TMyConnection` object in the TMyConnection Editor or Object Inspector.

1. Find the `TMyConnection` component in the `MyDAC` category on the Tool Palette.

2. Double-click the component. A new object will appear on the form. If this is the first time that you create a `TMyConnection` object in this project, the object will be named `MyConnection1`.

Using TMyConnection Editor

1. Double-click the `MyConnection1` object.

2. Specify the DNS name or IP address of the MySQL server in the `Server` edit box.

3. If you use a port other than the default port `3306`, specify it in the `Port` edit box.

4. Specify the database name in the `Database` edit box.

5. Specify the username (`root` by default) in the `Username` edit box.

6. Specify the password (empty value by default) in the `Password` edit box.

Using Object Inspector

1. Select the `MyConnection1` object on the form.

2. Set the `Database` property to the database name.

3. Set the `Password` property to the password (`root` by default).

4. If you use a port other than the default port `3306`, set the `Port` property to your port.

5. Set the `Server` property to the DNS name or IP address of the MySQL server.

6. Set the `Username` property to the username (empty value by default).

Connecting at Runtime

The same connection parameters at runtime are set up as follows:

Delphi

```
var
  MyConnection1: TMyConnection;
begin
  MyConnection1 := TMyConnection.Create(nil);
  try
    // adds connection parameters
    MyConnection1.Server := 'server';
    MyConnection1.Database := 'database';
    MyConnection1.Username := 'username';
    MyConnection1.Password := 'password';
    MyConnection1.Port := 3306;
    // disables a login prompt
    MyConnection1.LoginPrompt := False;
    // opens a connection
    MyConnection1.Open;
  finally
    MyConnection1.Free;
  end;
end;
```

C++ Builder

```
 TMyConnection* MyConnection1 = new TMyConnection(NULL);
try {
```

```
  // adds connection parameters
 MyConnection1->Server = "server";
    MyConnection1->Database = "database";
 MyConnection1->Username = "username";
 MyConnection1->Password = "password";
 MyConnection1->Port = 3306;
 // disables a login prompt
 MyConnection1->LoginPrompt = False;
 // opens a connection
 MyConnection1->Open();
}
__finally {
 MyConnection1->Free();
}
```

Opening a Connection

To open a connection at run-time, call the `Open` method:

Delphi

```
MyConnection1.Open;
```

C++Builder

```
MyConnection1->Open;
```

Another way to open a connection at runtime is to set the [Connected](#) property to `True`:

Delphi

```
MyConnection1.Connected := True;
```

C++ Builder

```
MyConnection1->Connected = True;
```

You can also set up the `Connected` property at design-time in the Object Inspector.

Modifying a Connection

You can modify a connection by changing properties of the `TMyConnection` object. Note that while some of the object's properties can be altered without changing the state of a connection, in most cases, a connection is closed when a new value is assigned to the property. For example, if you change the value of the `Server` property, a connection is closed immediately and you need to reopen it manually.

Closing a Connection

To close a connection, call the `Close` method or set the `Connected` property to `False`:

Delphi

```
MyConnection1.Close;
```

or:

```
MyConnection1.Connected := False;
```

C++ Builder

```
MyConnection1->Close;
```

or:

```
MyConnection1->Connected = False;
```

Additional Information

MyDAC offers a wide set of features to achieve better performance, balance network load, and enable additional capabilities, for example:

- Local Failover

- Connection Pooling

- Disconnected Mode

- Support for Unicode

- Data Type Mapping

See Also

- TMyConnection

- Server

- Port

- Database

- Username

- Password

- LoginPrompt

- ConnectString

Request Support   DAC Forum   Provide Feedback

3.5 Connecting to MySQL Embedded

This tutorial describes how to connect to MySQL Embedded.

Contents

Requirements

In order to connect to MySQL Embedded Server, you need MySQL Embedded Server files, MyDAC installed, and IDE running. Also, if authentication is enabled in the settings of MySQL Embedded Server, you need to know the login and password (by default, authentication is disabled).

General information

Components for connection to MySQL Embedded Server

It is possible to connect to MySQL Embedded Server using both TMyEmbConnection and TMyConnection components. But, in contrast to TMyConnection, TMyEmbConnection doesn't require my.ini and allows to set connection with MySQL Embedded Server via the

TMyEmbConnection.Params, TMyEmbConnection.BaseDir and TMyEmbConnection.DataDir properties.

Installation of MySQL Embedded Server

- Place libmysqld.dll to a directory with the application or in one of directories listed in the PATH environmental variable.

- In the directory with the application, create the **share\english** subdirectory and place there the errmsg.sys file - a MySQL file containing error messages in english.

- In the directory with the application, create the **data** subdirectory - a directory for database folders.

- In the "data" directory, create the **test** subdirectory - this one will be a **test** database folder that we will work with further.

- If you use TMyConnection, you will need the my.ini file, that should be located in the **C:** or **C:\Windows** directory and edited by adding or replacing the following lines:

```
[embedded]
    basedir=./
    datadir=./data/
```

- If you need stored procedure functionality, user access rights, etc., copy the **mysql** folder to the **data** folder.

Creating Connection

Design time creation

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

TMyEmbConnection:

1. Open the Component palette and find the TMyEmbConnection component in the MyDAC category.

2. Double-click on the component. Note that a new object appears on the form. If this is the first time you create TMyEmbConnection in this application, it is named

MyEmbConnection1.

TMyConnection:

1. Open the Component palette and find the TMyConnection component in the MyDAC category.

2. Double-click on the component. Note that a new object appears on the form. If this is the first time you create TMyConnection in this application, it is named MyConnection1.

After you have done these steps, you should set up the newly created MyEmbConnection1 or MyConnection1 component. You can do this in two ways:

Using Connection Editor

TMyEmbConnection:

1. Double-click on the created connection object (MyEmbConnection1 or MyConnection1).

2. For the MyConnection1 object enable the Embedded check box.

3. In the **Database** edit box specify the database name (for example, **test**). If the specified database does not exist, you will get an error on connection.

4. If authentication is enabled, specify the user name in the Username edit box and the password in the Password edit box.

Using Object Inspector

TMyEmbConnection:

1. Click on the created connection object (MyEmbConnection1 or MyConnection1) and press **F11** to focus on the object properties.

2. For MyConnection1 object set Options.Embedded to True.

3. In the **Database** edit box specify the database name (for example, **test**). If the specified database does not exist, you will get an error on connection.

4. If authentication is enabled, specify the user name in the Username property and the password in the Password property.

Run time creation

The same operations performed in runtime look as follows:

TMyEmbConnection:

[Delphi]

```
procedure TMainForm.ButtonConnectClick(Sender: TObject);
var
  con: TMyEmbConnection;
begin
  con := TMyEmbConnection.Create(nil);
  try
    con.Database := 'test'; // if the specified database does not exist, you
    con.Password := 'password'; // if authentication is enabled
    con.Username := 'username'; // if authentication is enabled
    con.LoginPrompt := False; // to prevent showing of the connection dialog
    con.Open;
  finally
    con.Free;
  end;
end;
```

Note: To run this code, you have to add the MSCompactConnection and OLEDBAccess

units to the USES clause of your unit.

[C++Builder]

```
void __fastcall TMainForm::ButtonConnectClick(TObject *Sender)
{
  TMyEmbConnection* con = new TMyEmbConnection(NULL);
  try
  {
    con->Database = "test"; // if the specified database does not exist, you
    con->Password = "password"; // if authentication is enabled
    con->Username = "username"; // if authentication is enabled
    con->LoginPrompt = False; // to prevent showing of the connection dialog
    con->Open();
  }
  __finally
  {
    con->Free();
  }
}
```

Note: To run this code, you have to include the MyEmbConnection.hpp header file to your

unit.

TMyConnection:

[Delphi]

```
procedure TMainForm.ButtonConnectClick(Sender: TObject);
var
  con: TMyConnection;
begin
  con := TMyConnection.Create(nil);
  try
    con.Options.Embedded := True;  // enable embedded mode
    con.Database := 'test'; // if the specified database does not exist, you
    con.Password := 'password'; // if authentication is enabled
    con.Username := 'username'; // if authentication is enabled
    con.LoginPrompt := False; // to prevent showing of the connection dialog
    con.Open;
  finally
    con.Free;
  end;
end;
```

Note: To run this code, you have to add the MSAccess unit to the USES clause of your unit.

[C++ Builder]

```
void __fastcall TMainForm::ButtonConnectClick(TObject *Sender)
{
  TMyConnection* con = new TMyConnection(NULL);
  try
  {
    con->Options->Embedded = True;  // enable embedded mode
    con->Database = "database"; // if the specified database does not exist,
    con->Password = "password"; // if authentication is enabled
    con->Username = "username"; // if authentication is enabled
    con->LoginPrompt = False; // to prevent showing of the connection dialog
    con->Open();
  }
  __finally
  {
    con->Free();
  }
}
```

Note: To run this code, you have to include the MSAccess.hpp header file to your unit.

Opening connection

As you can see above, opening connection at run-time is as simple as calling of the Open method:

[Delphi]

```
con.Open;
```

[C++ Builder]

```
con->Open();
```

Another way to open connection at run-time is to set the Connected property to True:

[Delphi]

```
con.Connected := True;
```

[C++ Builder]

```
con->Connected = True;
```

This way can be used at design-time as well. Of course, the connection (TMyEmbConnection or TMyConnection) must have valid connection options assigned earlier. When you call Open, MyDAC tries to open the database. If any problem occurs, it raises an exception with brief explanation on what is wrong. If no problem is encountered and the database is opened, the Open method returns and the Connected property is changed to True.

Closing connection

To close connection, call its Close method, or set its Connected property to False:
[Delphi]

```
con.Close;
```

[C++ Builder]

```
con.Close();
```

or:

[Delphi]

```
con.Connected := False;
```

[C++ Builder]

```
con.Connected = False;
```

Modifying connection

You can modify connection by changing the properties of the TMyEmbConnection or TMyConnection component. Keep in mind that while some of the properties can be altered freely, most of them close connection when a new value is assigned. For example, if you

change the Username property, it is closed immediately, and you have to reopen it manually.

Additional information

To connect to MySQL Embedded Server versions 5.6.x, it is necessary to set the MySQLEmbDisableEventLog global variable (declared in the MySqlApi module) to True before opening the connection. You can read additional information about connecting to MySQL Embedded Server in the Embedded Server article.

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

- Connection Pooling

- Disconnected Mode

- Data Type Mapping

See Also
- TMyEmbConnection

- TMyConnection

- Server

- Database

- Username

- Password

- LoginPrompt

Request Support   DAC Forum   Provide Feedback

3.6   Creating Database Objects

This tutorial describes how to create database objects in MySQL using the TMyCommand and TMyScript components.

1. Requirements

Requirements

This tutorial assumes that you have already connected to the server (see Connecting to MySQL and Connecting to MySQL Embedded). To create database objects at runtime, add the MyAccess and MyScript units to the `uses` clause for Delphi or include the `MyAccess.hpp` and `MyScript.hpp` header files for C++ Builder.

General Information

Database objects are created using Data Definition Language (DDL), which is part of the SQL language. The user must have the appropriate privileges to execute DDL statements on the server. There are two ways to create database objects: build DDL statements manually and execute them with a component like `TMyCommand`, or use GUI tools for databases like dbForge Studio for MySQL. This tutorial uses the data access components to create tables and stored procedures.

Creating Tables

To create tables, the `TMyCommand` component is used in this tutorial.

Design-Time

- Find the `TMyCommand` component in the MyDAC category on the Tool Palette.

- Double-click the component. A new object will appear on the form. If this is the first `TMyCommand` object in this project, it will be named `MyCommand1`. Note that the Connection property is automatically set to an existing connection.

- Double-click the `MyCommand1` object.

- Enter the following statements:

```
CREATE TABLE dept (
    deptno INT NOT null PRIMARY KEY,
    dname VARCHAR(14),
    loc VARCHAR(13)
);
CREATE TABLE emp (
    empno INT NOT null PRIMARY KEY,
    ename VARCHAR(10),
    job VARCHAR(9),
    mgr INT,
 hiredate TIMESTAMP,
 sal INT,
 comm INT,
 deptno INT REFERENCES dept (deptno)
);
```

- Click the `Execute` button to create two tables.

Runtime

The same tables created at runtime:

Delphi

```
var
  MyCommand1: TMyCommand;
begin
  MyCommand := TMyCommand.Create(nil);
  try
    // MyConnection1 (or MyEmbConnection1) was set up earlier
    MyCommand1.Connection := MyConnection1;
    //adds statements to create tables
    MyCommand1.SQL.Add('CREATE TABLE dept (');
    MyCommand1.SQL.Add('   deptno INT NOT NULL PRIMARY KEY,');
    MyCommand1.Add('  dname VARCHAR(14),');
    MyCommand1.SQL.Add('   loc VARCHAR(13)');
    MyCommand1.SQL.Add(');');
    MyCommand1.SQL.Add('CREATE TABLE emp (');
    MyCommand1.SQL.Add('   empno INT NOT NULL PRIMARY KEY,');
    MyCommand1.SQL.Add('   ename VARCHAR(10),');
    MyCommand1.SQL.Add('   job VARCHAR(9),');
```

```
    MyCommand1.SQL.Add('   mgr INT,');
    MyCommand1.SQL.Add('   hiredate TIMESTAMP,');
    MyCommand1.SQL.Add('   sal INT,');
    MyCommand1.SQL.Add('   comm INT,');
    MyCommand1.SQL.Add('   deptno INT REFERENCES dept (deptno)');
    MyCommand1.SQL.Add(');');
    // executes the statements
    MyCommand1.Execute;
  finally
    MyCommand1.Free;
  end;
end;
```

C++Builder

```
TMyCommand* MyCommand1 = new TMyCommand(NULL);
try {
    // MyConnection1 (or MyEmbConnection1) was set up earlier
 MyCommand1->Connection = MyConnection1;
 //adds statements to create tables
 MyCommand1->SQL->Add("CREATE TABLE dept (");
 MyCommand1->SQL->Add("   deptno INT NOT NULL PRIMARY KEY,");
 MyCommand1->SQL->Add("   dname VARCHAR(14),");
 MyCommand1->SQL->Add("   loc VARCHAR(13)");
 MyCommand1->SQL->Add(");");
 MyCommand1->SQL->Add("CREATE TABLE emp (");
 MyCommand1->SQL->Add("   empno INT NOT NULL PRIMARY KEY,");
 MyCommand1->SQL->Add("   ename VARCHAR(10),");
 MyCommand1->SQL->Add("   job VARCHAR(9),");
 MyCommand1->SQL->Add("   mgr INT,");
 MyCommand1->SQL->Add("   hiredate TIMESTAMP,");
 MyCommand1->SQL->Add("   sal INT,");
 MyCommand1->SQL->Add("   comm INT,");
 MyCommand1->SQL->Add("   deptno INT REFERENCES dept (deptno)");
 MyCommand1->SQL->Add(");");
 // executes the statements
 MyCommand1->Execute();
}
__finally {
 MyCommand1->Free();
}
```

Creating Stored Procedures

To create stored procedures, the TMyScript component is used in this tutorial. MySQL supports stored procedures since version 5.6.

Design-Time

- Find the TMyScript component in the MyDAC category on the Tool Palette.

- Double-click the component. A new object will appear on the form. If this is the first TMyScript object in this project, it will be named MyScript1. Note that the Connection

property is already set to an existing connection.

- Double-click the `MyScript1` object.

- Enter the following statement:

```
DELIMITER $$
CREATE PROCEDURE TenMostHighPaidEmployees()
BEGIN
 SELECT emp.ename, emp.sal FROM emp ORDER BY emp.sal DESC LIMIT 10;
END $$
CREATE PROCEDURE GetEmpNumberInDept (IN deptno INT, OUT empnumb INT)
BEGIN
SELECT COUNT(*) INTO empnumb FROM EMP WHERE EMP.deptno = deptno;
END $$
DELIMITER ;
```

- Click the `Execute` button to create two stored procedures.

Runtime

The same stored procedures created at runtime:

Delphi

```
var
  MyScript1: TMyScript;
begin
  MyScript1 := TMyScript.Create(nil);
  try
    // MyConnection1 (or MyEmbConnection1) was set up earlier
    IBCScript1.Connection := IBCConnection1;
    // adds statements to create procedures
    MyScript1.SQL.Add('DELIMITER $$');
    MyScript1.SQL.Add('BEGIN');
    MyScript1.SQL.Add('  SELECT emp.ename, emp.sal FROM emp ORDER BY emp.sal
    MyScript1.SQL.Add('END $$');
 MyScript1.SQL.Add('CREATE PROCEDURE GetEmpNumberInDept (IN deptno INT, OUT
    MyScript1.SQL.Add('BEGIN');
    MyScript1.SQL.Add('  SELECT COUNT(*) INTO empnumb FROM EMP WHERE EMP.dep
    MyScript1.SQL.Add('END $$');
    MyScript1.SQL.Add('DELIMITER ;');
    // executes the statements
    MyScript1.Execute;
  finally
    MyScript1.Free;
  end;
end;
```

C++Builder

```
TMyScript* MyScript1 = new TIBCScript(NULL);
try {
    // MyConnection1 (or MyEmbConnection1) was set up earlier
 MyScript1->Connection = IBCConnection1;
 // adds statements to create procedures
    MyScript1->SQL->Add("DELIMITER $$");
    MyScript1->SQL->Add("BEGIN");
    MyScript1->SQL->Add(" SELECT emp.ename, emp.sal FROM emp ORDER BY emp.sa
    MyScript1->SQL->Add("END $$");
    MyScript1->SQL->Add("  FOR");
    MyScript1->SQL->Add("CREATE PROCEDURE GetEmpNumberInDept (IN deptno INT,
 MyScript1->SQL->Add("BEGIN");
    MyScript1->SQL->Add("  SELECT COUNT(*) INTO empnumb FROM EMP WHERE EMP.d
    MyScript1->SQL->Add("END $$");
    MyScript1->SQL->Add("DELIMITER ;");
    // executes the statements
    MyScript1->Execute;
}
__finally {
 MyScript1->Free();
```

Additional Information

There are many ways to create database objects on the server. Any tool or component that is capable of running an SQL query can be used to manage database objects. For example, `TMyCommand` can be used to insert statements one by one, while `TMyScript` is intended to execute multiple DDL/DML statements as a single SQL script. For more information on DDL statements, refer to the MySQL documentation.

© 1997-2025
Devart. All Rights   Request Support   DAC Forum   Provide Feedback
Reserved.

3.7   Deleting Data From Tables

This tutorial describes how to delete data from tables using the TMyQuery and TMyTable components.

5. Additional Information

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

Data on server can be deleted using Data Manipulation Language (DML), which is a part of SQL. DML statements can be executed on server by an account that has necessary privileges. There are two ways to manipulate a database. You can build DML statements manually and run them within some component like TMyQuery. Another way is to use the dataset functionality (the Delete method) of the TMyQuery and TMyTable components. We will discuss both ways. The goal of this tutorial is to delete a record in the table dept.

Using DataSet Functionality

The Delete method of the TMyQuery and TMyTable components allows deleting data without using DML statements. DML statements are generated by MyDAC components internally. The code below demonstrates using this method:

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
```

```
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // retrieve data
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    // delete the current record
    q.Delete;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // retrieve data
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
    // delete the current record
    q->Delete();
  }
  __finally
  {
    q->Free();
  }
}
```

Building DML Statements Manually

DML Statements can contain plain text and text with parameters. This section describes both
ways.

DML Statements With Parameters

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for delete record
    q.SQL.Clear;
    q.SQL.Add('DELETE FROM dept WHERE deptno = :deptno;');
    // set parameters
    q.ParamByName('deptno').AsInteger := 10;
```

```
    // execute query
    q.Execute;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // set SQL query for delete record
    q->SQL->Clear();
    q->SQL->Add("DELETE FROM dept WHERE deptno = :deptno;");
    // set parameters
    q->ParamByName("deptno")->AsInteger = 10;
    // execute query
    q->Execute();
  }
  __finally
  {
    q->Free();
  }
}
```

DML Statements As Plain Text

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for delete record
    q.SQL.Clear;
    q.SQL.Add('DELETE FROM dept WHERE deptno = 10;');
    // execute query
    q.Execute;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
```

```
{
  // con is either TMyConnection or TMyEmbConnection already set up
  q->Connection = con;
  // set SQL query for delete record
  q->SQL->Clear();
  q->SQL->Add("DELETE FROM dept WHERE deptno = 10;");
  // execute query
  q->Execute();
}
__finally
{
  q->Free();
}
}
```

Additional Information

It is also possible to use stored procedures for deleting data. In this case, all data manipulation logic is defined on server. You can find more about using stored procedures in the tutorial "Stored Procedures".

Request Support   DAC Forum   Provide Feedback

3.8   Inserting Data Into Tables

This tutorial describes how to insert data into tables using the TMyQuery and TMyTable components.

1. Requirements

2. General information

3. Design Time

4. Run Time

- 4.1 Using DataSet Functionality

- 4.2 Building DML Statements Manually

  - 4.2.1 DML Statements With Parameters

  - 4.2.2 DML Statements As Plain Text

5. Additional Information

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded") and that necessary objects are already created on the server (tutorial "Creating Database Objects").

General information

Data on server can be inserted using Data Manipulation Language (DML), which is a part of SQL. DML statements can be executed on server by an account that has necessary privileges. There are two ways to manipulate a database. You can build DML statements manually and run them within some component like TMyQuery. Another way is to use the dataset functionality (the Insert, Append, and Post methods) of the TMyQuery and TMyTable components. We will discuss both ways.

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

Table dept

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

Table emp

ename	job	mgr	hiredate	sal	comm	deptno
SMITH	CLERK	7902	17.12.1980	800	NULL	20
ALLEN	SALESMAN	7698	20.02.1981	1600	300	30
WARD	SALESMAN	7698	22.02.1981	1250	500	30
JONES	MANAGER	7839	02.04.1981	2975	NULL	20
MARTIN	SALESMAN	7698	28.09.1981	1250	1400	30
BLAKE	MANAGER	7839	01.05.1981	2850	NULL	30
CLARK	MANAGER	7839	09.06.1981	2450	NULL	10
SCOTT	ANALYST	7566	13.07.1987	3000	NULL	20
KING	PRESIDENT	NULL	17.11.1981	5000	NULL	10
TURNER	SALESMAN	7698	08.09.1981	1500	0	30
ADAMS	CLERK	7788	13.07.1987	1100	NULL	20
JAMES	CLERK	7698	03.12.1981	950	NULL	30
FORD	ANALYST	7566	03.12.1981	3000	NULL	20
MILLER	CLERK	7782	23.01.1982	1300	NULL	10

Note: The empno field of the emp table is an IDENTITY(1,1) (i.e. autoincrement) field, so its value is filled automatically by the server.

Design time

- Open the Component palette and find the TMyQuery component in the MyDAC category.

- Double-click on the component. Note that a new object appears on the form. If this is the first time you create TMyQuery in this application, it is named MSQuery1. Note that the MSQuery1.Connection property is already set to an existent (on the form) connection.

- Double-click on the MSQuery1 object.

- Type the following lines:

```
INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');
```

- Press the Execute button.

Performing these steps adds a new record to the dept table.

Run time

Using DataSet Functionality

The Insert, Append, and Post methods of the TMyQuery and TMyTable components allow inserting data not using DML statements. DML statements are generated by MyDAC components internally. The difference between the Append and Insert methods is that Append creates a new empty record in the end of a dataset, when Insert creates it in the position of the current record of a dataset. The code below demonstrates using these methods:

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    q.Connection := con; // con is either TMyConnection or TMyEmbConnection
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    q.Append;
    q.FieldByName('deptno').AsInteger := 10;
    q.FieldByName('dname').AsString := 'ACCOUNTING';
    q.FieldByName('loc').AsString := 'NEW YORK';
    q.Post;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    q->Connection = con; // con is either TMyConnection or TMyEmbConnection
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
```

```
    q->Append();
    q->FieldByName("deptno")->AsInteger = 10;
    q->FieldByName("dname")->AsString = "ACCOUNTING";
    q->FieldByName("loc")->AsString = "NEW YORK";
    q->Post();
  }
  __finally
  {
    q->Free();
  }
}
```

Building DML Statements Manually

DML Statements can contain plain text and text with parameters. This section describes both ways.

DML Statements With Parameters

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    q.Connection := con; // con is either TMyConnection or TMyEmbConnection
    q.SQL.Clear;
    q.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dname,
    q.ParamByName('deptno').AsInteger := 10;
    q.ParamByName('dname').AsString := 'ACCOUNTING';
    q.ParamByName('loc').AsString := 'NEW YORK';
    q.Execute;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    q->Connection = con; // con is either TMyConnection or TMyEmbConnection
    q->SQL->Clear();
    q->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dnam
    q->ParamByName("deptno")->AsInteger = 10;
    q->ParamByName("dname")->AsString = "ACCOUNTING";
    q->ParamByName("loc")->AsString = "NEW YORK";
    q->Execute();
  }
  __finally
```

```
    {
      q->Free();
    }
}
```

DML Statements As Plain Text

[Delphi]

```delphi
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    q.Connection := con; // con is either TMyConnection or TMyEmbConnection
    q.SQL.Clear;
    q.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (10,''ACCOUNTING'
    q.Execute;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```cpp
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    q->Connection = con; // con is either TMyConnection or TMyEmbConnection
    q->SQL->Clear();
    q->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (10,'ACCOUNTING
    q->Execute();
  }
  __finally
  {
    q->Free();
  }
}
```

Additional Information

Actually, there are lots of ways to insert data into tables. Any tool or component capable of running a SQL query can be used to manage data. Some components are better for performing certain tasks. For example, TMyLoader is the fastest way to insert data, TMyScript is designed for executing series of statements one by one.

It is also possible to use stored procedures for inserting data. In this case, all data manipulation logic is defined on the server. You can find more about using stored procedures in the tutorial "Stored Procedures".

Request Support   DAC Forum   Provide Feedback

3.9   Retrieving Data

This tutorial describes how to retrieve data from tables using the TMyQuery and TMyTable components.

1. Requirements

2. General Information

3. TMyQuery

4. TMyTable

5. Additional information

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded Server"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

As we know, an original function of any database application is establishing connection to a data source and working with data contained in it. MyDAC provides several components that can be used for data retrieving, such as TMyQuery and TMyTable. We will discuss data retrieving using these components.

The goal of this tutorial is to retrieve data from a table dept.

TMyQuery

The following code demonstrates retrieving of data from the dept table using the TMyQuery component:

[Delphi]

```
var
  q: TMyQuery;
```

```
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // retrieve data
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(q.RecordCount));
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // retrieve data
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(q->RecordCount));
  }
  __finally
  {
    q->Free();
  }
}
```

TMyTable

The following code demonstrates retrieving of data from the dept table using the TMyTable component:

[Delphi]

```
var
  tbl: TMyTable;
begin
  tbl := TMyTable.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    tbl.Connection := con;
    // retrieve data
    tbl.TableName := 'dept';
    tbl.Open;
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(tbl.RecordCount));
```

```
    finally
      tbl.Free;
    end;
end;
```

[C++Builder]

```
{
  TMyTable* tbl = new TMyTable(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    tbl->Connection = con;
    // retrieve data
    tbl->TableName = "dept";
    tbl->Open();
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(tbl->RecordCount));
  }
  __finally
  {
    tbl->Free();
  }
}
```

Additional Information

It is also possible to use stored procedures for data retrieving. In this case, all data manipulation logic is defined on server. You can find more about using stored procedures in the tutorial "Stored Procedures".

Request Support   DAC Forum   Provide Feedback

3.10 Modifying Data

This tutorial describes how to modify data in tables using the TMyQuery and TMyTable components.

5. Additional Information

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

Data on server can be modified using Data Manipulation Language (DML), which is a part of SQL. DML statements can be executed on server by an account that has necessary privileges. There are two ways to manipulate a database. You can build DML statements manually and run them within some component like TMyQuery. Another way is to use the dataset functionality (the Edit and Post methods) of the TMyQuery and TMyTable components. We will discuss both ways. The goal of this tutorial is to modify the following record of the table dept:

10	ACCOUNTING	NEW YORK

to make it look as follows:

10	RESEARCH	LOS ANGELES

Using DataSet Functionality

The Edit and Post methods of the TMyQuery and TMyTable components allow deleting data without using DML statements. DML statements are generated by MyDAC components internally. The code below demonstrates using these methods:

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // retrieve data
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    // to make the record with deptno=10 the current record
```

```
      q.FindKey([10]);
      // modify record
      q.Edit;
      q.FieldByName('dname').AsString := 'RESEARCH';
      q.FieldByName('loc').AsString := 'LOS ANGELES';
      q.Post;
  finally
      q.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // retrieve data
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
    // to make the record with deptno=10 the current record
    q->FindKey(ARRAYOFCONST((10)));
    // modify record
    q->Edit();
    q->FieldByName("dname")->AsString = "RESEARCH";
    q->FieldByName("loc")->AsString = "LOS ANGELES";
    q->Post();
  }
  __finally
  {
    q->Free();
  }
}
```

Building DML Statements Manually

DML Statements can contain plain text and text with parameters. This section describes both ways.

DML Statements With Parameters

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for update record
```

```
      q.SQL.Clear;
      q.SQL.Add('UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :de
      // set parameters
      q.ParamByName('deptno').AsInteger := 10;
      q.ParamByName('dname').AsString := 'RESEARCH';
      q.ParamByName('loc').AsString := 'LOS ANGELES';
      // execute query
      q.Execute;
    finally
      q.Free;
    end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // set SQL query for update record
    q->SQL->Clear();
    q->SQL->Add("UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :
    // set parameters
    q->ParamByName("deptno")->AsInteger = 10;
    q->ParamByName("dname")->AsString = "RESEARCH";
    q->ParamByName("loc")->AsString = "LOS ANGELES";
    // execute query
    q->Execute();
  }
  __finally
  {
    q->Free();
  }
}
```

DML Statements As Plain Text

[Delphi]

```
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for update record
    q.SQL.Clear;
    q.SQL.Add('UPDATE dept SET dname = ''RESEARCH'', loc = ''LOS ANGELES'' W
    // execute query
    q.Execute;
  finally
    q.Free;
```

```
   end;
end;
```

[C++Builder]

```
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // set SQL query for update record
    q->SQL->Clear();
    q->SQL->Add("UPDATE dept SET dname = 'RESEARCH', loc = 'LOS ANGELES' WHE
    // execute query
    q->Execute();
  }
  __finally
  {
    q->Free();
  }
}
```

Additional Information

It is also possible to use stored procedures for modifying data. In this case, all data manipulation logic is defined on server. You can find more about using stored procedures in the tutorial "Stored Procedures".

Request Support   DAC Forum   Provide Feedback

3.11 Stored Procedures

This tutorial describes how to work with stored procedures using the TMyStoredProc component and insert data into tables.

1. Requirements

2. General information

3. Input parameters

4. Output parameters

5. Input/output parameters

6. Using MySQL Stored Functions

7. Returning result sets

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

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

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

Note: stored procedures and stored functions are supported since MySQL 5.0.

Input parameters

Input parameter (IN) is a parameter which value is passed into a stored procedure/function module. The procedure might modify the value, but the modification is not visible to the caller when the procedure is returned. The following procedure inserts a new row into the table dept:

```
CREATE PROCEDURE InsertDept(
IN p_deptno INT,
IN p_dname VARCHAR(14),
IN p_loc VARCHAR(13)
)
BEGIN
  INSERT INTO dept(deptno, dname, loc) VALUES(p_deptno, p_dname, p_loc);
END;
```

It needs to receive the values to be inserted into the new record, and thus the procedure has

three input parameters, corresponding to each field of the table. This procedure may be executed as follows:

```
call InsertDept(10,'ACCOUNTING','NEW YORK');
```

To execute the InsertDept stored procedure using the TMyStoredProc component, the following code can be used:

[Delphi]

```
var
  sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored procedure name to execute
    sp.StoredProcName := 'InsertDept';
    // build a query for a chosen stored procedure based on the Params and S
    sp.PrepareSQL;
    // assign parameter values
    sp.ParamByName('p_deptno').AsInteger := 10;
    sp.ParamByName('p_dname').AsString := 'ACCOUNTING';
    sp.ParamByName('p_loc').AsString := 'NEW YORK';
    // execute the stored procedure
    sp.Execute;
  finally
    sp.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    sp->Connection = con;
    // choose a stored procedure name to execute
    sp->StoredProcName = "InsertDept";
    // build a query for chosen stored procedure based on the Params and Sto
    sp->PrepareSQL();
    // assign parameter values
    sp->ParamByName("p_deptno")->AsInteger = 10;
    sp->ParamByName("p_dname")->AsString = "ACCOUNTING";
    sp->ParamByName("p_loc")->AsString = "NEW YORK";
    // execute the stored procedure
    sp->Execute();
  }
  __finally
  {
    sp->Free();
```

```
   }
}
```

Output parameters

Output parameter (OUT) is a parameter which value is passed out of the stored procedure/
function module. Its initial value is NULL within the procedure, and its value is visible to the
caller when the procedure is returned. The following stored procedure returns the count of
records in the table dept:

```
CREATE PROCEDURE CountDept (
OUT cnt INT
)
BEGIN
  SELECT count(*) FROM dept into cnt;
END;
```

To execute the CountDept stored procedure using the TMyStoredProc component, the
following code can be used:

[Delphi]

```
var
  sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored procedure name to execute
    sp.StoredProcName := 'CountDept';
    // build a query for chosen stored procedure based on the Params and Sto
    sp.PrepareSQL;
    // execute the stored procedure
    sp.Execute;
    // show the value of the output parameter
    ShowMessage(IntToStr(sp.ParamByName('cnt').AsInteger));
  finally
    sp.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    sp->Connection = con;
    // choose a stored procedure name to execute
    sp->StoredProcName = "CountDept";
```

```
      // build a query for chosen stored procedure based on the Params and Sto
      sp->PrepareSQL();
      // execute the stored procedure
      sp->Execute();
      // show the value of the output parameter
      ShowMessage(IntToStr(sp->ParamByName("cnt")->AsInteger));
    }
    __finally
    {
      sp->Free();
    }
}
```

Input/output parameters

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

For example, the following stored procedure returns the salary with five percents bonus:

```
CREATE PROCEDURE GiveBonus(INOUT sal FLOAT)
BEGIN
  SET sal = sal * 1.05;
END
```

To execute the GiveBonus stored procedure using the TMyStoredProc component, the following code can be used:

[Delphi]

```
var
  sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored procedure name to execute
    sp.StoredProcName := 'GiveBonus';
    // build a query for chosen stored procedure based on the Params and Sto
    sp.PrepareSQL;
    // assign parameter values
    sp.ParamByName('sal').AsFloat := 500.5;
    // execute the stored procedure
    sp.Execute;
    // show the value of the input/output parameter
    ShowMessage(sp.ParamByName('sal').AsString);
  finally
```

```
        sp.Free;
    end;
end;
```

[C++Builder]

```
{
  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    sp->Connection = con;
    // choose a stored procedure name to execute
    sp->StoredProcName = "GiveBonus";
    // build a query for chosen stored procedure based on the Params and Sto
    sp->PrepareSQL();
    // assign parameter values
    sp->ParamByName("sal")->AsFloat = 500.5;
    // execute of the stored procedure
    sp->Execute();
    // show the value of the input/output parameter
    ShowMessage(sp->ParamByName("sal")->AsString);
  }
  __finally
  {
    sp->Free();
  }
}
```

Using MySQL Stored Functions

The tasks described above can be performed using stored functions. For example, the following stored function returns the bonus salary like the GiveBonus stored procedure:

```
CREATE FUNCTION GiveBonusFunc(sal FLOAT)
RETURNS FLOAT
BEGIN
  RETURN sal * 1.05;
END
```

This function may be executed as follows:

```
select GiveBonusFunc(500.5);
```

To execute the GiveBonusFunc stored function using the TMyStoredProc component, the following code can be used:

[Delphi]

```
var
  sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
```

```
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored function name to execute
    sp.StoredProcName := 'GiveBonusFunc';
    // build a query for chosen stored procedure based on the Params and Sto
    sp.PrepareSQL;
    // assign parameter values
    sp.ParamByName('sal').AsFloat := 500.5;
    // execute the stored procedure
    sp.Execute;
    // show the returned value
    ShowMessage(sp.ParamByName('result').AsString);
  finally
    sp.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    sp->Connection = con;
    // choose a stored function name to execute
    sp->StoredProcName = "GiveBonusFunc";
    // build a query for chosen stored procedure based on the Params and Sto
    sp->PrepareSQL();
    // assign parameter values
    sp->ParamByName("sal")->AsFloat = 500.5;
    // execute of the stored procedure
    sp->Execute();
    // show the returned value
    ShowMessage(sp->ParamByName("result")->AsString);
  }
  __finally
  {
    sp->Free();
  }
}
```

Note: To retrieve the result returned by the stored function using TMyStoredProc, the 'result' parameter created automatically should be used.

Returning result sets

Besides scalar variables, a stored procedure can return result sets, i.e. the results of a SELECT statement. This question is discussed in details in the tutorial "Working With Result Sets Using Stored Procedures".

Request Support   DAC Forum   Provide Feedback

Reserved.

3.12 Working With Result Sets Using Stored Procedures

This tutorial describes how to retrieve and modify result sets obtained from stored procedures using the TMyStoredProc component.

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

Besides scalar variables, stored procedures can return result sets, i.e. the results of SELECT statements. Data can be inserted or modified in obtained result sets using the dataset functionality of the TMyStoredProc component.

The goal of this tutorial is to retrieve and modify data from the dept table using the TMyStoredProc component. The following stored procedure will be used to retrieve data:

```
CREATE PROCEDURE SelectDept()
BEGIN
  SELECT * FROM dept;
END
```

Using DataSet functionality

The Insert, Append, Edit, and Post methods of the TMyStoredProc component can be used to insert and modify data in obtained result sets. DML statements are generated by TMyStoredProc internally. The code below demonstrates using these methods:

[Delphi]

```
var
  sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored procedure name
```

```
      sp.StoredProcName := 'SelectDept';
      // build a query for a chosen stored procedure based on the Params and S
      sp.PrepareSQL;
      // retrieve data
      sp.Open;
      // append record
      sp.Append;
      sp.FieldByName('deptno').AsInteger := 50;
      sp.FieldByName('dname').AsString := 'SALES';
      sp.FieldByName('loc').AsString := 'NEW YORK';
      sp.Post;
      // insert record
      sp.Insert;
      sp.FieldByName('deptno').AsInteger := 60;
      sp.FieldByName('dname').AsString := 'ACCOUNTING';
      sp.FieldByName('loc').AsString := 'LOS ANGELES';
      sp.Post;
      // to make the record with deptno=10 the current record
      sp.FindKey([10]);
      // modify record
      sp.Edit;
      sp.FieldByName('dname').AsString := 'RESEARCH';
      sp.FieldByName('loc').AsString := 'LOS ANGELES';
      sp.Post;
  finally
      sp.Free;
  end;
end;
```

[C++Builder]

```
{
  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    sp->Connection = con;
    // choose a stored procedure name
    sp->StoredProcName = "SelectDept";
    // build a query for a chosen stored procedure based on the Params and S
    sp->PrepareSQL();
    // retrieve data
    sp->Open();
    // append record
    sp->Append();
    sp->FieldByName("deptno")->AsInteger = 50;
    sp->FieldByName("dname")->AsString = "SALES";
    sp->FieldByName("loc")->AsString = "NEW YORK";
    sp->Post();
    // insert record
    sp->Insert();
    sp->FieldByName("deptno")->AsInteger = 60;
    sp->FieldByName("dname")->AsString = "ACCOUNTING";
    sp->FieldByName("loc")->AsString = "LOS ANGELES";
    sp->Post();
    // to make the record with deptno=10 the current record
    sp->FindKey(ARRAYOFCONST((10)));
```

```
    // modify record
    sp->Edit();
    sp->FieldByName("dname")->AsString = "RESEARCH";
    sp->FieldByName("loc")->AsString = "LOS ANGELES";
    sp->Post();
  }
  __finally
  {
    sp->Free();
  }
}
```

Request Support   DAC Forum   Provide Feedback

3.13   Demo Projects

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

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

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

Where are the MyDAC demo projects located?

In most cases all the MyDAC demo projects are located in "%MyDac%\Demos\".

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

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

For most new IDEs the structure will be as follows.

```
Demos

  |—MyDacDemo [The main MyDAC demo project]

  |—TechnologySpecific
```

```
|   |— Embedded [Win32 version of the Embedded MySQL server using
demo]
|   |— SecureBridge [A component and a demo for integration with
the SecureBridge library]
|—ThirdParty
|   |— [A collection of demo projects on integration with third-
party components]
|—Miscellaneous
    |— [Some other demo projects on design technologies]
```

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

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

To explore a MyDAC demo project,

1. Launch your IDE.

2. In your IDE, choose File | Open Project from the menu bar.

3. Find the directory you installed MyDAC to and open the Demos folder.

4. Browse through the demo project folders located here and open the project file of the demo you would like to use.

5. Compile and launch the demo. If it exists, consult the *ReadMe.txt* file for more details.

The included sample applications are fully functional. To use the demos, you have to first set up a connection to MySQL. 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 MyDAC demo directory includes two sample SQL scripts for creating and dropping all the test schema objects used in the MyDAC 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 MyDAC demo project in the Getting Started topic. The other MyDAC demo projects include a *ReadMe.txt* file with individual building and launching instructions.

Demo project descriptions

MyDacDemo

MyDacDemo is one large project which includes three collections of demos.

Working with components

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

General demos

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

MySQL-specific demos

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

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

Working with Components

Name	Description
Command	Uses TMyCommand to execute SQL statements. Demonstrates how to execute commands in a separate thread, and how to break long-duration query execution.
ConnectDialog	Demonstrates how to customize the MyDAC connect dialog. Changes the standard MyDAC 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 MyDAC 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.
Loader	Uses the TMyLoader component to quickly load data into a server table. TMyLoader loads data by grouping several data rows into a single INSERT statement and executing this statement. This way is much faster then execuing one INSERT statement per row. This demo also compares the two TMyLoader data loading handlers: GetColumnData and PutData.
Query	Demonstrates working with TMyQuery, which is one of the most useful MyDAC components. Includes many TMyQuery usage scenarios. Demonstrates 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 MyDAC.
StoredProc	Uses TMyStoredProc to access an editable recordset from an MySQL stored procedure in the client application.
Table	Demonstrates how to use TMyTable 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.
UpdateSQL	Demonstrates using the TMyUpdateSQL component to customize update commands. Lets you optionally use TMyCommand and TMyQuery 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 the CachedUpdates mode, including highlighting uncommitted changes, managing transactions, and committing changes in a batch.
FilterAndIndex	Demonstrates MyDAC's local storage functionality. This sample shows how to perform local filtering, sorting and locating by multiple fields, including by calculated and lookup fields.
MasterDetail	Uses MyDAC functionality to work with master/detail relationships. This sample shows how to use local master/detail functionality.

	Demonstrates different kinds of master/detail linking, including linking by SQL, by simple fields, and by calculated fields.
Pictures	Uses MyDAC functionality to work with BLOB fields and graphics. The sample demonstrates how to retrieve binary data from MySQL database and display it on visual components. The sample also shows how to load and save pictures to files and to the database.
Text	Uses MyDAC functionality to work with text. The sample demonstrates how to retrieve text data from MySQL database and display it on visual components. The sample also shows how to load and save text to files and to the database.
Transactions	Demonstrates the recommended approach for managing transactions with the TMyConnection component. The TMyConnection interface provides a wrapper for MySQL server commands like START TRANSACTION, COMMIT, ROLLBACK.

MySQL-specific Demos

Name	Description
Lock	Demonstrates two kinds of row-level locking (immediate locking and delayed locking) with the InnoDB storage engine. This functionality is based on the following MySQL commands: SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE.

Supplementary Demo Projects

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

Location	Name	Description
ThirdParty	**FastReport**	Demonstrates how MyDAC can be used with FastReport components. This project consists of two parts. The first part is several packages that integrate MyDAC components into the FastReport editor. The second part is a demo application that lets you design and preview reports with MyDAC technology in the FastReport editor.
	InfoPower	Uses InfoPower components to display recordsets retrieved with MyDAC. This demo project displays an InfoPower grid component and fills it with the result of an MyDAC query. Shows how to link MyDAC data sources to InfoPower components.

	IntraWeb	A collection of sample projects that show how to use MyDAC 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 MyDAC. This demo project lets you modify the application in design-time.
	ReportBuilder	Uses MyDAC data sources to create a ReportBuilder report that takes data from MySQL database. Shows how to set up a ReportBuilder document in design-time and how to integrate MyDAC components into the Report Builder editor to perform document design in run-time.
Technology Specific	**Embedded**	Demonstrates working with Embedded MySQL server by using the TMyEmbConnection component. This demo creates a database structure, if it does not already exist, opens a table from this database. Also this demo shows how to process the log messages of the Embedded server.
	SecureBridge	The demo project demonstrates how to integrate the SecureBridge components with MyDAC to ensure secure connection to MySQL server through an SSH tunnel and SSL. This demo consists of three parts. The first part is a package that contains TMySSHIOHandler and TMySSLIOHandler component. These components provide integration with the SecureBridge library. The second part is two sample projects that demonstrate how to connect to MySQL server through an SSH server and through SSL, connect to the SSH server with SecureBridge by password or by public key, generate reliable random numbers, enable local port forwarding. For more information see the *Readme.html* file in the demo directory.
Miscellaneous	**CBuilder**	A general demo project about how to create MyDAC-based applications with C++Builder. Lets you execute SQL scripts and work with result sets in a grid. This is one of the two MyDAC demos for C++Builder.
	Dll	Demonstrates creating and loading DLLs for MyDAC-based projects. This demo project consists of two parts - an My_Dll project that creates a DLL of a form that sends a query to the server and displays its results, and an My_Exe project that can be executed to display a form for loading and running this DLL. Allows you to build a dll for one MyDAC-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 MyDAC. 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.
	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
MyDacDemo	*MyDacDemo*	*[Win32 version of the main MyDAC demo project - see above]*

Request Support   DAC Forum   Provide Feedback

3.14 Deployment

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

Deploying Windows applications built without run-time packages

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

You can check if your application does not require run-time packages by making sure the "Build with runtime packages" check box is not selected in the Project Options dialog box.

Trial Limitation Warning

If you are evaluating deploying Windows applications with MyDAC Trial Edition, you will need to deploy the following DAC BPL files and their dependencies (required IDE BPL files) with your application, even if it is built without run-time packages:

dacXX.bpl	always
mydacXX.bpl	always

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

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

Deploying Windows applications built with run-time packages

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

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

dacXX.bpl	always
mydacXX.bpl	always
dacvclXX.bpl	if your application uses the MyDacVcl unit
mydacvclXX.bpl	if your application uses the MyDacVcl unit
crcontrolsXX.bpl	if your application uses the CRDBGrid component

Request Support   DAC Forum   Provide Feedback

4   **Using MyDAC**

This section describes basics of using MySQL Data Access Components

- Updating Data with MyDAC Dataset Components

- Master/Detail Relationships

- [Data Type Mapping](#)

- [Data Encryption](#)

- [Working in an Unstable Network](#)

- [Disconnected Mode](#)

- [Secure Connections](#)

- [Network Tunneling](#)

- [Increasing Performance](#)

- [Macros](#)

- [DataSet Manager](#)

- [Embedded Server](#)

- [National Characters](#)

- [Connection Pooling](#)

- [DBMonitor](#)

- [Writing GUI Applications with MyDAC](#)

- [Compatibility with Previous Versions](#)

- [64-bit Development with Embarcadero RAD Studio XE2](#)

- [Database Specific Aspects of 64-bit Development](#)

- [Demo Projects](#)

- [Deployment](#)

© 1997-2025 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

4.1   Updating Data with MyDAC Dataset Components

MyDAC 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 MyDAC 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 which 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 P:Devart.MyDac.TCustomMyDataSet.UpdateObject property with the TMyUpdateSQL class instance which holds all updating SQL statements in one place. You can generate all these SQL statements using MyDAC design time editors. For more careful customization of data update operations you can use InsertObject, ModifyObject and DeleteObject properties of TMyUpdateSQL component.

See Also

- TMyQuery

- TMyStoredProc

- TMyTable

- TMyUpdateSQL

Request Support   DAC Forum   Provide Feedback

4.2 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 MyDAC implements this feature.

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

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

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

It is necessary to display and edit these tables.

MyDAC provides two ways to bind tables. First code example shows how to bind two TCustomMyDataSet components (TMyQuery or TMyTable) into MD relationship via parameters.

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

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

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

Now suppose that detail table "Department" foreign key field is named DepLink but not Dept_No. In such case detail dataset described in above code example will not autofill DepLink field with current "Department".Dept_No value on insert. This issue is solved in second code example.

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

Request Support   DAC Forum   Provide Feedback

4.3   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, MyDAC 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 MySQL database:

```
CREATE TABLE DECIMAL_TYPES
(
  ID INT(11) NOT NULL AUTO_INCREMENT 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:

MySQL 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       := myDecimal;
  MinPrecision := 0;
  MaxPrecision := 4;
  MinScale     := 0;
  MaxScale     := 0;
  FieldType    := ftSmallint;
  MyConnection.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecision,
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
MyConnection.DataTypeMap.Clear;
// rule for DECIMAL(4,0)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0,    4, 0,    0, ftSma
// rule for DECIMAL(10,0)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 5,    10, 0,    0, ftInt
// rule for DECIMAL(15,0)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 11, rlAny, 0,    0, ftLar
// rule for DECIMAL(5,2)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0,    9, 1, rlAny, ftFlo
// rule for DECIMAL(10,4)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 10, rlAny, 1,    4, ftBCD
// rule for DECIMAL(15,6)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 10, rlAny, 5, rlAny, ftFMT
```

Rules order

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

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

```
CREATE TABLE DECIMAL_TYPES
(
  ID INT(11) NOT NULL AUTO_INCREMENT 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:

MySQL 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:

```
MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0,       9, rlAny, rlAny, ft
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny,      0,     4, ft
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny,      0, rlAny, ft
```

it will lead to the following result:

MySQL 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:

```
MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny,      0, rlAny, ft
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny,      0,     4, ft
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0,       9, rlAny, rlAny, ft
```

it will lead to the following result:

MySQL data type	Delphi field type
DECIMAL(5,2)	ftFMTBCD
DECIMAL(10,4)	ftFMTBCD
DECIMAL(15,6)	ftFMTBCD

This happens because the rule

```
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny,      0, rlAny, ft
```

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 MySQL:

```
CREATE TABLE PERSON
(
  ID INT(11) NOT NULL AUTO_INCREMENT 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:

```
MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDateTime, 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:

```
MyQuery.DataTypeMap.Clear;
MyQuery.DataTypeMap.AddDBTypeRule(myDateTime, 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:

```
MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDateTime, ftDate);
MyQuery.DataTypeMap.Clear;
MyQuery.DataTypeMap.AddDBTypeRule(myDateTime, ftDateTime);
```

In this case, in all DataSets for the DATETIME type fields with the ftDate type will be created, and for MyQuery - 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(11) NOT NULL AUTO_INCREMENT 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:

```
MyQuery.DataTypeMap.Clear;
MyQuery.DataTypeMap.AddDBTypeRule(myChar, 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:

```
MyQuery.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:

```
MyConnection.DataTypeMap.AddDBTypeRule(myVarchar, ftInteger, True);
```

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

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

			an integer value

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

Request Support   DAC Forum   Provide Feedback

4.4   Data Encryption

MyDAC 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(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  ENAME VARBINARY(2000) DEFAULT NULL,
  HIREDATE VARBINARY(200) DEFAULT NULL,
  SAL VARBINARY(200) DEFAULT NULL,
  FOTO BLOB DEFAULT NULL
)
```

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.

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

Request Support   DAC Forum   Provide Feedback

4.5   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), MyDAC can implicitly perform the following operations:

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

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

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

See Also
- FailOver demo

- Working with Disconnected Mode

- TCustomDAConnection.Options

- TCustomDAConnection.Pooling

Request Support   DAC Forum   Provide Feedback

4.6   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.MyDac.TMyQuery.LockMode

- TCustomDAConnection.Pooling

- TCustomDAConnection.Connect

- TCustomDAConnection.Disonnect

- Working in unstable network

© 1997-2025   Request Support   DAC Forum   Provide Feedback

4.7 Secure Connections

This section describes the ways to secure connection between a client and a MySQL server

- Connecting via SSL

- Connecting via SSH

- HTTP/HTTPS Network Tunneling

Request Support   DAC Forum   Provide Feedback

4.7.1 Connecting via SSL

Connecting to MySQL Through SSL in Delphi

Security is very important when sending messages from the server to the client and vice versa. There are many data protection methods, including the use of SSL encryption to connect to a remote MySQL server from a Delphi application. MySQL supports data transfer via the TCP/IP protocol stack both using SSL encryption or without it.

Devart offers a solution called SecureBridge, which allows you to embed an SSL client into a Delphi or C++ Builder application to establish a secure connection to MySQL server. This tutorial demonstrates how to create a sample Delphi application that connects to MySQL using SSL as the encryption method.

Before connecting to MySQL via SSL, create SSL certificates as explained in the MySQL documentation and configure SSL parameters in the my.ini file.

Sample Delphi App That Connects to MySQL Using SSL

To create an SSL connection to MySQL, set the Protocol property in **Options** to **mpSSL**. It forces the application to only connect through SSL - if a connection attempt fails, an exeption is raised.

```
MyConnection.Options.Protocol := mpSSL;
```

Note: Starting with version MyDAC 12.3.0, the additional component TMySSLIOHandler is

considered deprecated and is not recommended for use in new developments. Use the TCRSSLIOHandler component instead. The TMySSLIOHandler component will be provided for backward compatibility, and its source files will be located in the SecureBridge(Deprecated) folder.

After installing MyDAC and SecureBridge in your system, install the TCRSSLIOHandler component in RAD Studio to bind MyDAC with SecureBridge. The installation instructions are provided in the Readme.html, which is located by default in "My Documents\Devart\MyDAC for RAD Studio\Demos\TechnologySpecific\SecureBridge\DelphiXX".

1. Run RAD Studio and select 'File -> New – > VCL Forms Application – Delphi'.

2. Place the **TCRSSLIOHandler** component, which allows MyDAC to connect to MySQL server through SSL, onto the form. Also add the **TMyConnection**, **TMyQuery**, **TDataSource**, **TDBGrid**, and **TButton** components to the form - they are required to create a sample application that connects to the MySQL server via SSL, runs a selection operation against the database, and displays the obtained rows in a data grid.
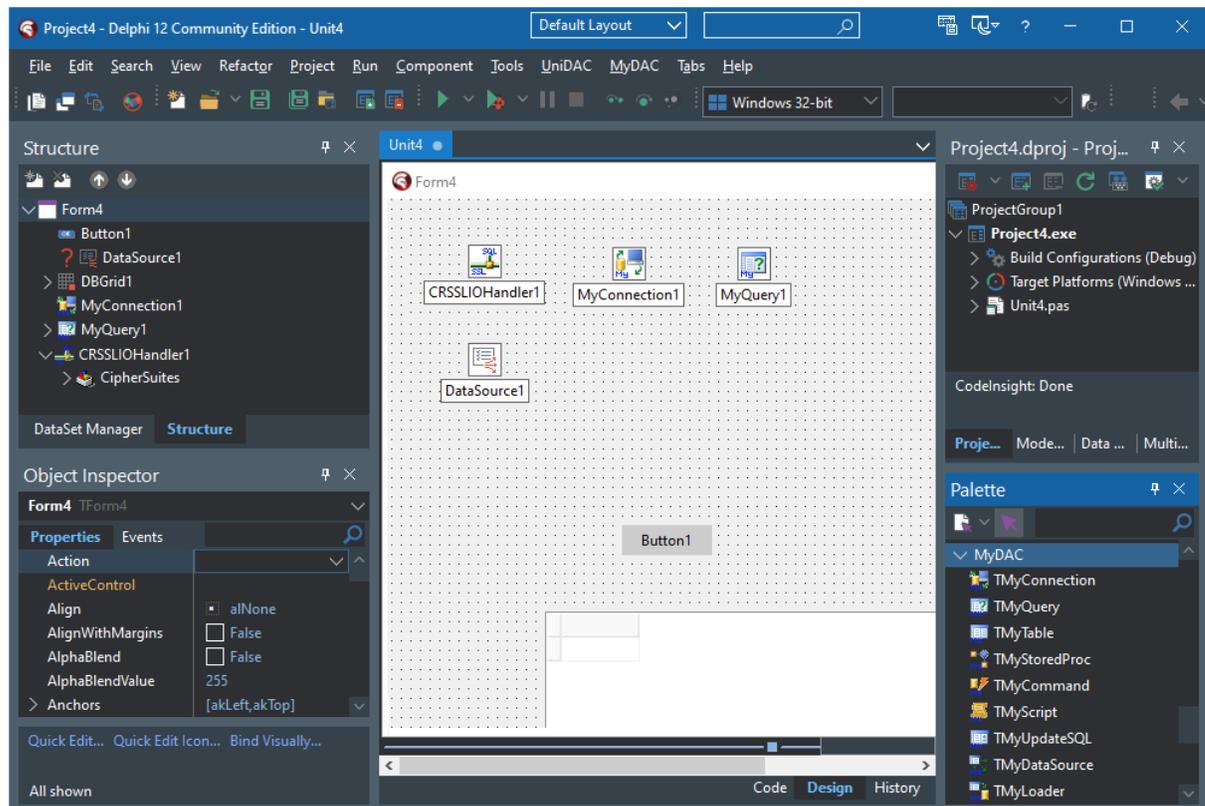
3. Select the **TMyConnection** component and assign the **TCRSSLIOHandler** object to the
IOHandler property in the Object Inspector.
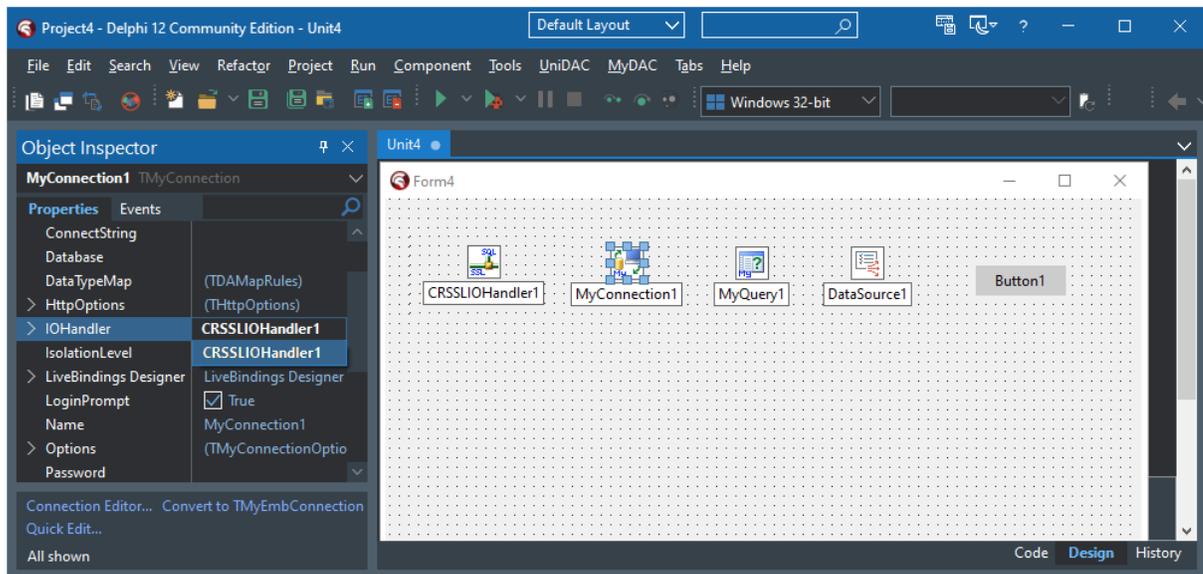
4. In the Object Inspector, expand **Options** and set the **Protocol** property to **mpSSL**.

5. Then expand **SSLOptions** and specify the server certificate in the **CACert** property, the
client certificate in the **Cert** property, and the private client key in the **Key** property.

6. Double-click **TMyConnection** and specify the server address, port, username, password,
and, optionally, database name. Click **Connect** to test connection to the MySQL server.
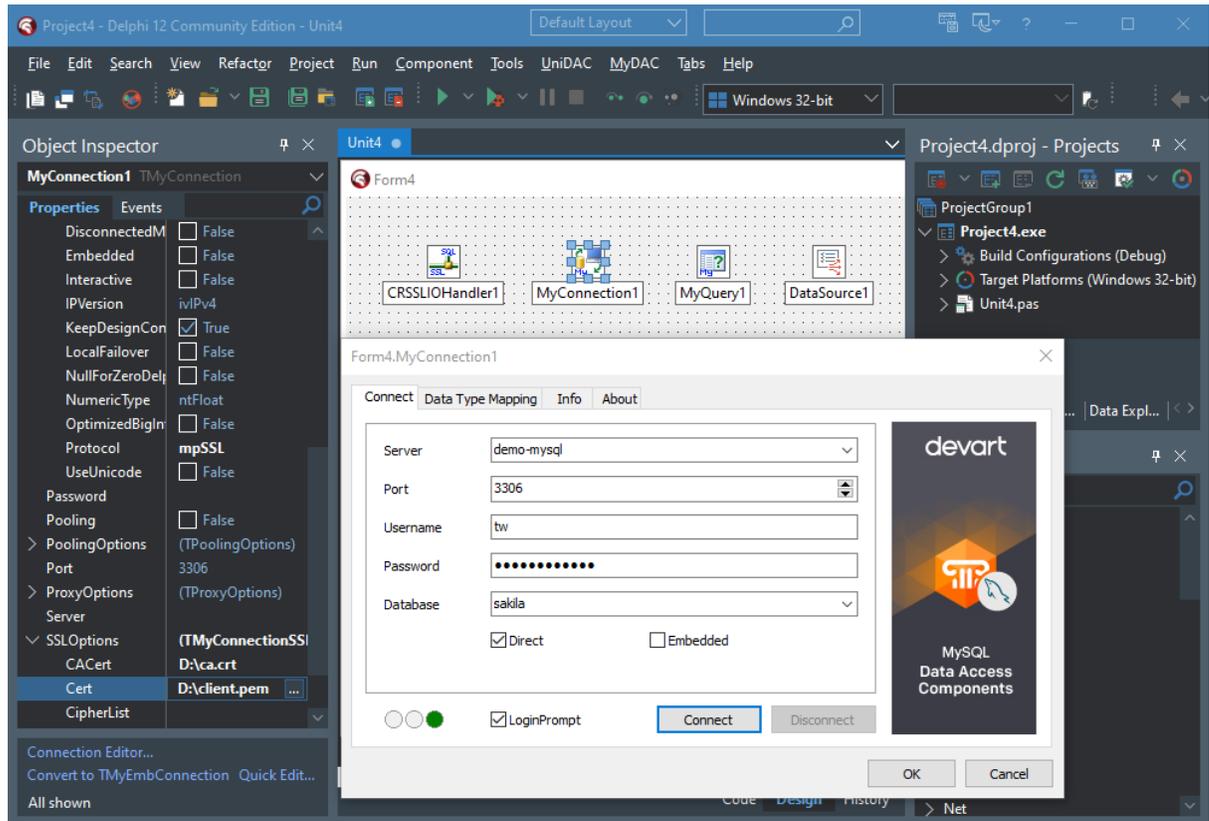
7. Select the **TDataSource** component and assign the **MyQuery1** object to the **DataSet** property.
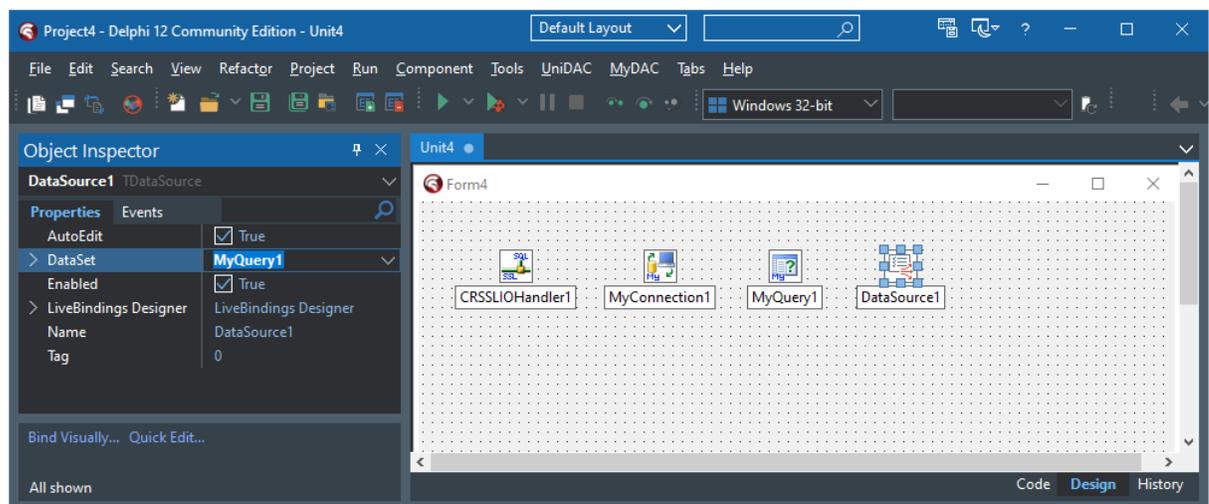
8. Assign the **DataSource1** object to the **DataSource** property in the **TDBGrid** component.
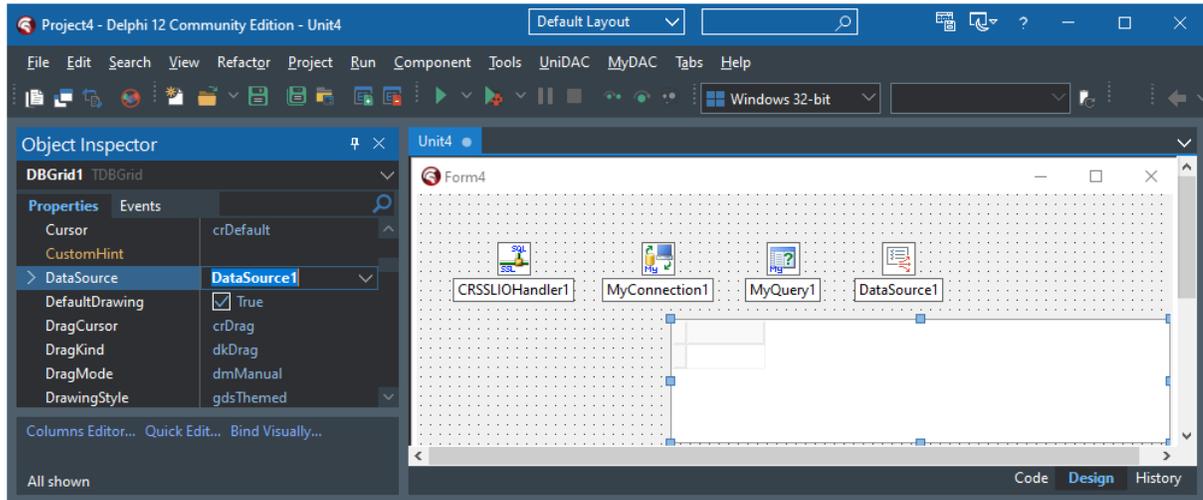
9. Double-click the **TMyQuery** component and add a SQL query that will be run against the MySQL database.
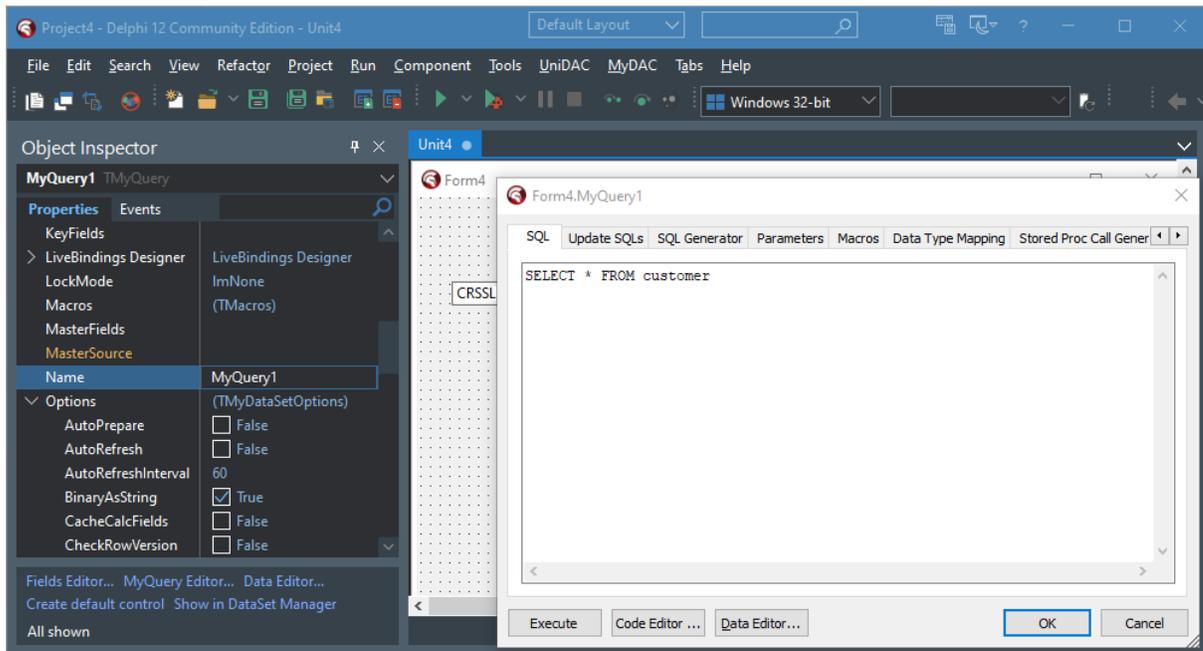
10. Select the **TButton** component and create the **OnClick** event. Add the code that will call the **Open** method in the **TMyQuery** component when you click the button.
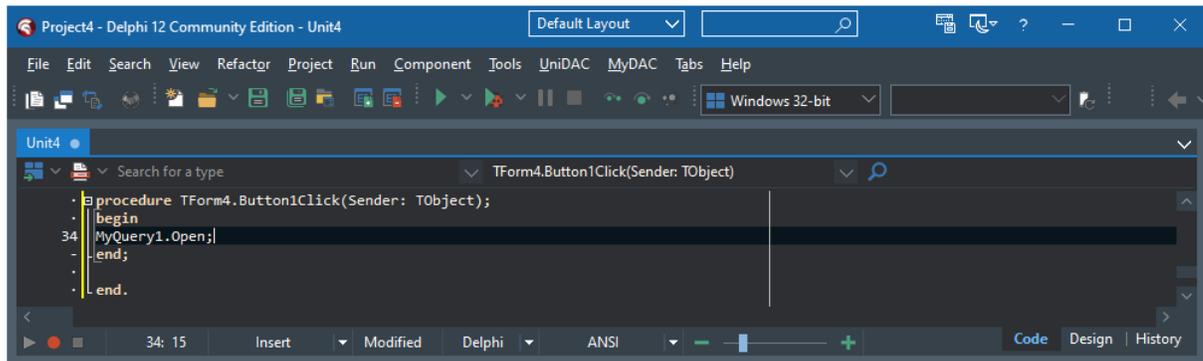
11. Press F9 to compile and run the application. Click the button on the form to execute the query and display data in the grid.

SSL connection to MySQL in Delphi using the OpenSSL library

Another way to embed SSL client functionality into your Delphi app, which uses MyDAC components to access MySQL, is by using the OpenSSL library that implements the SSL

protocol and enables servers to securely communicate with their clients.

Important: CA certificate authenticity is not verified when you use the OpenSSL library. MyDAC supports only the basic OpenSSL functionality, which does not include CA certificate verification. To configure advanced SSL features such as CA certificate verification, you need to use SecureBridge components instead of OpenSSL.

The following options must be set for SSL connection:

- SSLCACert - the pathname to the certificate authority file.

- SSLCert - the pathname to the certificate file.

- SSLKey - the pathname to the key file.

- SSLCipherList - a list of allowable ciphers to use for SSL encryption.

Note: The ssleay32.dll and libeay32.dll files are required to use the SSL protocol with the OpenSSL library.

Request Support   DAC Forum   Provide Feedback

4.7.2   Connecting via SSH

Connecting to MySQL Through SSH in Delphi

SSH is a protocol that allows users to securely log onto and interact with remote systems on the Internet by connecting a client program to an SSH server. SSH provides a mechanism for establishing a cryptographically secured connection between two endpoints, a client and a remote server, which authenticate each other and exchange messages. It employs different forms of symmetrical encryption, asymmetrical encryption, and hashing.

It is possible to use SSH to secure the network connection between a Delphi application and a MySQL server. You execute shell commands in the same fashion as if you were physically operating the remote machine.

Devart offers a solution called SecureBridge that allows you to create a Delphi SSH client and a server. You can embed the SSH client into your application and install the SSH server on a remote machine where the MySQL server resides. The SSH client connects to the SSH server, which sends all commands to the remote MySQL server. This tutorial demonstrates how to create a sample Delphi application that connects to MySQL using SSH as the

encryption method.

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

An SSH server is required to replicate the steps in this tutorial and encrypt the network connection between the client application and the MySQL server. You can build the SSH server demo project that is distributed with SecureBridge ('Documents\Devart\MyDAC for RAD Studio\Demos\TechnologySpecific\SecureBridge\Demo') and run the executable file.

After installing MyDAC and SecureBridge software on your system, install the TCRSSHIOHandler component in RAD Studio to bind MyDAC with SecureBridge. The installation instructions are provided in the Readme.html, which is located by default in "My Documents\Devart\MyDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge\".

Note: Starting with version MyDAC 12.3.0, the additional component TMySSHIOHandler is considered deprecated and is not recommended for use in new developments. Use the TCRSSHIOHandler component instead. The TMySSHIOHandler component will be provided for backward compatibility, and its source files will be located in the SecureBridge(Deprecated) folder.

Sample Delphi App that Connects to MySQL Using SSH

1. Run RAD Studio and select 'File -> New – > VCL Forms Application – Delphi'.

2. Place the following components on the form: **TCRSSHIOHandler**, **TMyConnection**, **TMyQuery**, **TScFileStorage**, **TScSSHClient**, **TDataSource**, **TDBGrid**, and **TButton**. The sample application will connect to the MySQL server via SSH, run a selection operation against the database, and display the obtained rows in the grid.
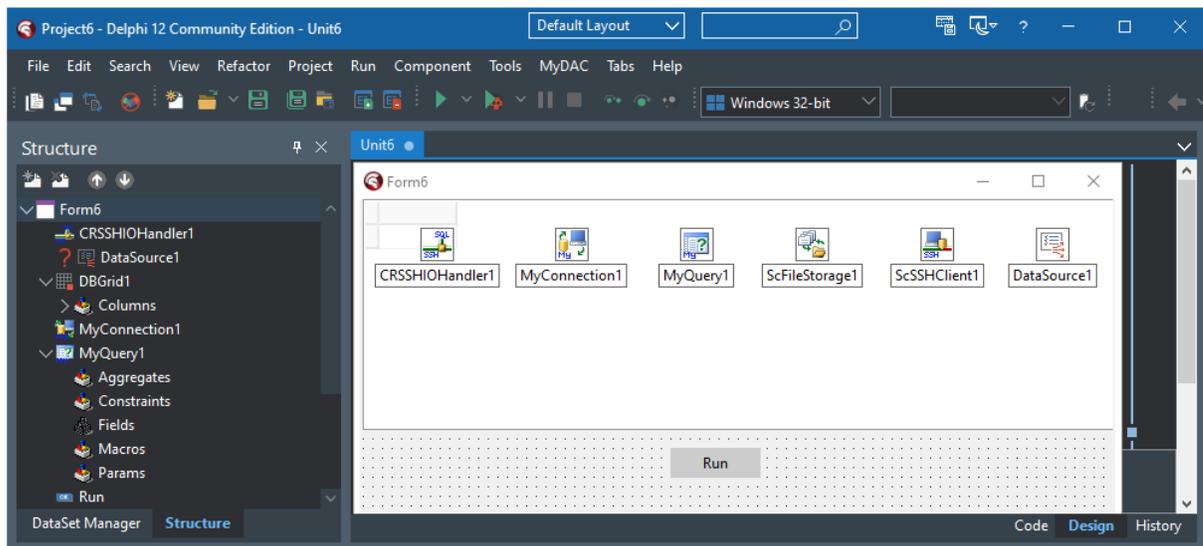
3. Select the **TDBGrid** and set the **DataSource** property to **DataSource1**.
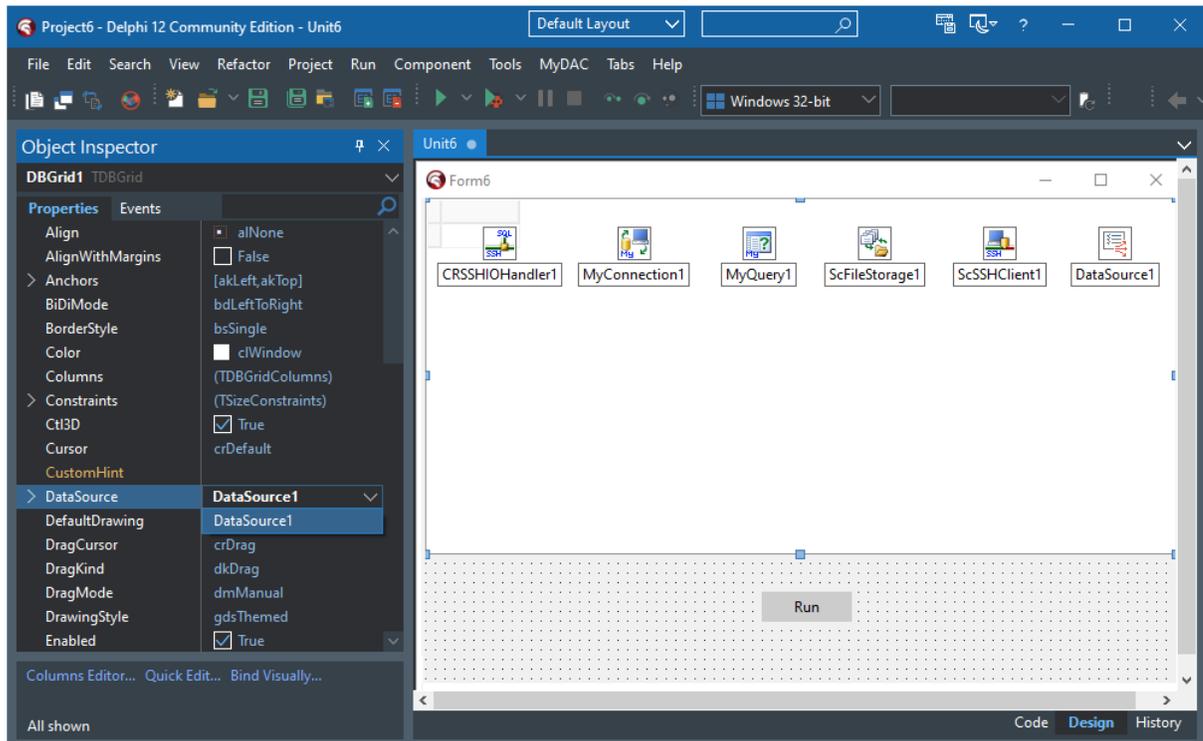
4. In the **TDataSource** component, assign **MyQuery1** to the **DataSet** property.
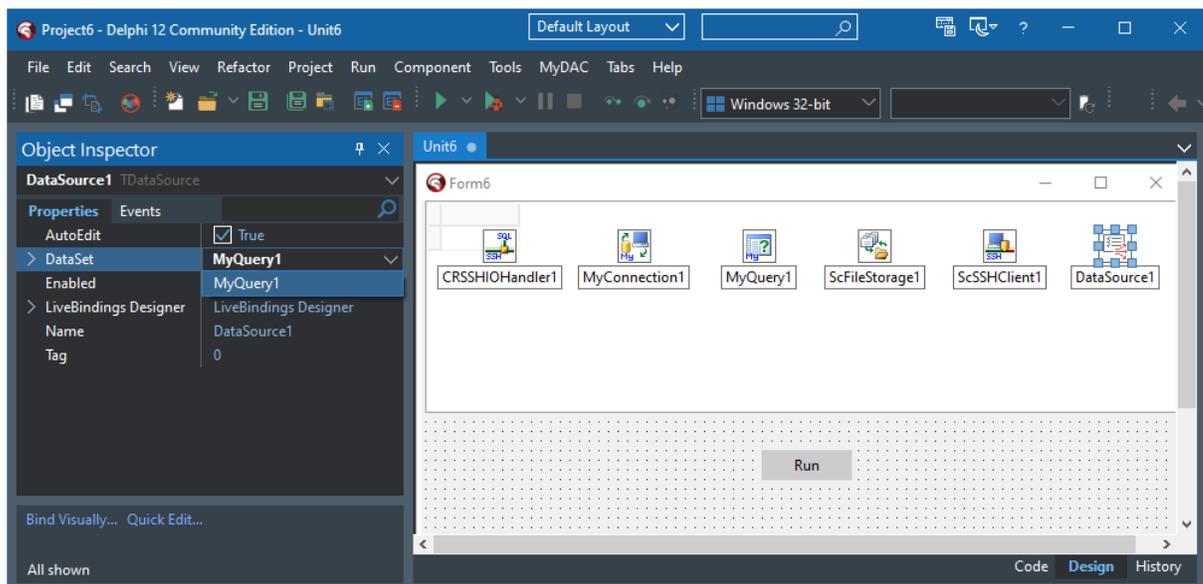
5. Select the **TMyQuery** and set the **Connection** property to **MyConnection1**. Double-click the component and enter a SQL statement to be executed against the MySQL database.
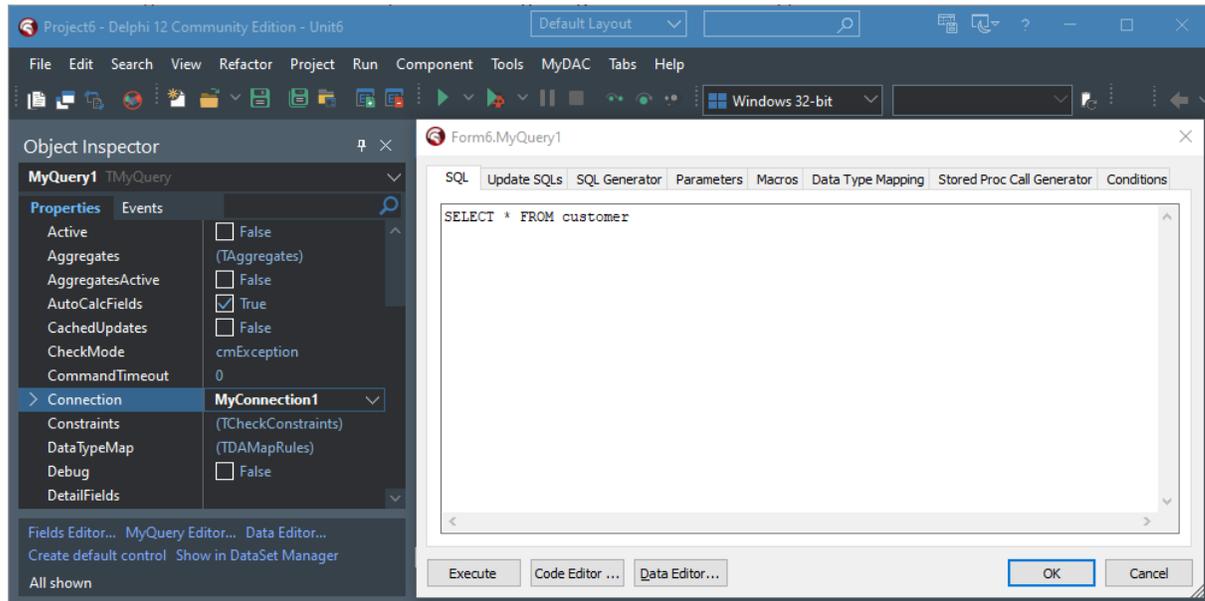
6. Double-click the **TButton** to switch to the unit view. Add the code to call the **Open** method on the **MyQuery1** object to activate the dataset when the button is clicked.
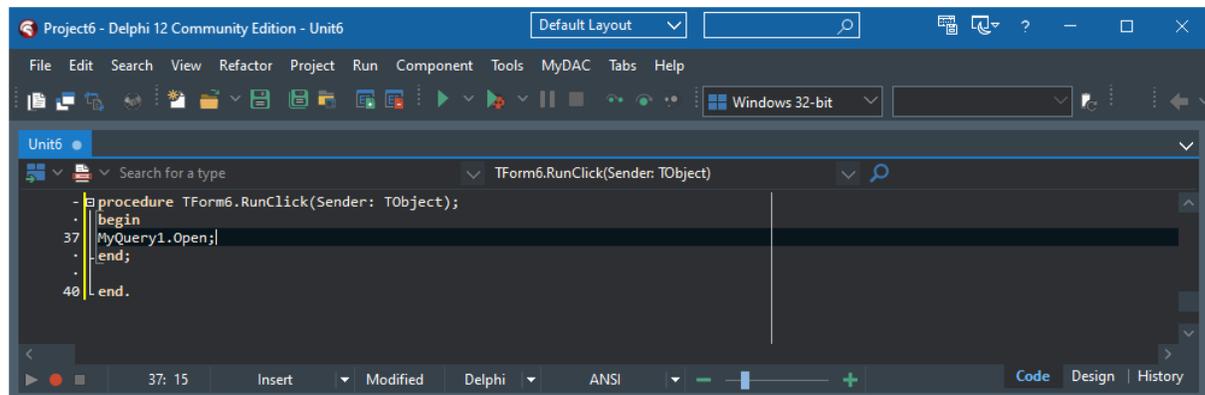
7. In the **TCRSSHIOHandler** component, assign **ScSSHClient1** to the **Client** property.
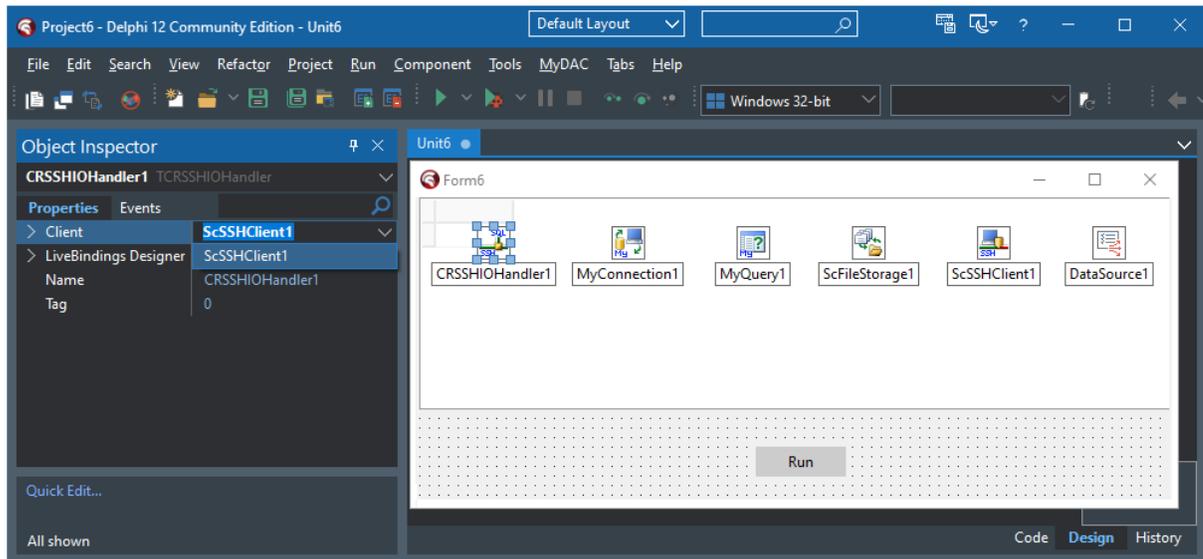
8. Select the **TScFileStorage** component and specify in the **Path** property the directory where keys are stored on your system. Double-click the component and generate a pair of keys for authenticating the server by the client.
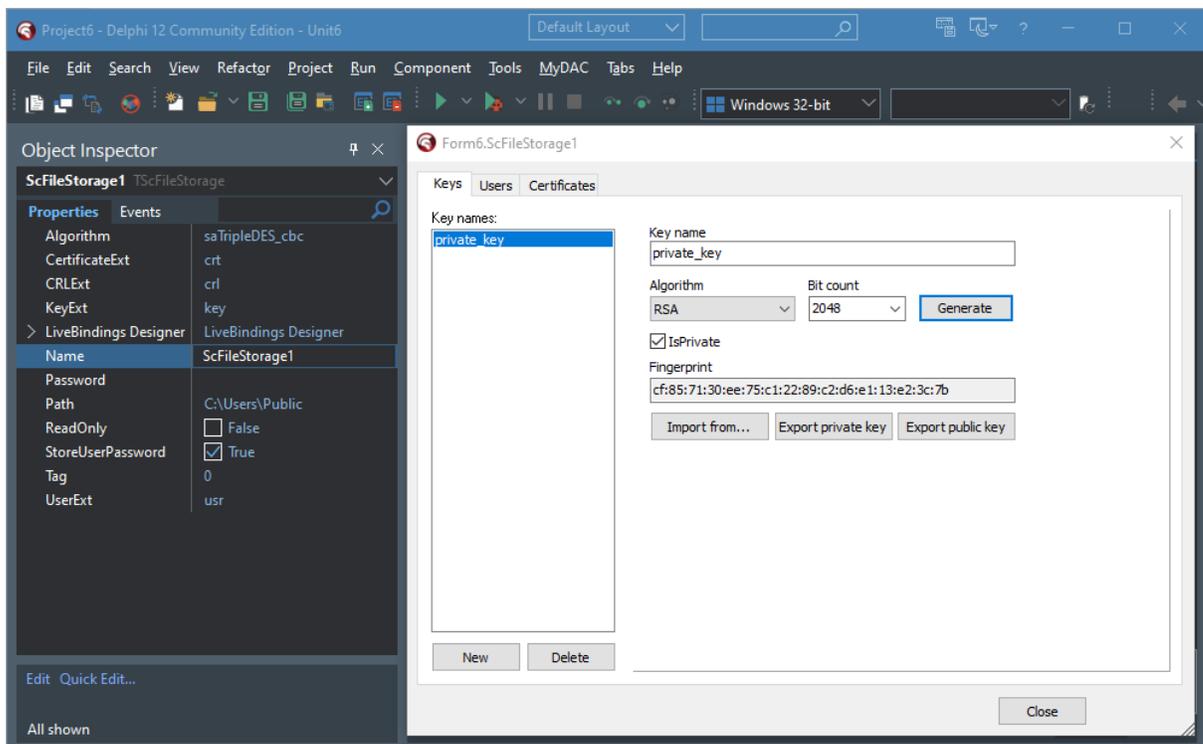
9. Set the **Authentication** property to **atPublicKey** in the **TScSSHClient** component. In

HostKeyName, specify the server public key. In **PrivateKeyName**, specify the client private key. The **Hostname** property holds the address of your server. Assign **ScFileStorage1** to the **KeyStorage** property. Enter your username on the server in the **User** property. Specify the SSH port in the **Port** property.
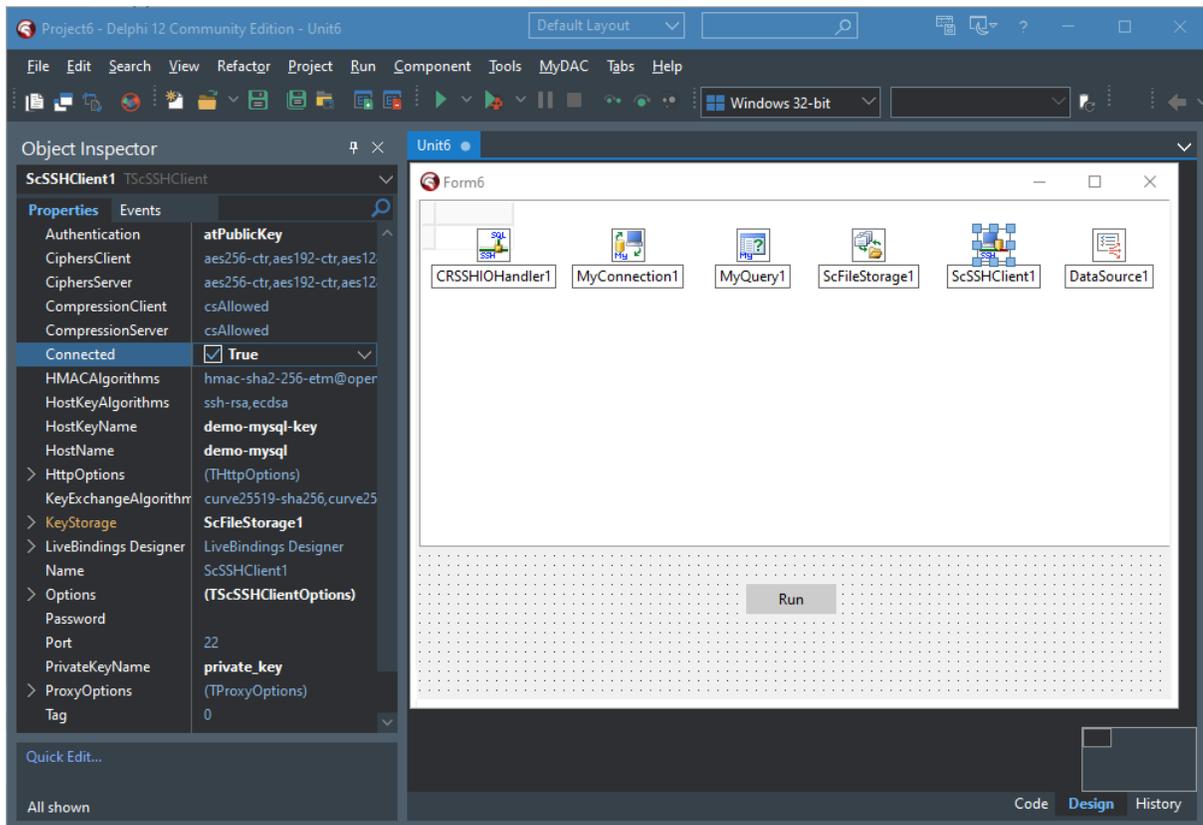
10. Double-click the **TMyConnection** component. Specify your server address, port, database name (optionally), and username and password for the MySQL user. Set the **IOHandler** property to **CRSSHIOHandler1**. Click Connect to check connection to the MySQL server.
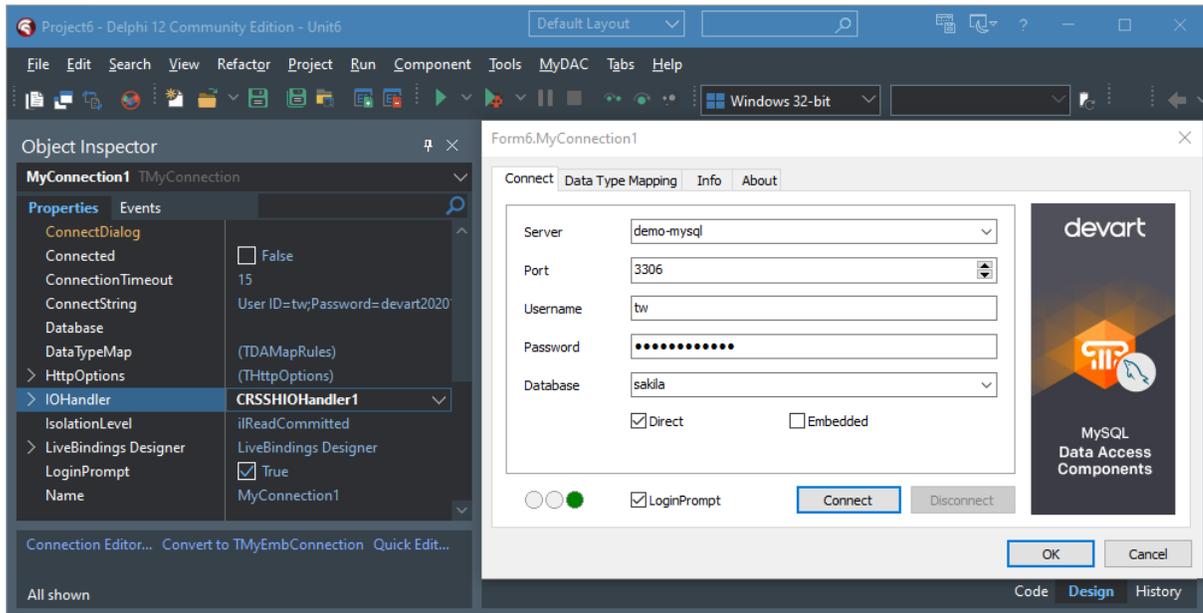
11. Press **F9** to compile and run the project. Click the button to run the query against the database and display the data in the form.

2. SSH Using OpenSSH or Any Other Third-Party SSH Tunnel

It is not obligatory to use SecureBridge TScSSHServer component as an SSH server - you can use any other server that implements the SSH protocol.

The following is the step-by-step sequence of actions for the easiest case of using OpenSSH for Windows. The detailed description of each command you can see in the documentation for OpenSSH.

1. Download OpenSSH for Windows from http://www.sourceforge.net/projects/sshwindows/

2. Install SSH server

- Choose a machine that will be used as SSH server. It does not have to be the same machine that is a MySQL server, but communication channel between SSH server and MySQL server must be protected

- Using Windows Control Panel create a user and set a password for him. For example, SSHUser with password SSHPass

- Install Open SSH. It is enough to install only Server components

- Open OpenSSH/bin folder

- Add SSHUser to the list of allowed users:

  mkpasswd -l -u SSHUser >> ..\etc\passwd

- Use mkgroup to create a group permissions file

  mkgroup -l >> ..\etc\group

- Run OpenSSH service

  net start opensshd

3. Install SSH client

- Choose a machine that will be used as SSH client. It does not have to be the same machine where client application (MySQL client) is running, but communication channel between SSH client and MySQL client must be protected

- Install Open SSH to SSH client. You may not install server components

- Run SSH client

  ssh.exe -L <SSH port>:<MySQL server>:<MySQL server port> <SSHUser>@<SSH

server>

<SSH port> - port number of SSH client that will be redirected to the corresponding port of MySQL server

<MySQL server> - name or IP address of the machine where MySQL server is installed

<MySQL server port> - number of MySQL server port. As usual, 3306.

<SSHUser> - user name created in p. 2

<SSH server> - name or IP address of the machine where SSH server is installed in p. 2

For example,

ssh.exe -L 3307:server:3306 SSHUser@192.168.0.116

At the first start you will be suggested to confirm a connection with the specified SSH server. Enter "yes" for confirmation.

On each start of SSH you must enter a password set in p. 2

4. Configure TMyConnection

MyConnection1.Server := <SSH client>;
MyConnection1.Port := <SSH port>;

If SSH client was installed at the same machine as MySQL client, you can assign 'localhost' to MyConnection1.Server.

Pay attention that in the specified sequence above check of SSHUser authentication is performed by Windows. About the methods of higher protection (key authentication etc) see documentation for OpenSSH.

To get more detailed information on using encrypted connections refer to MySQL Reference Manual.

Request Support   DAC Forum   Provide Feedback

4.7.3   Network Tunneling

Usually when a client needs to connect to server it is assumed that direct connection can be established. Nowadays though, due to security reasons or network topology, it is often necessary to use a proxy or bypass a firewall. This article describes different ways to connect to MySQL server with MyDAC.

- Direct connection

- Connection through HTTP tunnel

  - Connection Through proxy and HTTP tunnel

- Sample Delphi App That Connects to MySQL Using HTTP/HTTPS Tunneling

- Additional Information

Direct Connection

Direct connection to server means that server host is accessible from client without extra routing and forwarding. This is the simplest case. The only network setting you need is the host name and port number. This is also the fastest and most reliable way of communicating with server. Use it whenever possible.

The following code illustrates the simplicity:

```
MyConnection := TMyConnection.Create(self);
MyConnection.Server := 'localhost';
MyConnection.Port := 3306;
MyConnection.Username := 'root';
MyConnection.Password := 'root';
MyConnection.Connect;
```

Connection Through HTTP Tunnel

Sometimes client machines are shielded by a firewall that does not allow you to connect to server directly at the specified port. If the firewall allows HTTP connections, you can use MyDAC together with HTTP tunneling software to connect to MySQL server. MyDAC supports HTTP tunneling based on the PHP script.

An example of the web script tunneling usage can be the following: you have a remote website, and access to its database through the port of the database server is forbidden. Only access through HTTP port 80 is allowed, and you need to access the database from a remote computer, like when using usual direct connection.

You need to deploy the tunnel.php script, which is included into the provider package on the web server. It allows access to the database server to use HTTP tunneling. The script must be available through the HTTP protocol. You can verify if it is accessible with a web browser.

The script can be found in the HTTP subfolder of the installed provider folder, e. g. %Program Files%\Devart\MyDac for Delphi X\HTTP\tunnel.php. The only requirement to the server is PHP 5 support.

To connect to the database, you should set TMyConnection parameters for usual direct connection, which will be established from the web server side, the Options.Protocol property to mpHttp, and set the following parameters, specific for the HTTP tunneling:

Property	Mandatory	Meaning
HttpOptions.Url	Yes	URL of the tunneling PHP script. For example, if the script is in the server root, the URL can be the following: http://localhost/tunnel.php.
HttpOptions.Username, HttpOptions.Password	No	Set these properties if the access to the website folder with the script is available only for registered users authenticated with user name and password.

Connection Through Proxy and HTTP Tunnel

HTTP tunneling server is not directly accessible from client machine. For example, client address is 10.0.0.2, server address is 192.168.0.10, and the MySQL server listens on port 3307. The client and server reside in different networks, so the client can reach it only through proxy at address 10.0.0.1, which listens on port 808. In this case in addition to the TMyConnection.HttpOptions options you have to setup a HttpOptions.ProxyOptions object as follows:

```
MyConnection := TMyConnection.Create(self);
MyConnection.Server := '192.168.0.10';
MyConnection.Port := 3307;
MyConnection.Username := 'root';
MyConnection.Password := 'root';
MyConnection.Options.Protocol := mpHttp;
MyConnection.HttpOptions.Url := 'http://server/tunnel.php';
MyConnection.HttpOptions.ProxyOptions.Hostname := '10.0.0.1';
MyConnection.HttpOptions.ProxyOptions.Port := 808;
MyConnection.HttpOptions.ProxyOptions.Username := 'ProxyUser';
MyConnection.HttpOptions.ProxyOptions.Password := 'ProxyPassword';
MyConnection.Connect;
```

Note that setting parameters of MyConnection.HttpOptions.ProxyOptions automatically enables proxy server usage.

Sample Delphi App That Connects to MySQL Using

HTTP/HTTPS Tunneling

Note: Starting with version MyDAC 12.3.0, the additional component TMySSLIOHandler is considered deprecated and is not recommended for use in new developments. Use the TCRSSLIOHandler component instead. The TMySSLIOHandler component will be provided for backward compatibility, and its source files will be located in the SecureBridge(Deprecated) folder.

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

- Run RAD Studio and select 'File -> New – > VCL Forms Application – Delphi'.

- Place the following components on the form: **TMyConnection**, **TMyQuery**, **TDataSource**, **TDBGrid**, **TButton**, and **TCRSSLIOHandler**. The last component is required when you connect through HTTPS. Note that the **TCRSSLIOHandler** is distributed with SecureBridge and is required to bind MyDAC with SecureBridge. The installation instructions for the component are provided in the Readme.html, which is located by default in "My Documents\Devart\MyDAC for RAD Studio\Demos\TechnologySpecific \SecureBridge\DelphiXX".
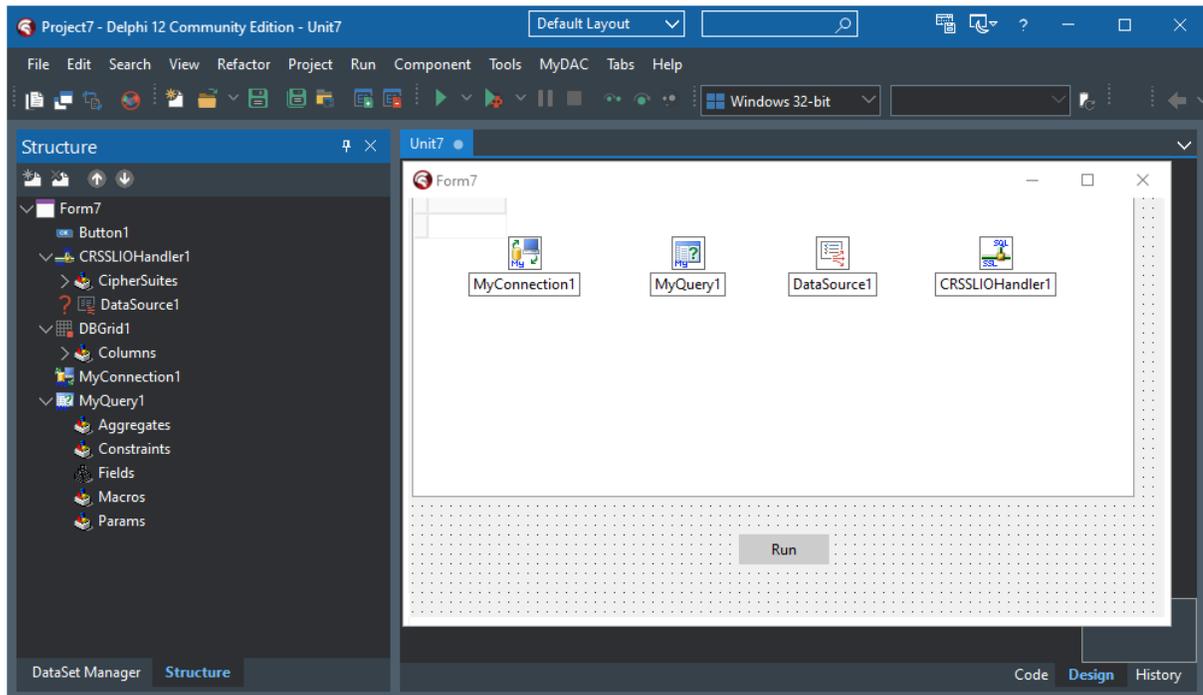
- Select **TDBGrid** and set the **DataSource** property to **DataSource1**.
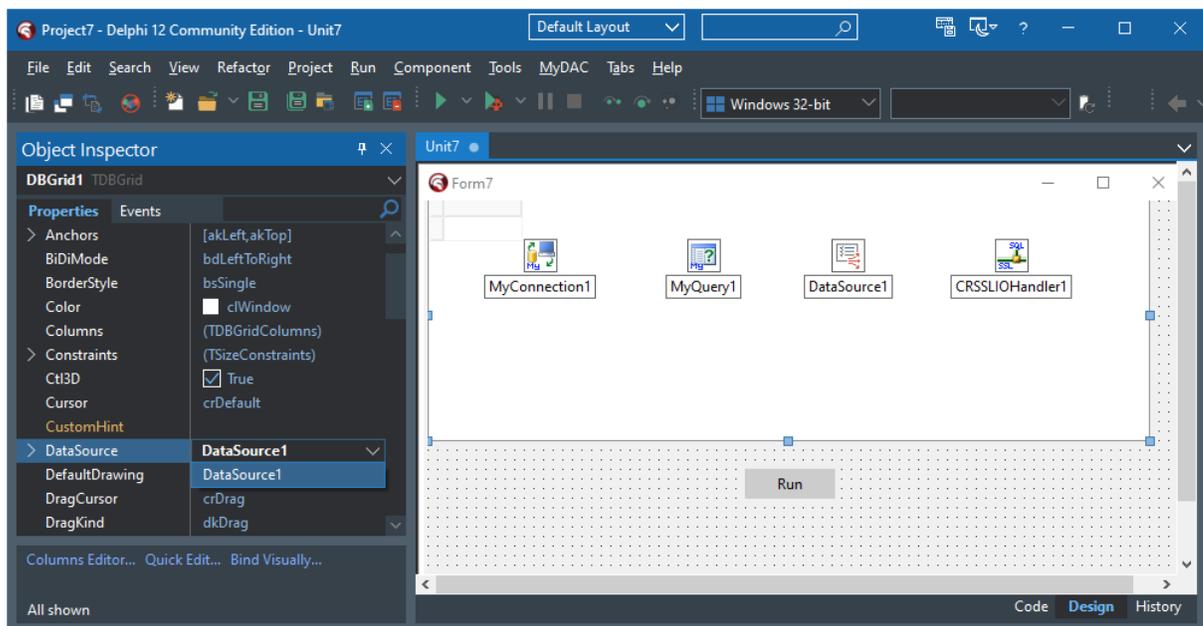
- In the **TDataSource** component, assign **TMyQuery1** to the **DataSet** property.
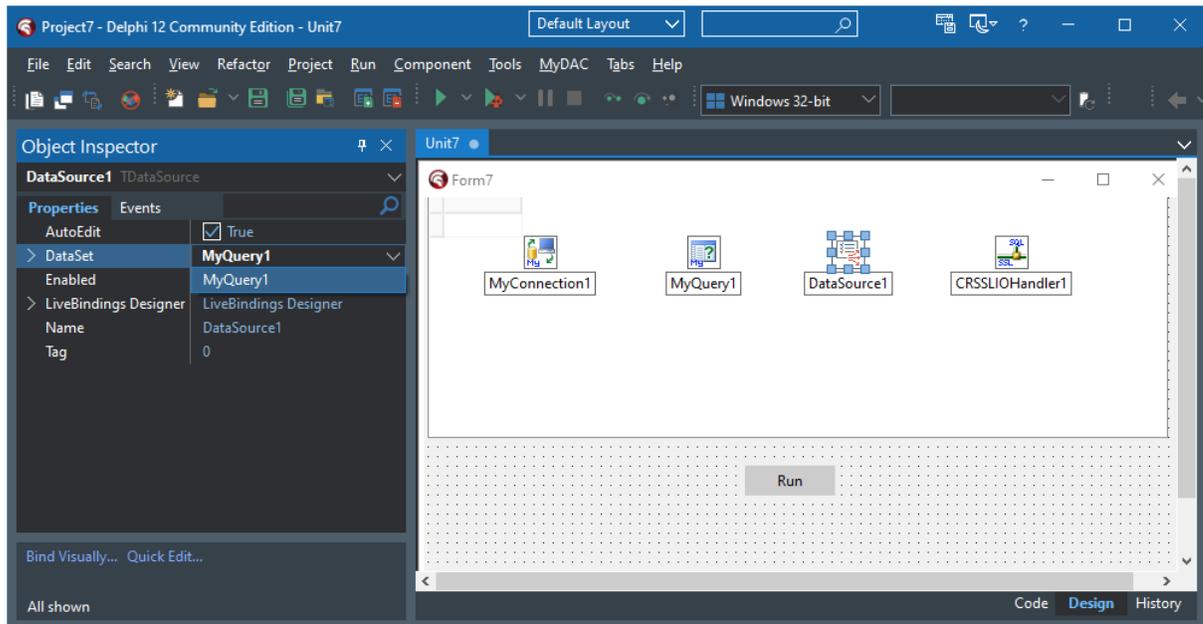
- Select **TMyQuery** and set the **Connection** property to **MyConnection1**. Double-click the component and enter an SQL statement to be executed against the MySQL database.
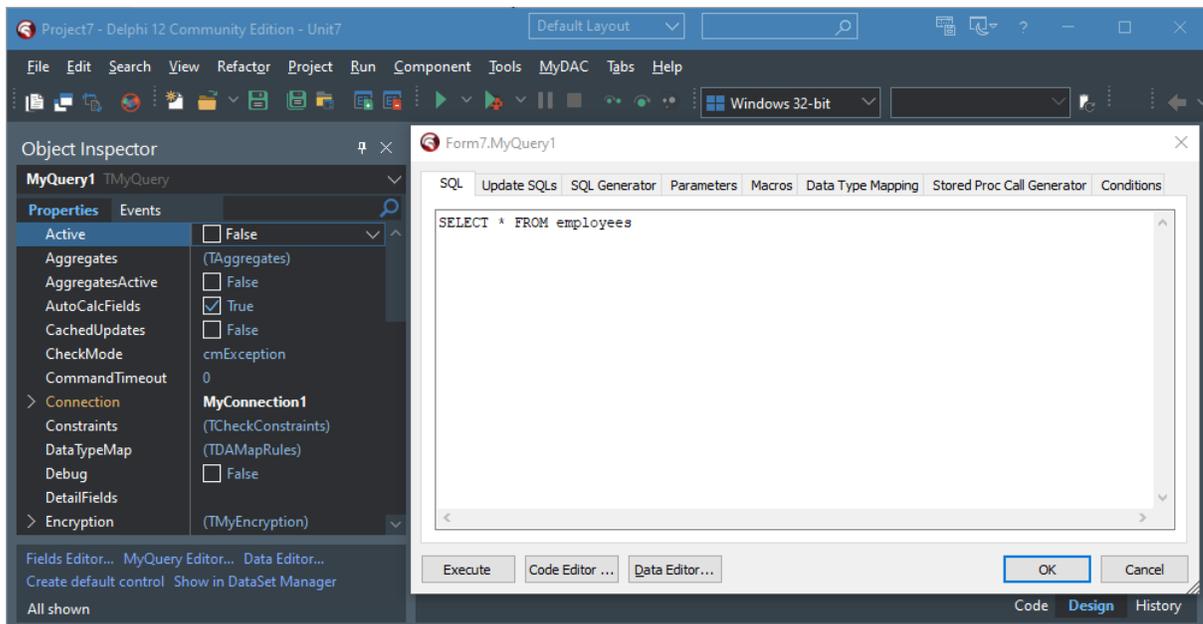
- Double-click **TButton** to switch to the unit view. Add code that invokes the **Open** method on the **MyQuery1** object to activate the dataset when the button is clicked.
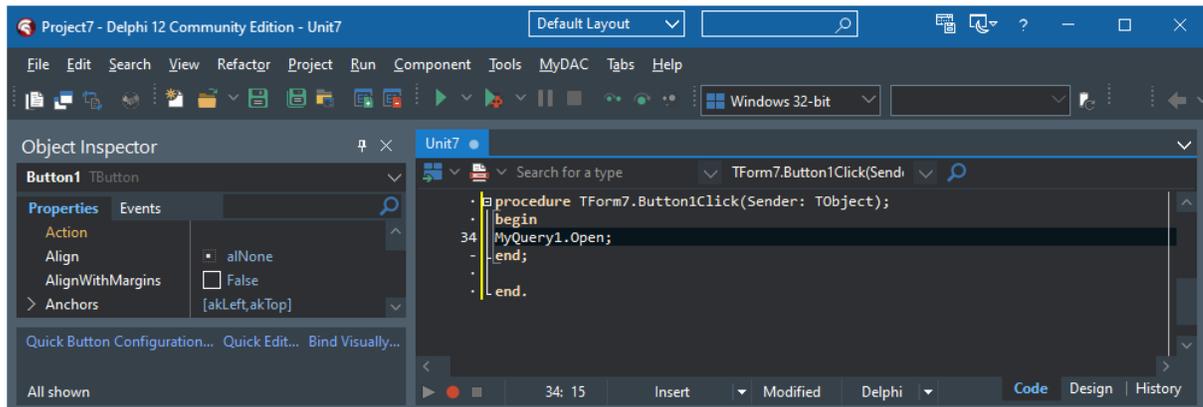
- Select the **TMyConnection** component. If you use the secure tunnel, set the **IOHandler** property to **CRSSLIOHandler1**. Expand **HttpOptions** and enter the URL of the tunnel.php script on your server.
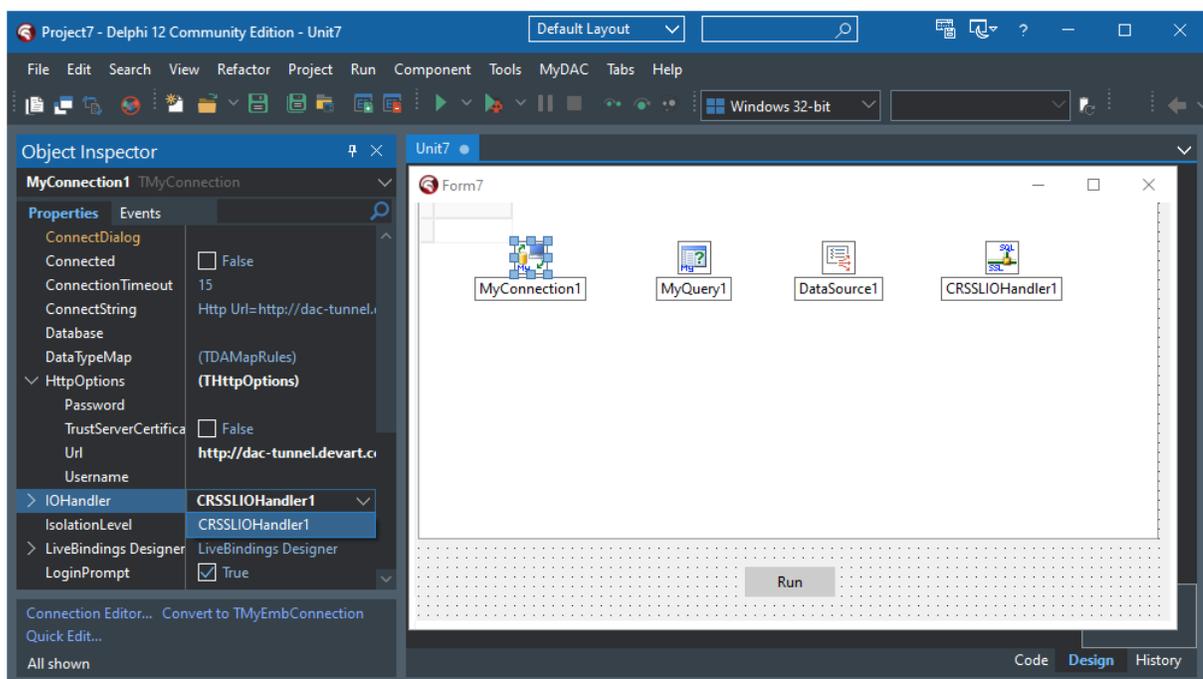
- In **TMyConnection**, expand **Options** and set the **Protocol** property to **mpHttp**.
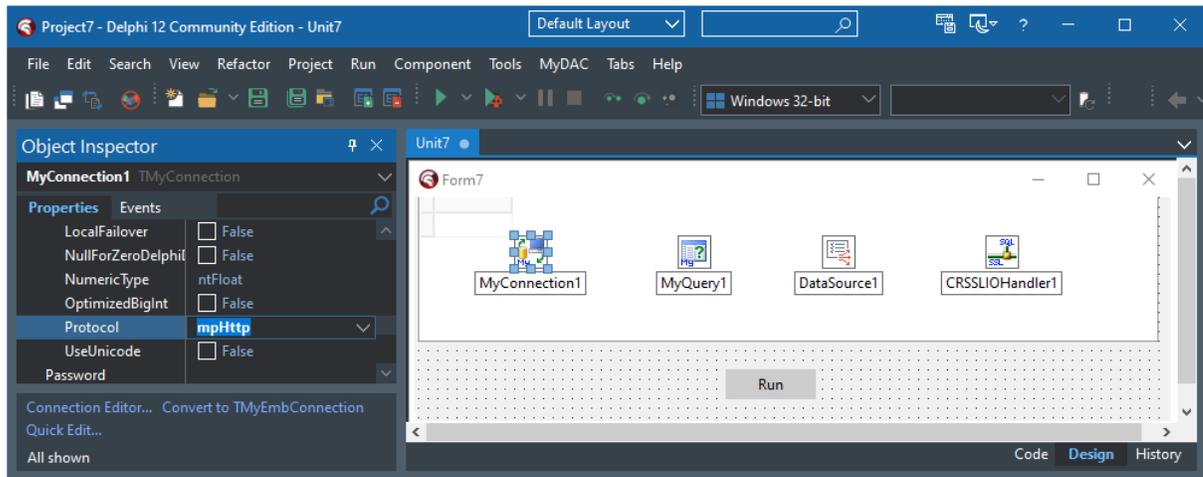
- Double-click the **TMyConnection** component. Specify your server address, port, database name (optionally), username and password for the MySQL user. Click **Connect** to test connection to the MySQL server.
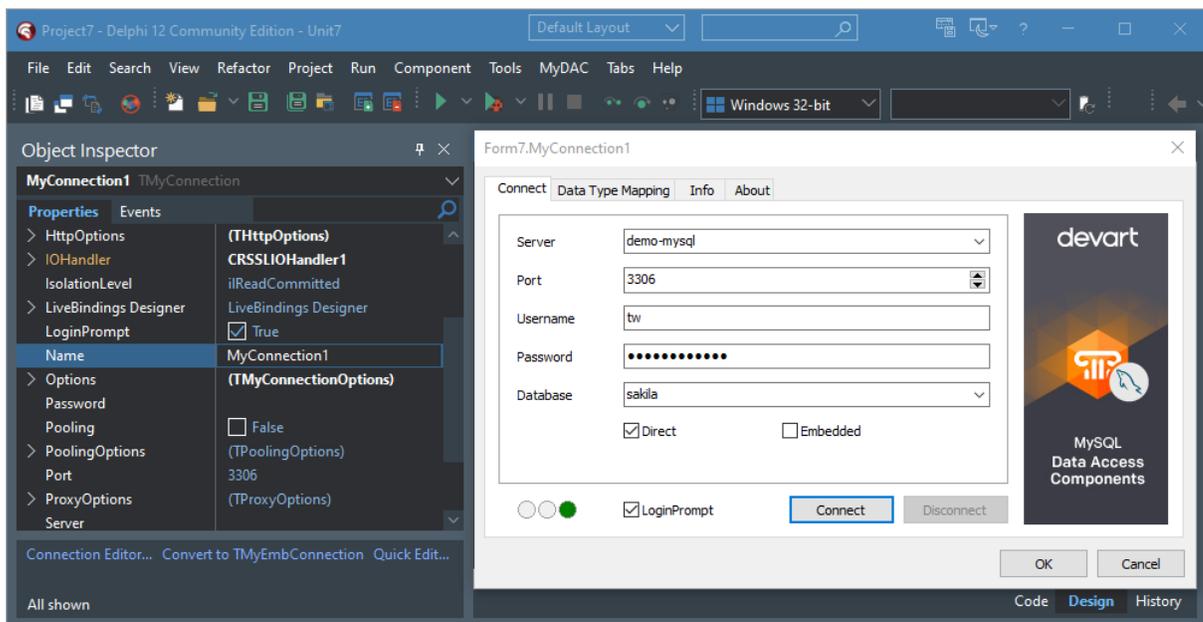
- Press **F9** to compile and run the project, and click the button to run the query against the database through HTTPS and display the data in the form.

Additional Information

There is one more way to tunnel network traffic. The Secure Shell forwarding, or SSH, can be used for forwarding data. However, main purpose of SSH is traffic encryption rather than avoiding firewalls or network configuration problems. The Connecting via SSH article describes how to use SSH protocol in MyDAC.

Keep in mind that traffic tunneling or encryption always increase CPU usage and network load. It is recommended that you use direct connection whenever possible.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

4.8   Batch Operations

Data amount processed by modern databases grows steadily. In this regard, there is an acute problem – database performance. Insert, Update and Delete operations have to be performed as fast as possible. Therefore Devart provides several solutions to speed up

processing of huge amounts of data. So, for example, insertion of a large portion of data to a DB is supported in the TMyLoader. Unfortunately, TMyLoader allows to insert data only – it can't be used for updating and deleting data.

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

Let's have a better look at capabilities of Batch operations with an example of the BATCH_TEST table containing attributes of the most popular data types.

Batch_Test table generating scripts

```
CREATE TABLE BATCH_TEST
(
  ID     INT,
  F_INTEGER INT,
  F_FLOAT FLOAT,
  F_STRING  VARCHAR(250),
  F_DATE  DATETIME,
  CONSTRAINT PK_BATCH_TEST PRIMARY KEY (ID)
)
```

Batch operations execution

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

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

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

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

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

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

Parameters

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

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

How to implement a Batch operation in the code?

Batch INSERT operation sample

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

```
var
  i: Integer;
begin
  // describe the SQL query
  MyQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F_F
  // define the parameter types passed to the query :
  MyQuery1.Params[0].DataType := ftInteger;
  MyQuery1.Params[1].DataType := ftInteger;
  MyQuery1.Params[2].DataType := ftFloat;
  MyQuery1.Params[3].DataType := ftString;
  MyQuery1.Params[4].DataType := ftDateTime;
  // specify the array dimension:
  MyQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to MyQuery1.Params.ValueCount - 1 do begin
    MyQuery1.Params[0][i].AsInteger := i + 1;
    MyQuery1.Params[1][i].AsInteger := i + 2000 + 1;
    MyQuery1.Params[2][i].AsFloat := (i + 1) / 12;
    MyQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
    MyQuery1.Params[4][i].AsDateTime := Now;
  end;
  // insert 1000 rows into the BATCH_TEST table
  MyQuery1.Execute(1000);
end;
```

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

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

All 1000 rows at a time:

```
MyQuery1.Execute(1000);
```

2×500 rows:

```
// insert first 500 rows
MyQuery1.Execute(500, 0);
// insert next 500 rows
MyQuery1.Execute(500, 500);
```

500 rows, then 300, and finally 200:

```
// insert 500 rows
MyQuery1.Execute(500, 0);
// insert next 300 rows starting from 500
MyQuery1.Execute(300, 500);
// insert next 200 rows starting from 800
MyQuery1.Execute(200, 800);
```

Batch UPDATE operation sample

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

```
var
  i: Integer;
begin
  // describe the SQL query
  MyQuery1.SQL.Text := 'UPDATE BATCH_TEST SET F_INTEGER=:F_INTEGER, F_FLOAT=
  // define parameter types passed to the query:
  MyQuery1.Params[0].DataType := ftInteger;
  MyQuery1.Params[1].DataType := ftFloat;
  MyQuery1.Params[2].DataType := ftString;
  MyQuery1.Params[3].DataType := ftDateTime;
  MyQuery1.Params[4].DataType := ftInteger;
  // specify the array dimension:
  MyQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to 1000 - 1 do begin
    MyQuery1.Params[0][i].AsInteger := i - 2000 + 1;
    MyQuery1.Params[1][i].AsFloat := (i + 1) / 100;
    MyQuery1.Params[2][i].AsString := 'New Values ' + IntToStr(i + 1);
    MyQuery1.Params[3][i].AsDateTime := Now;
    MyQuery1.Params[4][i].AsInteger := i + 1;
  end;
  // update 1000 rows in the BATCH_TEST table
  MyQuery1.Execute(1000);
end;
```

Batch DELETE operation sample

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

```
var
  i: Integer;
begin
  // describe the SQL query
  MyQuery1.SQL.Text := 'DELETE FROM BATCH_TEST WHERE ID=:ID';
  // define parameter types passed to the query:
  MyQuery1.Params[0].DataType := ftInteger;
  // specify the array dimension
  MyQuery1.Params.ValueCount := 1000;
  // populate the arrays with parameter values
  for i := 0 to 1000 - 1 do
    MyQuery1.Params[0][i].AsInteger := i + 1;
  // delete 1000 rows from the BATCH_TEST table
  MyQuery1.Execute(1000);
end;
```

Performance comparison

The example with BATCH_TEST table allows to analyze execution speed of normal operations with a database and Batch operations:

Operation Type	25 000 records	
	Standard Operation (sec.)	Batch Operation (sec.)
Insert	1138	11.02
Update	1637	26.72
Delete	1444	17.66
The less, the better.		

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

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

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

```
for i := 0 to 9999 do begin
  MyQuery1.Params.ParamByName('ID')[i].AsInteger := i + 1;
  MyQuery1.Params.ParamByName('F_INTEGER')[i].AsInteger := i + 2000 + 1;
  MyQuery1.Params.ParamByName('F_FLOAT')[i].AsFloat := (i + 1) / 12;
  MyQuery1.Params.ParamByName('F_STRING')[i].AsString := 'Values ' + IntToSt
  MyQuery1.Params.ParamByName('F_DATE')[i].AsDateTime := Now;
end;
```

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

Request Support   DAC Forum   Provide Feedback

4.9   Increasing Performance

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

Connect

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

Execute

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

To execute SQL statements the TMyCommand component is more preferable than TMyQuery. 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.

Fetch

In some situations you can increase performance a bit by using TDADataSetOptions.CompressBlobMode. You can also use

TMyConnection.Options.Compress. Setting TMyTable.Options.UseHandler can give an additional performance under high server load.

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.MyDac.TMyLoader component instead of Insert/Post methods, or execution of the
INSERT commands multiple times in a cycle. Sometimes usage of
TDevart.MyDac.TMyLoader improves performance several times.

    Request Support   DAC Forum   Provide Feedback

4.10 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 MyDAC 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;
```

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

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

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

Macros can be especially useful in scripts that perform similar operations on different objects.
You can use macros that will be replaced with an object name. It allows you to have the same

script text and to change only macro values. For example, the following is a script that creates a new user account and grants required privileges.

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

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

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

See Also

- TMacro

- TCustomDADataSet.MacroByName

- TCustomDADataSet.Macros

Request Support   DAC Forum   Provide Feedback

4.11   DataSet Manager

DataSet Manager window

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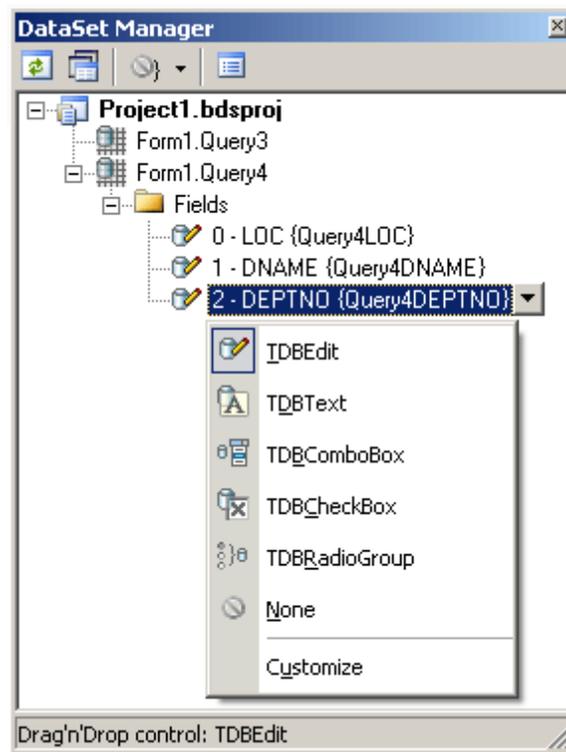
Using the DataSet Manager window, you can:

- Create forms that display data by dragging items from the DataSet Manager window onto forms.

- Customize the list of controls available for each data type in the DataSet Manager window.

- Choose which control should be created when dragging an item onto a form in your Windows application.

- Create and delete TField objects in the DataSets of your project.

Opening the DataSet Manager window

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

Observing project DataSets in the DataSet Manager Window

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

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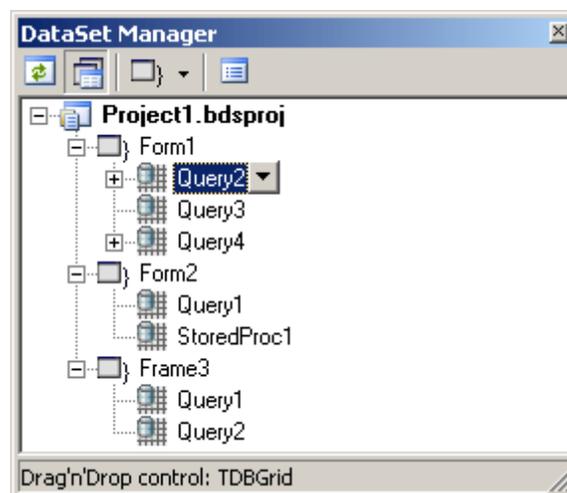
Creating Data-bound Controls

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

To use grid layout drag the dataset node on the form. By default TDataSource and TDBGrid components are created. You can choose the control to be created prior to dragging by selecting an item in the DataSet Manager window and choosing the control from the item's

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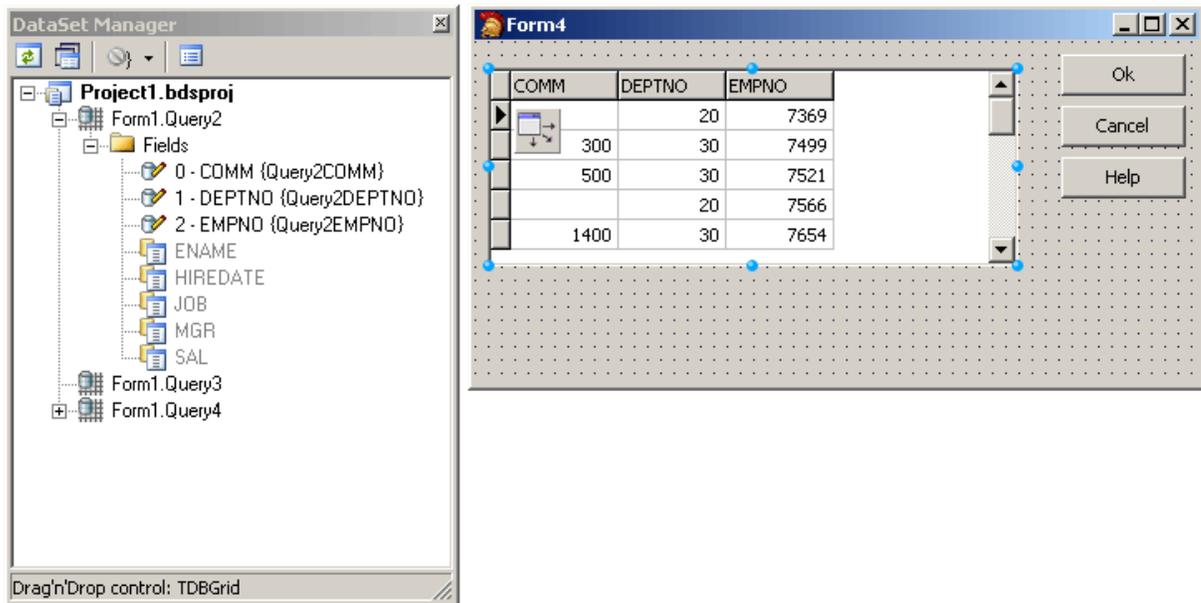
To use Details layout choose Details from the DataSet node drop-down control list in the DataSet Manager window. Then select required controls in the drop-down control list for each DataSet field. DataSet fields must be created. After setting required options you can drag the DataSet to the form from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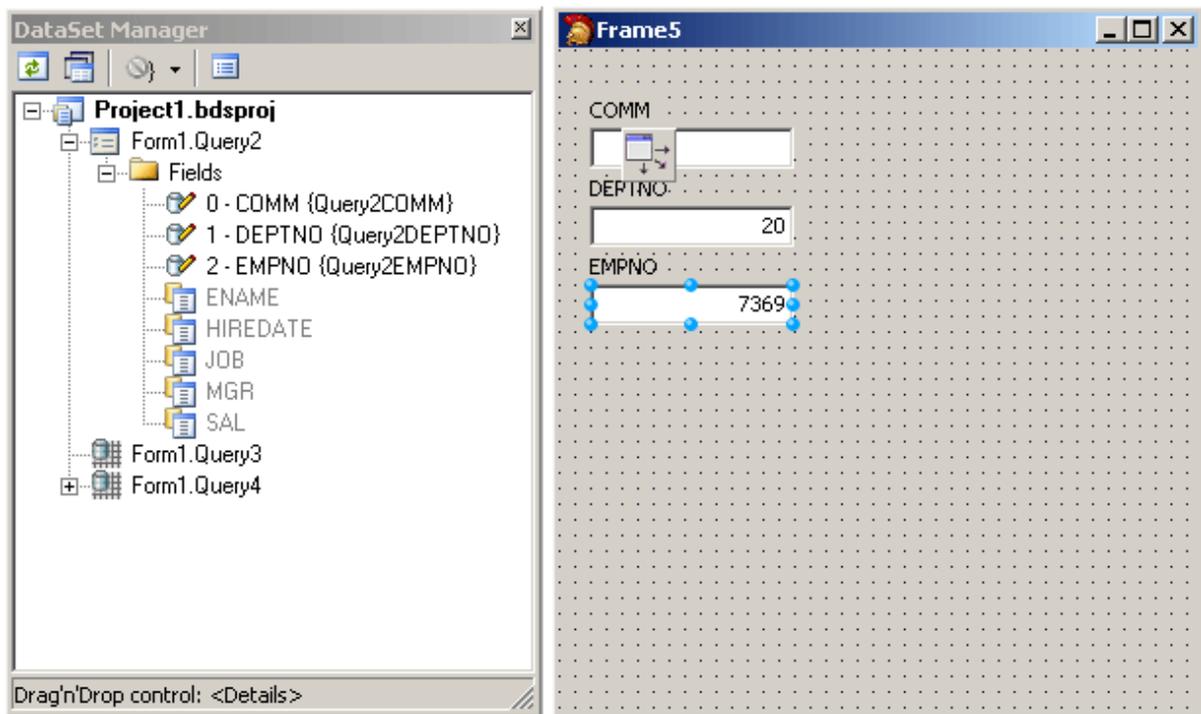
Adding custom controls to the DataSet Manager window

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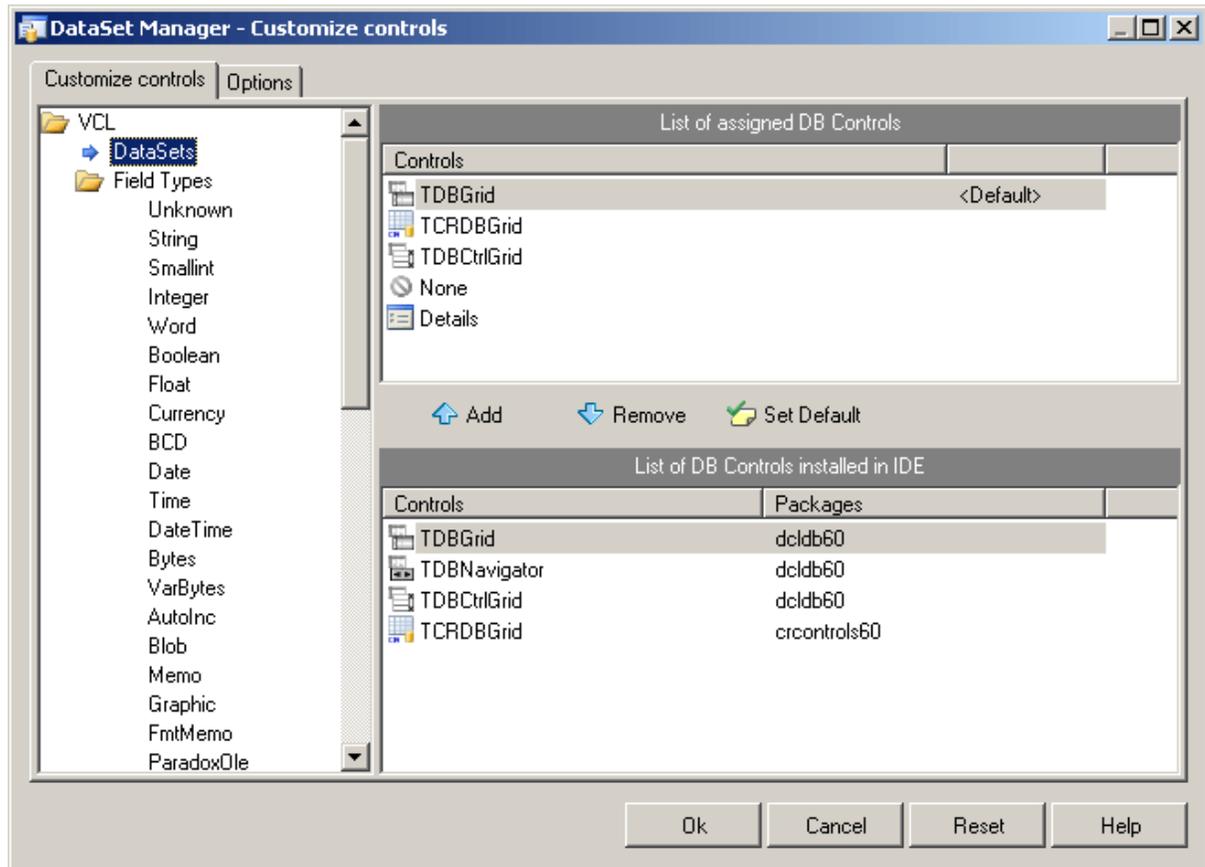
The default configuration can easily be restored by pressing Reset button in the *DataSet Manager - Options* dialog.

Working with TField objects

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

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

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

OK button.

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

DataSet Manager allows you to change view of the fields displayed in the main window. Open the *Customize controls* dialog, and jump to the Options page.
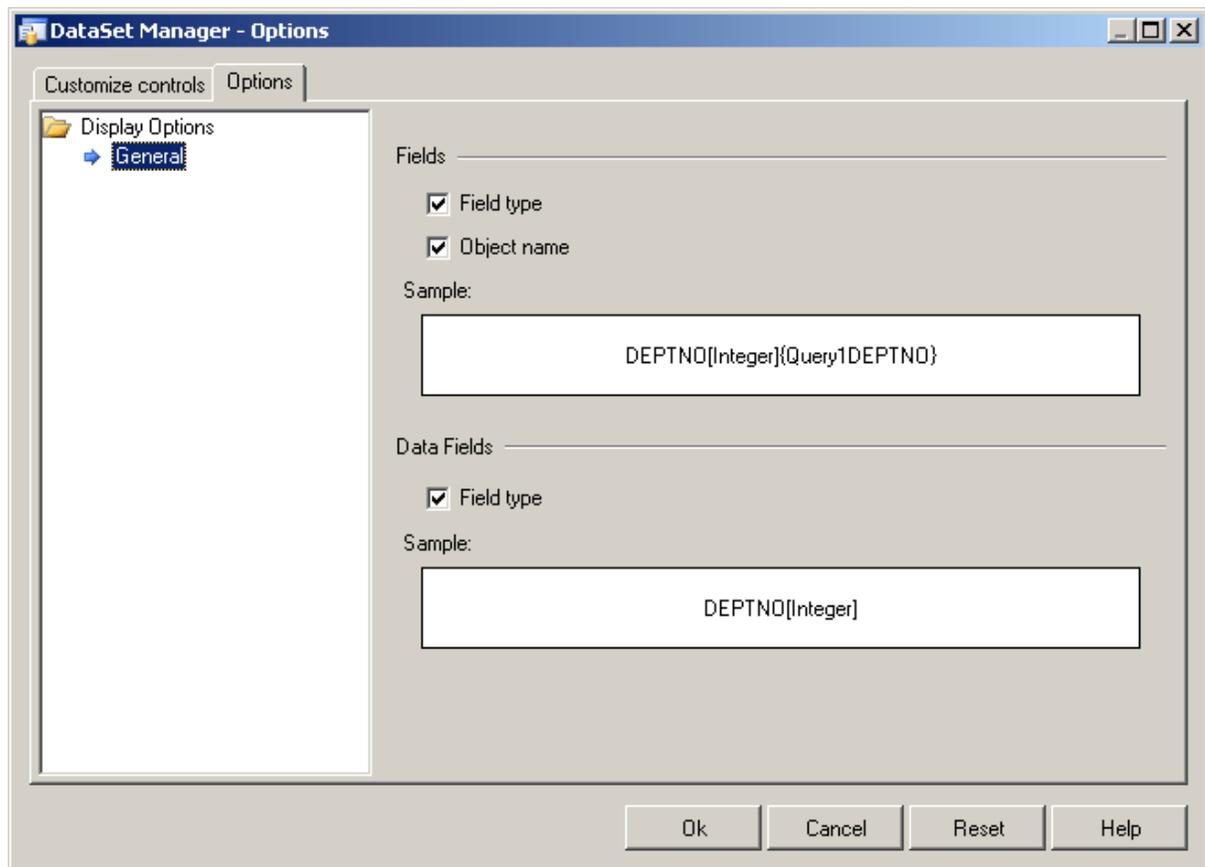
You can chose what information will be added to names of the Field and Data Field objects in the main window of DataSet Manager. Below you can see the example.

Request Support   DAC Forum   Provide Feedback

4.12   Embedded Server

Since version 4.0 MySQL server supports Embedded server. Embedded server is an easy to install server used by applications that do not require multi-user work with MySQL server. For example, Embedded server can be used for money access machines, automatic cash

desks, different electronic facilities and so on. Please refer to MySQL Reference Manual for more details on features and using of Embedded server. Also you can find some information about licensing Embedded server in MySQL Reference Manual. Please refer to Embedded Demo for a sample.

Which version of Embedded server to use

MySQL Embedded Server 4.0 should be recompiled to be used in your application.

MySQL Embedded Sever 5.0 is not included into the binary installation pack. Below is a quotation from the MySQL Reference Manual:

"The Embedded MySQL server library is NOT part of MySQL 5.0. It is part of previous editions and will be included in future versions, starting with MySQL 5.1."

That is why we have not tested MyDAC with the MySQL Embedded Sever 5.0.

So, we recommend using MySQL Embedded Server 4.1. As MySQL Embedded Server has some problems working with the InnoDB storage, we recommend disabling this storage engine. You can do this by checking the "Disable InnoDB storage engine" option in the TMyEmbConnection editor on the Params tab. Another way is to add the --skip-innodb parameter to the TMyEmbConnection component manually.

Installation
- Windows

Copy libmysqld.dll file to the folder available for executable file of the application. Please see a detailed description of accessible paths at LoadLibrary description
A typical structure of folders for an application using Embedded Server:
Project.exe - executable file of your application
libmysqld.dll - MySQL Embedded server library
share/english/errmsg.sys - file with MySQL Embedded server messages
data/ - data directory (DataDir). See a structure of this folder in MySQL Reference Manual
data/mysql/ - directory with service data of MySQL (user access rights, and so on) data/ DataBase/ - directory with user data. See TCustomMyConnection.DataBase

- Linux

  - Copy libmysqld.so.14.0.0 file to /usr/lib folder

- At /usr/lib folder execute the following commands to create links:

ln libmysqld.so.14.0.0 libmysqld.so

ln libmysqld.so.14.0.0 libmysqld.so.14

- Copy files needed for working of Embedded Server. As a rule, it is errormessage file, for example share/english/errmsg.sys.

- Create a folder for data

- If it is necessary, copy files with data to the data folder

Settings

On the start (first opening a connection), MySQL Embedded Server searches for the setting values in the next order:

- TMyEmbConnection.Params

- [<Application exe-file name (with extension)>] section of configuration file (my.ini or my.cnf) - settings specific to particular application.

- [Embedded] section of configuration file - settings specific to Embedded server

- [Server] section of configuration file - common settings for MySQL server and Embedded server.

Usually to set-up Embedded Server it is enough to set basedir and datadir.But sometimes some additional settings are required, for example to disable usingInnoDB engine (--skip-innodb). The detailed list of settings you can find at MySQL Reference Manual.

Pay attention that all paths must be set through "/" but not "\".

Note, parameters names are case-sensitive.

If datadir is located in the read-only storage, then you need to set OnLog andOnLogError event handlers to prevent server from attempts to create log-files in datadir.

Limitations

Simultaneous access to the same data from several instances of MySQL server (for example, to MySQL server and Embedded server) can be a reason of data loss.

See Also

- Embedded Demo

- [TMyEmbConnection](#)

- [TMyEmbConnection.Params](#)

- [TMyConnection.Options.Embedded](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

4.13 National Characters

When data is transferred data between the client and server, the server needs to know the encoding used by the client. You can set the encoding by configuring the MySQL server settings (see the MySQL Reference Manual for details), or by using the [TMyConnection.Options.Charset](#) or [TMyConnection.Options.UseUnicode](#) properties on the client. The former way is less suitable as it requires access to the server settings, which is not always possible. The latter way is more convenient, but it may cause some delays.

The `Charset` and `UseUnicode` properties are mutually exclusive, thus when you set `UseUnicode` to `True`, the value of `Charset` is ignored. By default, `Charset = ''` and `UseUnicode = False`. The server makes conversions according to its settings.

If a character set is assigned to the `Charset` property, the `SET NAMES <Charset>` statement is passed to the server when establishing a connection, to notify it about the character used by the client. To get a list of available charsets, execute the `SHOW CHARSET` statement. Note that when you assign `utf8` to this property, the values of all string fields are converted to UTF-8, which may prevent you from using data-aware components.

If you to need to receive string data in the Unicode format on the client to work with almost any language, or you connect with a username or password that contains characters like £, set the `UseUnicode` property to `True`. All `TStringField` values will be converted to `TWideStringField`. This behaviour is useful when, for example, you create a database of books in the library, and you need to store the original title of the book alongside the translated title. The downside is that all string data is converted on the client, which may cause some delays.

See Also

- TCustomMyConnection.Options

- TMyConnection.Options

Request Support   DAC Forum   Provide Feedback

4.14   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 , Server, T:Devart.Odac.TOraSession, Password, Database, IsolationLevel, Port, IOHandler, ConnectionTimeout, Compress, Direct, Embedded, Protocol, Charset, UseUnicode, NumericType . 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 TMyConnectionPoolManager. 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

Request Support   DAC Forum   Provide Feedback

4.15 DBMonitor

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

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 TMySQLMonitor component onto the form;

- turn moDBMonitor option on;

- set to True the Debug property for components you want to trace;

- start DBMonitor before running your program.

Request Support   DAC Forum   Provide Feedback

4.16   Writing GUI Applications with MyDAC

MyDAC 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 MyDacVcl unit in your application. This feature is needed for writing console applications.

Delphi and C++Builder

By default MyDAC does not require Forms, Controls and other GUI related units. Only TMyConnectDialog component require the Forms unit.

Request Support   DAC Forum   Provide Feedback

4.17   Syntax Highlighting in Component Editors

MyDAC supports syntax highlighting in component editors at design time. This topic explains how to enable or disable this feature in MyDAC component editors.

- Enable syntax highlighting
  - Delphi
  - Lazarus
- Disable syntax highlighting in Delphi

Enable syntax highlighting

Delphi

To enable syntax highlighting for Delphi, install the freeware SynEdit component set. You can download SynEdit from the official website or install it using the GetIt Package Manager. Once installed, syntax highlighting will be enabled automatically in all component editors.
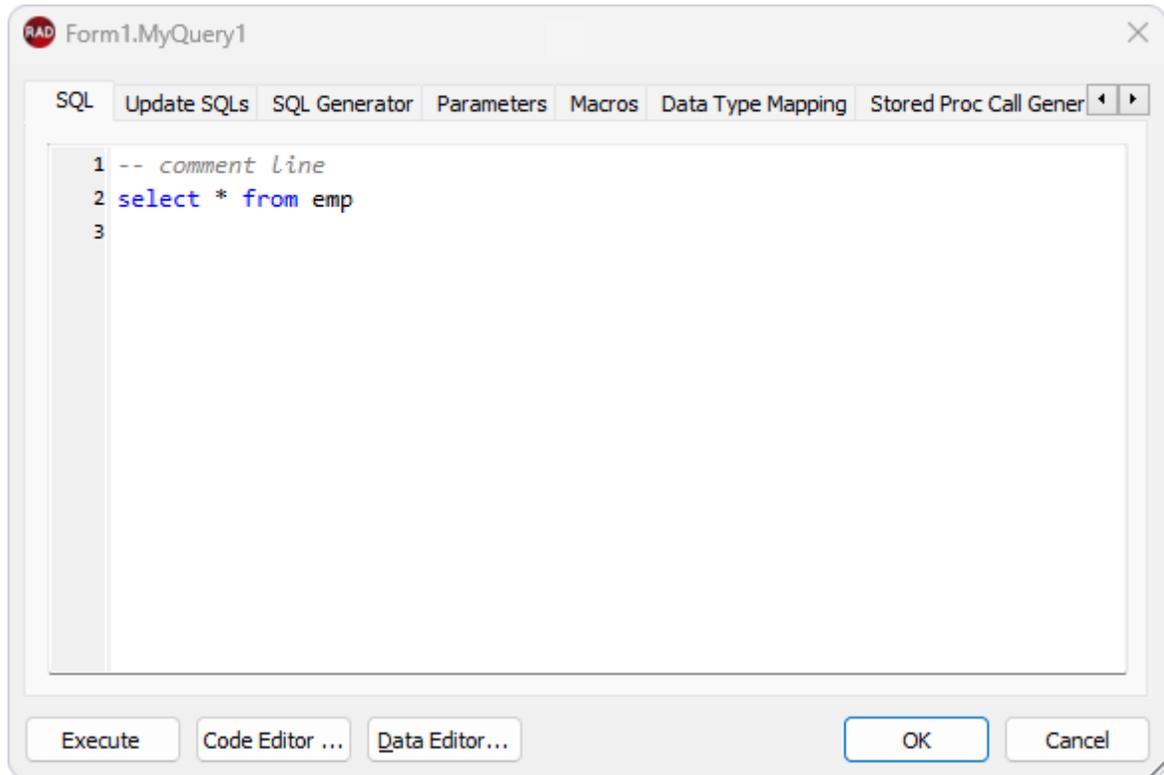
Lazarus

In Lazarus, SynEdit is a system package integrated into the IDE. This means that syntax highlighting for MyDAC component editors is enabled automatically and requires no additional setup.

Disable syntax highlighting in Delphi

If you have SynEdit installed but prefer not to use syntax highlighting in MyDAC component editors, you can disable it in the following way:

1. Open the Registry Editor.

2. Navigate to the **HKEY_CURRENT_USER\Software\Devart\MyDAC\Editors** registry key.

3. Create a **DWORD** value named **UseSynEdit** and set its value to **0**.
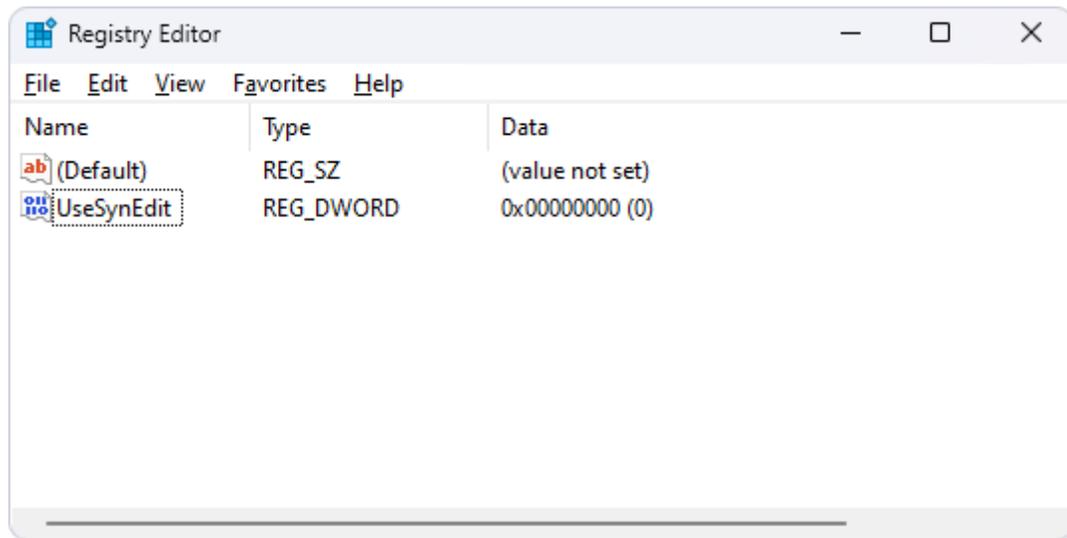
Syntax highlighting is now disabled for MyDAC. To enable syntax highlighting again, delete the **UseSynEdit** value or set it to 1.

© 1997-2025
Devart. All Rights
Reserved.

    Request Support   DAC Forum   Provide Feedback

4.18   Compatibility with Previous Versions

We always try to keep MyDAC compatible with previous versions, but sometimes we have to change the behaviour of MyDAC in order to enhance its functionality, or avoid bugs. This topic describes such changes, and how to revert the old MyDAC behaviour. We strongly recommend not to turn on the old behaviour of MyDAC. Use options described below only if changes applied to MyDAC 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 MyDAC 4.00.2.8. To restore old behavior, set the BaseSQLOldBehavior variable to True.

DBAccess.SQLGeneratorCompatibility:

If the manually assigned RefreshSQL property contains only "WHERE" clause, MyDAC 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 MyDAC 5.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

MemDS.SendDataSetChangeEventAfterOpen:

Starting with MyDAC 5.20.0.11, the DataSetChange event is sent after the dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

MemDS.DoNotRaiseExcetionOnUaFail:

Starting with MyDAC 5.20.0.12, if the OnUpdateRecord event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.

MyClasses.__Strings65535ToMemo:

Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields. It causes wrong description of these fields by MyDAC and truncating their data. To avoid these problems, MyDAC tries to restore the correct data type. This behaviour was introduced in MyDAC 5.10.0.9. To disable this behaviour, set the __Strings65535ToMemo variable to False.

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.

DBAccess.RefreshParamsOnInsert:

Starting with MyDAC 5.50.0.36, when master/detail relationship is used on inserting a new record into master table parameters in detail table are not updated. To restore the old behavior, set the RefreshParamsOnInsert variable to True.

Request Support   DAC Forum   Provide Feedback

4.19   64-bit Development with Embarcadero RAD Studio XE2

RAD Studio XE2 Overview

RAD Studio XE2 is the major breakthrough in the line of all Delphi versions of this product. It allows deploying your applications both on Windows and Mac OS platforms. Additionally, it is now possible to create 64-bit Windows applications to fully benefit from the power of new hardware. Moreover, you can create visually spectacular applications with the help of the FireMonkey GPU application platform.

Its main features are the following:

- Windows 64-bit platform support;

- Mac OS support;

- FireMonkey application development platform;

- Live data bindings with visual components;

- VCL styles for Windows applications.

Changes in 64-bit Application Development

64-bit platform support implies several important changes that each developer must keep in mind prior to the development of a new application or the modernization of an old one.

General

RAD Studio XE2 IDE is a 32-bit application. It means that it cannot load 64-bit packages at design-time. So, all design-time packages in RAD Studio XE2 IDE are 32-bit.

Therefore, if you develop your own components, you should remember that for the purpose of developing components with the 64-bit platform support, you have to compile run-time packages both for the 32- and 64-bit platforms, while design-time packages need to be

compiled only for the 32-bit platform. This might be a source of difficulties if your package is simultaneously both a run-time and a design-time package, as it is more than likely that this package won't be compiled for the 64-bit platform. In this case, you will have to separate your package into two packages, one of which will be used as run-time only, and the other as design-time only.

For the same reason, if your design-time packages require that certain DLLs be loaded, you should remember that design-time packages can be only 32-bit and that is why they can load only 32-bit versions of these DLLs, while at run-time 64-bit versions of the DLLs will be loaded. Correspondingly, if there are only 64-bit versions of the DLL on your computer, you won't be able to use all functions at design-time and, vice versa, if you have only 32-bit versions of the DLLs, your application won't be able to work at run-time.

Extended type

For this type in a 64-bit applications compiler generates SSE2 instructions instead of FPU, and that greatly improves performance in applications that use this type a lot (where data accuracy is needed). For this purpose, the size and precision of Extended type is reduced:

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

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

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

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

Pointer and Integers

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

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

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

TYPE	32-bit	64-bit
Integer	4 bytes	4 bytes

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

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

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

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

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

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

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

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

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

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

Out parameters

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

For example:

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

Win API

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

Correct:

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

Wrong:

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

Replace SetWindowLong/GetWindowLog with SetWindowLongPtr/GetWindowLongPtr for GWLP_HINSTANCE, GWLP_ID, GWLP_USERDATA, GWLP_HWNDPARENT and GWLP_WNDPROC as they return pointers and handles. Pointers that are passed to SetWindowLongPtr should be type-casted to LONG_PTR and not to Integer/Longint.

Correct:

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

Wrong:

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

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

Correct:

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

Wrong:

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

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

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

Assembler

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

- rewrite your code that mixes Pascal code and assembly code. Mixing them is not supported in 64-bit applications;

- rewrite assembly code that doesn't consider architecture and processor specifics.

You can use conditional defines to make your application work with different architectures.

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

Exception handling

The biggest difference in exception handling between Delphi 32 and 64-bit is that in Delphi XE2 64-bit you will gain more performance because of different internal exception mechanism. For 32-bit applications, the Delphi compiler (dcc32.exe) generates additional code that is executed any way and that causes performance loss. The 64-bit compiler (dcc64.exe) doesn't generate such code, it generates metadata and stores it in the PDATA section of an executable file instead.

But in Delphi XE2 64-bit it's impossible to have more than 16 levels of nested exceptions. Having more than 16 levels of nested exceptions will cause a Run Time error.

Debugging

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

To enable it, perform the following steps:

1. Open Project Options (in the main menu **Project->Options**).

2. In the Target combobox, select **Debug configuration - 64-bit Windows platform**. If there is no such option in the combobox, right click "Target Platforms" in Project Manager and select **Add platform**. After adding the 64-bit Windows platform, the **Debug configuration - 64-bit Windows platform** option will be available in the Target combobox.

3. Select **Linking** in the left part of the Project Options form.

4. enable the **Include remote debug symbols** option.

After that, you can run and debug your 64-bit application.

To enable remote debugging, perform the following steps:

1. Install Platform Assistant Server (PAServer) on a remote computer. You can find PAServer in the %RAD_Studio_XE2_Install_Directory%\PAServer directory. The setup_paserver.exe file is an installation file for Windows, and the setup_paserver.zip file is an istallation file for MacOS.

2. Run the PAServer.exe file on a remote computer and set the password that will be used to connect to this computer.

3. On a local computer with RAD Studio XE2 installed, right-click the target platform that you want to debug in Project Manager and select **Assign Remote Profile**. Click the **Add** button in the displayed window, input your profile name, click the **Next** button, input the name of a remote computer and the password to it (that you assigned when you started PAServer on a remote computer).

After that, you can test the connection by clicking the **Test Connection** button. If your connection failed, check that your firewalls on both remote and local computers do not block your connection, and try to establish a connection once more. If your connection succeeded,

click the Next button and then the Finish button. Select your newly created profile and click **OK**.

After performing these steps you will be able to debug your application on a remote computer. You application will be executed on a remote computer, but you will be able to debug it on your local computer with RAD Studio XE2.

For more information about working with Platform Assistant Server, please refer to http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Running_the_Platform_Assistant_on_Windows

Request Support   DAC Forum   Provide Feedback

4.20   Database Specific Aspects of 64-bit Development

MySQL Connectivity Aspects

Client mode:

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 MySQL 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 MySQL database in run-time (64-bit application), you must have the 64-bit client library placed to the C:\Windows\System32 directory.

DIRECT mode:

Since there is no need to install client library for the DIRECT mode, the specifics of developing applications that use MyDAC as data access components, depends exclusively on peculiarities of each target platform.

Request Support   DAC Forum   Provide Feedback

5 Reference

This page shortly describes units that exist in MyDAC.

Units

Unit Name	Description
[CRAccess](#)	This unit contains base classes for accessing databases.
[CRBatchMove](#)	This unit contains implementation of the TCRBatchMove component.
[CREncryption](#)	This unit contains base classes for data encryption.
[CRGrid](#)	This unit contains the TCRDBGrid component.
[CRVio](#)	This unit contains classes for HTTP connections.
[DAAlerter](#)	This unit contains the base class for the TMyAlerter component.
[DADump](#)	This unit contains the base class for the TMyDump component.
[DALoader](#)	This unit contains the base class for the TMyLoader component.
[DAScript](#)	This unit contains the base class for the TMyScript component.
[DASQLMonitor](#)	This unit contains the base class for the TMySQLMonitor component.
[DBAccess](#)	This unit contains base classes for most of the components.
Devart.Dac.DataAdapter	This unit contains implementation of the DADataAdapter class.
Devart.MyDac.DataAdapter	This unit contains implementation of the MyDataAdapter class.

MemData	This unit contains classes for storing data in memory.
MemDS	This unit contains implementation of the TMemDataSet class.
MyAccess	This unit contains implementation of most public classes of MyDAC.
MyBackup	This unit contains implementation of the TMyBackup component.
MyBuilderClient	This unit contains implementation of the MyBuilder class.
MyClasses	This unit contains implementation of the EMyError class.
MyConnectionPool	This unit contains the TMyConnectionPoolManager class for managing connection pool.
MyDacVcl	This unit contains the visual constituent of MyDAC.
MyDataTypeMap	This unit contains the implementation of mapping between MySQL and Delphi data types.
MyDump	This unit contains implementation of the TMyDump component.
MyEmbConnection	This unit contains implementation of the TMyEmbConnection component.
MyLoader	This unit contains implementation of the TMyLoader component.
MyScript	This unit contains implementation of the TMyScript component.
MyServerControl	This unit contains implementation of the TMyServerControl component.

MySqlApi	This unit contains implementation of the class.
MySQLMonitor	This unit contains implementation of the TMySQLMonitor component.
MySqlVioPipe	This unit contains implementation of the TCRIOHandler class.
VirtualDataSet	Description is not available at the moment.

© 1997-2025 Devart. All Rights Reserved.   **Request Support**   **DAC Forum**   **Provide Feedback**

5.1   CRAccess

This unit contains base classes for accessing databases.

Classes

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

Types

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

Enumerations

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

	the active transaction.
TCursorState	Used to set cursor state

Request Support   DAC Forum   Provide Feedback

5.1.1 Classes

Classes in the **CRAccess** unit.

Classes

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

Request Support   DAC Forum   Provide Feedback

5.1.1.1 TCRCursor Class

A base class for classes that work with database cursors.

For a list of all members of this type, see TCRCursor members.

Unit

CRAccess

Syntax

```
TCRCursor = class(TSharedObject);
```

Remarks

TCRCursor is a base class for classes that work with database cursors.

Inheritance Hierarchy

TSharedObject

    TCRCursor

Request Support   DAC Forum   Provide Feedback

5.1.1.1.1 Members

TCRCursor class overview.

Properties

Name	Description
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

Request Support   DAC Forum   Provide Feedback

5.1.2 Types

Types in the **CRAccess** unit.

Types

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

Request Support   DAC Forum   Provide Feedback

5.1.2.1 TBeforeFetchProc Procedure Reference

This type is used for the TCustomDADataSet.BeforeFetch event.

Unit

CRAccess

Syntax

```
TBeforeFetchProc = procedure (var Cancel: boolean) of object;
```

Parameters

Cancel
   True, if the current fetch operation should be aborted.

Request Support   DAC Forum   Provide Feedback

5.1.3   Enumerations

Enumerations in the **CRAccess** unit.

Enumerations

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

Request Support   DAC Forum   Provide Feedback

5.1.3.1 TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

Unit

CRAccess

Syntax

```
TCRIsolationLevel = (ilReadCommitted, ilReadUnCommitted,
ilRepeatableRead, ilIsolated, ilSnapshot, ilCustom);
```

Values

Value	Meaning
ilCustom	The parameters of the transaction are set manually in the Params property.
ilIsolated	The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting additional processing on range locks. Used to put aside all undesired effects observed in the concurrent accesses to the same set of data, but may lead to a greater latency at times of a congested database environment.
ilReadCommitted	Sets isolation level at which transaction cannot see changes made by outside transactions until they are committed. Only dirty reads (changes made by uncommitted transactions) are eliminated by this state of the isolation level. The default value.
ilReadUnCommitted	The most unrestricted level of the transaction isolation. All types of data access interferences are possible. Mainly used for browsing database and to receive instant data with prospective changes.
ilRepeatableRead	Prevents concurrent transactions from modifying data in the current uncommitted transaction. This level eliminates dirty reads as well as nonrepeatable reads (repeatable reads of the same data in one transaction before and after outside transactions may have started and committed).
ilSnapshot	Uses row versioning. Provides transaction-level read consistency. A data snapshot is taken when the snapshot transaction starts, and remains consistent for the duration of a transaction.

Request Support   DAC Forum   Provide Feedback

5.1.3.2   **TCRTransactionAction Enumeration**

Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Unit

CRAccess

Syntax

```
TCRTransactionAction = (taCommit, taRollback);
```

Values

Value	Meaning
taCommit	Transaction is committed.
taRollback	Transaction is rolled back.

   Request Support   DAC Forum   Provide Feedback

5.1.3.3   **TCursorState Enumeration**

Used to set cursor state

Unit

CRAccess

Syntax

```
TCursorState = (csInactive, csOpen, csParsed, csPrepared, csBound,
csExecuteFetchAll, csExecuting, csExecuted, csFetching,
csFetchingAll, csFetched);
```

Values

Value	Meaning
csBound	Parameters bound
csExecuted	Statement successfully executed
csExecuteFetchAll	Set before FetchAll
csExecuting	Statement is set before executing

csFetched	Fetch finished or canceled
csFetching	Set on first
csFetchingAll	Set on the FetchAll start
csInactive	Default state
csOpen	statement open
csParsed	Statement parsed
csPrepared	Statement prepared

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.2   CRBatchMove

This unit contains implementation of the TCRBatchMove component.

Classes

Name	Description
[TCRBatchMove](#)	Transfers records between datasets.

Types

Name	Description
[TCRBatchMoveProgressEvent](#)	This type is used for the [TCRBatchMove.OnBatchMoveProgress](#) event.

Enumerations

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

	list is empty.

Request Support   **DAC Forum**   **Provide Feedback**

5.2.1   Classes

Classes in the **CRBatchMove** unit.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

Request Support   **DAC Forum**   **Provide Feedback**

5.2.1.1   TCRBatchMove Class

Transfers records between datasets.

For a list of all members of this type, see TCRBatchMove members.

Unit

CRBatchMove

Syntax

```
TCRBatchMove = class(TComponent);
```

Remarks

The TCRBatchMove component transfers records between datasets. Use it to copy dataset records to another dataset or to delete datasets records that match records in another dataset. The TCRBatchMove.Mode property determines the desired operation type, the TCRBatchMove.Source and TCRBatchMove.Destination properties indicate corresponding datasets.

Note: A TCRBatchMove component is added to the Data Access page of the component palette, not to the MyDAC page.

Request Support   **DAC Forum**   **Provide Feedback**

5.2.1.1.1 Members

TCRBatchMove class overview.

Properties

Name	Description
AbortOnKeyViol	Used to specify whether the batch operation should be terminated immediately after key or integrity violation.
AbortOnProblem	Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.
ChangedCount	Used to get the number of records changed in the destination dataset.
CommitCount	Used to set the number of records to be batch moved before commit occurs.
Destination	Used to specify the destination dataset for the batch operation.
FieldMappingMode	Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.
KeyViolCount	Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.
Mappings	Used to set field matching between source and destination datasets for the batch operation.

[Mode](#)	Used to set the type of the batch operation that will be executed after calling the [TCRBatchMove.Execute](#) method.
[MovedCount](#)	Used to get the number of records that were read from the source dataset during the batch operation.
[ProblemCount](#)	Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.
[RecordCount](#)	Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.
[Source](#)	Used to specify the source dataset for the batch operation.

Methods

Name	Description
[Execute](#)	Performs the batch operation.

Events

Name	Description
[OnBatchMoveProgress](#)	Occurs when providing feedback to the user about the batch operation in progress is needed.

Request Support   **DAC Forum**   **Provide Feedback**

5.2.1.1.2 Properties

Properties of the **TCRBatchMove** class.

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

Public

Name	Description
ChangedCount	Used to get the number of records changed in the destination dataset.
KeyViolCount	Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.
MovedCount	Used to get the number of records that were read from the source dataset during the batch operation.
ProblemCount	Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.

Published

Name	Description
AbortOnKeyViol	Used to specify whether the batch operation should be terminated immediately after key or integrity violation.
AbortOnProblem	Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.
CommitCount	Used to set the number of records to be batch moved before commit occurs.

[Destination](#)	Used to specify the destination dataset for the batch operation.
[FieldMappingMode](#)	Used to specify the way fields of destination and source datasets will be mapped to each other if the [TCRBatchMove.Mappings](#) list is empty.
[Mappings](#)	Used to set field matching between source and destination datasets for the batch operation.
[Mode](#)	Used to set the type of the batch operation that will be executed after calling the [TCRBatchMove.Execute](#) method.
[RecordCount](#)	Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.
[Source](#)	Used to specify the source dataset for the batch operation.

See Also

- [TCRBatchMove Class](#)

- [TCRBatchMove Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.2.1.1.2.1 AbortOnKeyViol Property

Used to specify whether the batch operation should be terminated immediately after key or integrity violation.

Class

[TCRBatchMove](#)

Syntax

```
property AbortOnKeyViol: boolean default True;
```

Remarks

Use the AbortOnKeyViol property to specify whether the batch operation is terminated immediately after key or integrity violation.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.2.1.1.2.2 AbortOnProblem Property

Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.

Class

[TCRBatchMove](#)

Syntax

```
property AbortOnProblem: boolean default True;
```

Remarks

Use the AbortOnProblem property to specify whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit the specified Destination.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.2.1.1.2.3 ChangedCount Property

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

Class

[TCRBatchMove](#)

Syntax

```
property ChangedCount: Integer;
```

Remarks

Use the ChangedCount property to get the number of records changed in the destination dataset. It shows the number of records that were updated in the bmUpdate or bmAppendUpdate mode or were deleted in the bmDelete mode.

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.4 CommitCount Property

Used to set the number of records to be batch moved before commit occurs.

Class

TCRBatchMove

Syntax

```
property CommitCount: integer default 0;
```

Remarks

Use the CommitCount property to set the number of records to be batch moved before the commit occurs. If it is set to 0, the operation will be chunked to the number of records to fit 32 Kb.

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.5 Destination Property

Used to specify the destination dataset for the batch operation.

Class

TCRBatchMove

Syntax

```
property Destination: TDataSet;
```

Remarks

Specifies the destination dataset for the batch operation.

5.2.1.1.2.6 FieldMappingMode Property

Used to specify the way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.

Class

`TCRBatchMove`

Syntax

```
property FieldMappingMode: TCRFieldMappingMode default
mmFieldIndex;
```

Remarks

Specifies in what way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.

5.2.1.1.2.7 KeyViolCount Property

Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.

Class

`TCRBatchMove`

Syntax

```
property KeyViolCount: Integer;
```

Remarks

Use the KeyViolCount property to get the number of records that could not be replaced,

added, deleted from the destination dataset because of integrity or key violations.

If AbortOnKeyViol is True, then KeyViolCount will never exceed one, because the operation aborts when the integrity or key violation occurs.

See Also

- AbortOnKeyViol

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.8 Mappings Property

Used to set field matching between source and destination datasets for the batch operation.

Class

`TCRBatchMove`

Syntax

```
property Mappings: TStrings;
```

Remarks

Use the Mappings property to set field matching between the source and destination datasets for the batch operation. By default fields matching is based on their position in the datasets. To map the column ColName in the source dataset to the column with the same name in the destination dataset, use:

ColName

Example

To map a column named SourceColName in the source dataset to the column named DestColName in the destination dataset, use:

```
DestColName=SourceColName
```

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.9 Mode Property

Used to set the type of the batch operation that will be executed after calling the Execute method.

Class

`TCRBatchMove`

Syntax

```
property Mode: TCRBatchMode default bmAppend;
```

Remarks

Use the Mode property to set the type of the batch operation that will be executed after calling the Execute method.

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.10 MovedCount Property

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

Class

`TCRBatchMove`

Syntax

```
property MovedCount: Integer;
```

Remarks

Use the MovedCount property to get the number of records that were read from the source dataset during the batch operation. This number includes records that caused key or integrity violations or were trimmed.

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.11 ProblemCount Property

Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.

Class

`TCRBatchMove`

Syntax

```
property ProblemCount: Integer;
```

Remarks

Use the ProblemCount property to get the number of records that could not be added to the destination dataset because of the field type mismatch.

If AbortOnProblem is True, then ProblemCount will never exceed one, because the operation aborts when the problem occurs.

See Also

* AbortOnProblem

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.12 RecordCount Property

Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.

Class

`TCRBatchMove`

Syntax

```
property RecordCount: Integer default 0;
```

Remarks

Determines the maximum number of records in the source dataset, that will be applied to the

destination dataset. If it is set to 0, all records in the source dataset will be applied to the destination dataset, starting from the first record. If RecordCount is greater than 0, up to the RecordCount records are applied to the destination dataset, starting from the current record in the source dataset. If RecordCount exceeds the number of records left in the source dataset, batch operation terminates after reaching last record.

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.13 Source Property

Used to specify the source dataset for the batch operation.

Class

TCRBatchMove

Syntax

```
property Source: TDataSet;
```

Remarks

Specifies the source dataset for the batch operation.

Request Support   DAC Forum   Provide Feedback

5.2.1.1.3 Methods

Methods of the **TCRBatchMove** class.

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

Public

Name	Description
Execute	Performs the batch operation.

See Also

- [TCRBatchMove Class](#)

- [TCRBatchMove Class Members](#)

   Request Support   **DAC Forum**   **Provide Feedback**

5.2.1.1.3.1 Execute Method

Performs the batch operation.

Class

`TCRBatchMove`

Syntax

```
procedure Execute;
```

Remarks

Call the Execute method to perform the batch operation.

   Request Support   **DAC Forum**   **Provide Feedback**

5.2.1.1.4 Events

Events of the **TCRBatchMove** class.

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

Published

Name	Description
[OnBatchMoveProgress](#)	Occurs when providing feedback to the user about the batch operation in progress is needed.

See Also

- [TCRBatchMove Class](#)

- [TCRBatchMove Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.2.1.1.4.1 OnBatchMoveProgress Event

Occurs when providing feedback to the user about the batch operation in progress is needed.

Class

[TCRBatchMove](#)

Syntax

```
property OnBatchMoveProgress: TCRBatchMoveProgressEvent;
```

Remarks

Write the OnBatchMoveProgress event handler to provide feedback to the user about the batch operation progress.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.2.2 Types

Types in the **CRBatchMove** unit.

Types

Name	Description
[TCRBatchMoveProgressEvent](#)	This type is used for the [TCRBatchMove.OnBatchMoveProgress](#) event.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.2.2.1 TCRBatchMoveProgressEvent Procedure Reference

This type is used for the TCRBatchMove.OnBatchMoveProgress event.

Unit

CRBatchMove

Syntax

```
TCRBatchMoveProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender

  An object that raised the event.

Percent

  Percentage of the batch operation progress.

Request Support   DAC Forum   Provide Feedback

5.2.3 Enumerations

Enumerations in the **CRBatchMove** unit.

Enumerations

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

Request Support   DAC Forum   Provide Feedback

5.2.3.1 TCRBatchMode Enumeration

Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.

Unit

CRBatchMove

Syntax

```
TCRBatchMode = (bmAppend, bmUpdate, bmAppendUpdate, bmDelete);
```

Values

Value	Meaning
bmAppend	Appends the records from the source dataset to the destination dataset. The default mode.
bmAppendUpdate	Replaces records in the destination dataset with the matching records from the source dataset. If there is no matching record in the destination dataset, the record will be appended to it.
bmDelete	Deletes records from the destination dataset if there are matching records in the source dataset.
bmUpdate	Replaces records in the destination dataset with the matching records from the source dataset.

Request Support   DAC Forum   Provide Feedback

5.2.3.2 TCRFieldMappingMode Enumeration

Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.

Unit

CRBatchMove

Syntax

```
TCRFieldMappingMode = (mmFieldIndex, mmFieldName);
```

Values

Value	Meaning
mmFieldIndex	Specifies that the fields of the destination dataset will be mapped to the fields of the source dataset by field index.
mmFieldName	Mapping is performed by field names.

Request Support   DAC Forum   Provide Feedback

5.3 CREncryption

This unit contains base classes for data encryption.

Classes

Name	Description
TCREncryptor	The class that performs data encryption and decryption in a client application using various encryption algorithms.

Enumerations

Name	Description
TCREncDataHeader	Specifies whether the additional information is stored with the encrypted data.
TCREncryptionAlgorithm	Specifies the algorithm of data encryption.
TCRHashAlgorithm	Specifies the algorithm of generating hash data.
TCRInvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.

Request Support   DAC Forum   Provide Feedback

5.3.1 Classes

Classes in the **CREncryption** unit.

Classes

Name	Description
[TCREncryptor](#)	The class that performs data encryption and decryption in a client application using various [encryption algorithms](#).

   Request Support   DAC Forum   Provide Feedback

5.3.1.1 TCREncryptor Class

The class that performs data encryption and decryption in a client application using various [encryption algorithms](#).

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

Unit

CREncryption

Syntax

```
TCREncryptor = class(TComponent);
```

   Request Support   DAC Forum   Provide Feedback

5.3.1.1.1 Members

TCREncryptor class overview.

Properties

Name	Description
[DataHeader](#)	Specifies whether the additional information is stored with the encrypted

	data.
[EncryptionAlgorithm](#)	Specifies the algorithm of data encryption.
[HashAlgorithm](#)	Specifies the algorithm of generating hash data.
[InvalidHashAction](#)	Specifies the action to perform on data fetching when hash data is invalid.
[Password](#)	Used to set a password that is used to generate a key for encryption.

Methods

Name	Description
[SetKey](#)	Sets a key, using which data is encrypted.

[Request Support](#)   **[DAC Forum](#)**   **[Provide Feedback](#)**

5.3.1.1.2 Properties

Properties of the **TCREncryptor** class.

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

Published

Name	Description
[DataHeader](#)	Specifies whether the additional information is stored with the encrypted data.
[EncryptionAlgorithm](#)	Specifies the algorithm of data encryption.
[HashAlgorithm](#)	Specifies the algorithm of generating hash data.
[InvalidHashAction](#)	Specifies the action to perform on data fetching when hash data is invalid.

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

See Also

- TCREncryptor Class

- TCREncryptor Class Members

Request Support   DAC Forum   Provide Feedback

5.3.1.1.2.1 DataHeader Property

Specifies whether the additional information is stored with the encrypted data.

Class

`TCREncryptor`

Syntax

```
property DataHeader: TCREncDataHeader default ehTagAndHash;
```

Remarks

Use DataHeader to specify whether the additional information is stored with the encrypted data. Default value is ehTagAndHash.

Request Support   DAC Forum   Provide Feedback

5.3.1.1.2.2 EncryptionAlgorithm Property

Specifies the algorithm of data encryption.

Class

`TCREncryptor`

Syntax

```
property EncryptionAlgorithm: TCREncryptionAlgorithm default
```

```
eaBlowfish;
```

Remarks

Use EncryptionAlgorithm to specify the algorithm of data encryption. Default value is

eaBlowfish.

Request Support   DAC Forum   Provide Feedback

5.3.1.1.2.3 HashAlgorithm Property

Specifies the algorithm of generating hash data.

Class

TCREncryptor

Syntax

```
property HashAlgorithm: TCRHashAlgorithm default haSHA1;
```

Remarks

Use HashAlgorithm to specify the algorithm of generating hash data. This property is used
only if hash is stored with the encrypted data (the DataHeader property is set to
ehTagAndHash). Default value is haSHA1.

Request Support   DAC Forum   Provide Feedback

5.3.1.1.2.4 InvalidHashAction Property

Specifies the action to perform on data fetching when hash data is invalid.

Class

TCREncryptor

Syntax

```
property InvalidHashAction: TCRInvalidHashAction default ihFail;
```

Remarks

Use InvalidHashAction to specify the action to perform on data fetching when hash data is invalid. This property is used only if hash is stored with the encrypted data (the DataHeader property is set to ehTagAndHash). Default value is ihFail.

If the DataHeader property is set to ehTagAndHash, then on data fetching from a server the hash check is performed for each record. After data decryption its hash is calculated and compared with the hash stored in the field. If these values don't coincide, it means that the stored data is incorrect, and depending on the value of the InvalidHashAction property one of the following actions is performed:

ihFail - the EInvalidHash exception is raised and further data reading from the server is interrupted.

ihSkipData - the value of the field for this record is set to Null. No exception is raised.

ihIgnoreError - in spite of the fact that the data is not valid, the value is set in the field. No exception is raised.

Request Support   DAC Forum   Provide Feedback

5.3.1.1.2.5 Password Property

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

Class

TCREncryptor

Syntax

```
property Password: string stored False;
```

Remarks

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

Note: Calling of the SetKey method clears the Password property.

Request Support   DAC Forum   Provide Feedback

5.3.1.1.3 Methods

Methods of the **TCREncryptor** class.

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

Public

Name	Description
SetKey	Sets a key, using which data is encrypted.

See Also

- TCREncryptor Class

- TCREncryptor Class Members

Request Support   DAC Forum   Provide Feedback

5.3.1.1.3.1 SetKey Method

Sets a key, using which data is encrypted.

Class

TCREncryptor

Syntax

```
procedure SetKey(const Key; Count: Integer); overload;procedure
SetKey(const Key: TBytes; Offset: Integer; Count: Integer);
overload;
```

Parameters

Key
  Holds bytes that represent a key.

Offset
  Offset in bytes to the position, where the key begins.

Count
  Number of bytes to use from Key.

Remarks

Use SetKey to set a key, using which data is encrypted.

Note: Calling of the SetKey method clears the Password property.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.3.2   Enumerations

Enumerations in the **CREncryption** unit.

Enumerations

Name	Description
[TCREncDataHeader](#)	Specifies whether the additional information is stored with the encrypted data.
[TCREncryptionAlgorithm](#)	Specifies the algorithm of data encryption.
[TCRHashAlgorithm](#)	Specifies the algorithm of generating hash data.
[TCRInvalidHashAction](#)	Specifies the action to perform on data fetching when hash data is invalid.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.3.2.1   TCREncDataHeader Enumeration

Specifies whether the additional information is stored with the encrypted data.

Unit

[CREncryption](#)

Syntax

```
TCREncDataHeader = (ehTagAndHash, ehTag, ehNone);
```

Values

Value	Meaning
ehNone	No additional information is stored.
ehTag	GUID and the random initialization vector are stored with the encrypted data.
ehTagAndHash	Hash, GUID, and the random initialization vector are stored with the encrypted data.

Request Support   DAC Forum   Provide Feedback

5.3.2.2   TCREncryptionAlgorithm Enumeration

Specifies the algorithm of data encryption.

Unit

[CREncryption](#)

Syntax

```
TCREncryptionAlgorithm = (eaTripleDES, eaBlowfish, eaAES128,
eaAES192, eaAES256, eaCast128, eaRC4);
```

Values

Value	Meaning
eaAES128	The AES encryption algorithm with key size of 128 bits is used.
eaAES192	The AES encryption algorithm with key size of 192 bits is used.
eaAES256	The AES encryption algorithm with key size of 256 bits is used.
eaBlowfish	The Blowfish encryption algorithm is used.
eaCast128	The CAST-128 encryption algorithm with key size of 128 bits is used.
eaRC4	The RC4 encryption algorithm is used.
eaTripleDES	The Triple DES encryption algorithm is used.

Request Support   DAC Forum   Provide Feedback

5.3.2.3   TCRHashAlgorithm Enumeration

Specifies the algorithm of generating hash data.

Unit

CREncryption

Syntax

```
TCRHashAlgorithm = (haSHA1, haMD5);
```

Values

Value	Meaning
haMD5	The MD5 hash algorithm is used.
haSHA1	The SHA-1 hash algorithm is used.

Request Support   DAC Forum   Provide Feedback

5.3.2.4   TCRInvalidHashAction Enumeration

Specifies the action to perform on data fetching when hash data is invalid.

Unit

CREncryption

Syntax

```
TCRInvalidHashAction = (ihFail, ihSkipData, ihIgnoreError);
```

Values

Value	Meaning
ihFail	The EInvalidHash exception is raised and further data reading from the server is interrupted.
ihIgnoreError	In spite of the fact that the data is not valid, the value is set in the field. No exception is raised.
ihSkipData	The value of the field for this record is set to Null. No exception is raised.

Request Support   DAC Forum   Provide Feedback

5.4 CRGrid

This unit contains the TCRDBGrid component.

Classes

Name	Description
TCRDBGrid	Extends the functionality of the TDBGrid component.

Request Support   DAC Forum   Provide Feedback

5.4.1 Classes

Classes in the **CRGrid** unit.

Classes

Name	Description
TCRDBGrid	Extends the functionality of the TDBGrid component.

Request Support   DAC Forum   Provide Feedback

5.4.1.1 TCRDBGrid Class

Extends the functionality of the TDBGrid component.

For a list of all members of this type, see TCRDBGrid members.

Unit

CRGrid

Syntax

```
TCRDBGrid = class(TCustomDBGrid);
```

Request Support   DAC Forum   Provide Feedback

Reserved.

5.4.1.1.1 Members

TCRDBGrid class overview.

Properties

Name	Description
Columns	To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.
Filtered	Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.
LevelDelimiterChar	Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.
OnMemoClick	Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor.
OptionsEx	OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.
OptionsMenu	TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this

	menu using OptionsMenu property.

Methods

Name	Description
[ActivateFilterEdit](#)	Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.
[ActivateSearchEdit](#)	Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.
[AdjustColumns](#)	Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.
[ApplyFilter](#)	Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.
[CalcTitleLevel](#)	Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.
[ClearFilters](#)	Call CalcFilters method to clear all filter statements for grid columns.
[ClearSorting](#)	Call ClearSorting method to discard sorting previously applied to the grid data.

[DataChanged](#)	Call DataChanged method to update displayed data and status information for the grid.
[GetGridSize](#)	Call GetGridSize method to obtain width of a client area for the grid component.
[GetTitleLevel](#)	GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.

Events

Name	**Description**
[OnGetCellParams](#)	Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.2 Properties

Properties of the **TCRDBGrid** class.

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

Public

Name	**Description**
[OptionsMenu](#)	TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property.

Published

Name	Description
[Columns](#)	To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.
[Filtered](#)	Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.
[LevelDelimiterChar](#)	Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.
[OnMemoClick](#)	Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor.
[OptionsEx](#)	OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.

See Also

- [TCRDBGrid Class](#)

- [TCRDBGrid Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.2.1 Columns Property

To provide extended functionality TCRDBGrid component is shipped with a TCRColumn

class, which is derived from the standard TColumn class but is tailored to work with

TCRDBGrid only.

Class

[TCRDBGrid](#)

Syntax

```
property Columns: TCRDBGridColumns stored False;
```

Remarks

When browsing TDBGridColumns remember that its Items property actually returns TCRColumn objects when indexed.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.2.2 Filtered Property

Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.

Class

[TCRDBGrid](#)

Syntax

```
property Filtered: boolean default True;
```

Remarks

Conditional statements of all columns are combined to form a single filter.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.2.3 LevelDelimiterChar Property

Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.

Class

[TCRDBGrid](#)

Syntax

```
property LevelDelimiterChar: char default '|';
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.2.4 OnMemoClick Property

Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor.

Class

[TCRDBGrid](#)

Syntax

```
property OnMemoClick: TOnMemoClick;
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.2.5 OptionsEx Property

OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.

Class

[TCRDBGrid](#)

Syntax

```
property OptionsEx: TCRDBGridOptionsEx default [dgeEnableSort,
dgeLocalFilter, dgeLocalSorting, dgeRecordCount];
```

Remarks

Following values are supported for the OptionsEx set:

dgeEnableSort - enables sorting of records by the specified column clicking on the column

title.

dgeFilterBar - shows filter bar below grid caption where filter expressions for every column are entered manually.

dgeLocalFilter - records are filtered using associated dataset component facilities. Otherwise database server processes all filter operations.

dgeLocalSorting - records are sorted using associated dataset component facilities. Otherwise database server processes all sorting operations.

dgeRecordCount - displays status bar with current record number in respect to the total number of records. This option is mutually exclusive with dgeSummary option.

dgeSearchBar - shows search bar below grid caption where incremental search expression is entered for every column.

dgeStretch - makes all columns adjust their widths so that they fin onto entire grid area.

dgeSummary - displays status bar with summary information for every column. See TCRColumn.SummaryMode property on description of available options for individual columns. dgeSummary is mutually exclusive with dgeRecordCount option.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.2.6 OptionsMenu Property

TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property.

Class

[TCRDBGrid](#)

Syntax

```
property OptionsMenu: TPopupMenu;
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.3 Methods

Methods of the **TCRDBGrid** class.

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

Public

Name	Description
ActivateFilterEdit	Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.
ActivateSearchEdit	Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.
AdjustColumns	Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.
ApplyFilter	Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.
CalcTitleLevel	Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.
ClearFilters	Call CalcFilters method to clear all filter statements for grid columns.
ClearSorting	Call ClearSorting method to

	discard sorting previously applied to the grid data.
DataChanged	Call DataChanged method to update displayed data and status information for the grid.
GetGridSize	Call GetGridSize method to obtain width of a client area for the grid component.
GetTitleLevel	GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.

See Also

- TCRDBGrid Class

- TCRDBGrid Class Members

Request Support   DAC Forum   Provide Feedback

5.4.1.1.3.1 ActivateFilterEdit Method

Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.

Class

TCRDBGrid

Syntax

```
procedure ActivateFilterEdit(Column: TColumn);
```

Parameters

Column

Request Support   DAC Forum   Provide Feedback

5.4.1.1.3.2 ActivateSearchEdit Method

Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.

Class

`TCRDBGrid`

Syntax

```
procedure ActivateSearchEdit(Column: TColumn);
```

Parameters

Column

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.3.3 AdjustColumns Method

Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.

Class

`TCRDBGrid`

Syntax

```
procedure AdjustColumns;
```

Remarks

Visual feedback is immediate if OptionsEx property includes dgeLocalFilter option.

Note: Since filtering is performed at session level then other data-aware controls may also be updated.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.4.1.1.3.4 ApplyFilter Method

Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.

Class

TCRDBGrid

Syntax

```
procedure ApplyFilter;
```

Remarks

Visual feedback is immediate if OptionsEx property includes dgeLocalFilter option.

Note: Since filtering is performed at session level then other data-aware controls may also be updated.

© 1997-2025 Devart. All Rights Reserved. | Request Support | DAC Forum | Provide Feedback

5.4.1.1.3.5 CalcTitleLevel Method

Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.

Class

TCRDBGrid

Syntax

```
procedure CalcTitleLevel(Level: integer; var aRect: TRect);
```

Parameters

Level

aRect

© 1997-2025 Devart. All Rights Reserved. | Request Support | DAC Forum | Provide Feedback

5.4.1.1.3.6 ClearFilters Method

Call CalcFilters method to clear all filter statements for grid columns.

Class

TCRDBGrid

Syntax

```
procedure ClearFilters;
```

5.4.1.1.3.7 ClearSorting Method

Call ClearSorting method to discard sorting previously applied to the grid data.

Class

TCRDBGrid

Syntax

```
procedure ClearSorting;
```

5.4.1.1.3.8 DataChanged Method

Call DataChanged method to update displayed data and status information for the grid.

Class

TCRDBGrid

Syntax

```
procedure DataChanged;
```

5.4.1.1.3.9 GetGridSize Method

Call GetGridSize method to obtain width of a client area for the grid component.

Class

TCRDBGrid

Syntax

```
function GetGridSize: integer;
```

Remarks

Client area accommodates cell values only without column grid lines.

Request Support DAC Forum Provide Feedback

5.4.1.1.3.10 GetTitleLevel Method

GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.

Class

TCRDBGrid

Syntax

```
function GetTitleLevel(Level: integer): TRect;
```

Parameters

Level

Request Support DAC Forum Provide Feedback

5.4.1.1.4 Events

Events of the **TCRDBGrid** class.

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

Published

Name	Description
OnGetCellParams	Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.

See Also

- TCRDBGrid Class

- TCRDBGrid Class Members

Request Support   DAC Forum   Provide Feedback

5.4.1.1.4.1 OnGetCellParams Event

Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.

Class

TCRDBGrid

Syntax

```
property OnGetCellParams: TGetCellParamsEvent;
```

Request Support   DAC Forum   Provide Feedback

5.5   CRVio

This unit contains classes for HTTP connections.

Classes

Name	Description
THttpOptions	This class is used to establish an HTTP connection.
TProxyOptions	This class is used to

	establish an HTTP connection through a proxy server.

Enumerations

Name	Description
TIPVersion	Specifies Internet Protocol version.

Request Support   DAC Forum   Provide Feedback

5.5.1   Classes

Classes in the **CRVio** unit.

Classes

Name	Description
THttpOptions	This class is used to establish an HTTP connection.
TProxyOptions	This class is used to establish an HTTP connection through a proxy server.

Request Support   DAC Forum   Provide Feedback

5.5.1.1   THttpOptions Class

This class is used to establish an HTTP connection.

For a list of all members of this type, see THttpOptions members.

Unit

CRVio

Syntax

```
THttpOptions = class(TPersistent);
```

Remarks

The THttpOptions class is used to establish an HTTP connection.

For more information about HTTP tunneling, see Network Tunneling .

See Also

- Network Tunneling

Request Support   DAC Forum   Provide Feedback

5.5.1.1.1 Members

THttpOptions class overview.

Properties

Name	Description
Enabled	Enables an HTTP connection.
Password	Holds the password for HTTP authorization.
ProxyOptions	Holds a TProxyOptions object that contains settings for a proxy connection.
TrustServerCertificate	Verifies the server certificate during an SSL handshake.
Url	Holds the URL of the PHP script for HTTP tunneling.
Username	Holds the username for HTTP authorization.

Request Support   DAC Forum   Provide Feedback

5.5.1.1.2 Properties

Properties of the **THttpOptions** class.

For a complete list of the **THttpOptions** class members, see the THttpOptions Members

topic.

Public

Name	Description
Enabled	Enables an HTTP connection.
ProxyOptions	Holds a TProxyOptions object that contains settings for a proxy connection.

Published

Name	Description
Password	Holds the password for HTTP authorization.
TrustServerCertificate	Verifies the server certificate during an SSL handshake.
Url	Holds the URL of the PHP script for HTTP tunneling.
Username	Holds the username for HTTP authorization.

See Also

- THttpOptions Class

- THttpOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.5.1.1.2.1 Enabled Property

Enables an HTTP connection.

Class

`THttpOptions`

Syntax

```
property Enabled: boolean default False;
```

Remarks

The Enabled property specifies that a connection is established through HTTP.

5.5.1.1.2.2 Password Property

Holds the password for HTTP authorization.

Class

THttpOptions

Syntax

```
property Password: string;
```

Remarks

The Password property holds the password for the password-protected directory that contains the HTTP tunneling script.

5.5.1.1.2.3 ProxyOptions Property

Holds a TProxyOptions object that contains settings for a proxy connection.

Class

THttpOptions

Syntax

```
property ProxyOptions: TProxyOptions;
```

Remarks

The ProxyOptions property holds a TProxyOptions object that contains settings for a proxy connection.

If it is necessary to connect to the server that resides in a different network, sometimes the client can only connect to it through a proxy server. In this case, besides the connection string, you have to set up ProxyOptions.

　　Request Support　　　　DAC Forum　　　　　　Provide Feedback

5.5.1.1.2.4 TrustServerCertificate Property

Verifies the server certificate during an SSL handshake.

Class

THttpOptions

Syntax

```
property TrustServerCertificate: boolean default False;
```

Remarks

The TrustServerCertificate property specifies whether to verify the server certificate during an SSL handshake. When True, the MyDac bypasses walking the certificate chain to verify the certificate. The default value is False.

　　Request Support　　　　DAC Forum　　　　　　Provide Feedback

5.5.1.1.2.5 Url Property

Holds the URL of the PHP script for HTTP tunneling.

Class

THttpOptions

Syntax

```
property Url: string;
```

Remarks

The Url property holds the URL of the PHP script for HTTP tunneling. For example, if the

script is located in the server root, the URL can be the following: http://server/tunnel.php.

5.5.1.1.2.6 Username Property

Holds the username for HTTP authorization.

Class

`THttpOptions`

Syntax

```
property Username: string;
```

Remarks

The Username property holds the username for the password-protected directory that contains the HTTP tunneling script.

5.5.1.2   TProxyOptions Class

This class is used to establish an HTTP connection through a proxy server.

For a list of all members of this type, see TProxyOptions members.

Unit

`CRVio`

Syntax

```
TProxyOptions = class(TPersistent);
```

Remarks

The TProxyOptions class is used to establish an HTTP connection through a proxy server.

Reserved.

5.5.1.2.1 Members

TProxyOptions class overview.

Properties

Name	Description
[Hostname](#)	Holds the hostname or IP address of the proxy server.
[Password](#)	Holds the proxy password.
[Port](#)	Holds the port number of the proxy server.
[Username](#)	Holds the proxy username.

Request Support   **DAC Forum**   **Provide Feedback**

5.5.1.2.2 Properties

Properties of the **TProxyOptions** class.

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

Published

Name	Description
[Hostname](#)	Holds the hostname or IP address of the proxy server.
[Password](#)	Holds the proxy password.
[Port](#)	Holds the port number of the proxy server.
[Username](#)	Holds the proxy username.

See Also

- [TProxyOptions Class](#)

- [TProxyOptions Class Members](#)

Request Support   DAC Forum   Provide Feedback

5.5.1.2.2.1 Hostname Property

Holds the hostname or IP address of the proxy server.

Class

TProxyOptions

Syntax

```
property Hostname: string;
```

Remarks

The Hostname property holds the hostname or IP address of the proxy server.

Request Support   DAC Forum   Provide Feedback

5.5.1.2.2.2 Password Property

Holds the proxy password.

Class

TProxyOptions

Syntax

```
property Password: string;
```

Remarks

The Password property holds the proxy password.

Request Support   DAC Forum   Provide Feedback

5.5.1.2.2.3 Port Property

Holds the port number of the proxy server.

Class

TProxyOptions

Syntax

```
property Port: integer default 0;
```

Remarks

Use the Port property to specify the port number of the proxy server.

© 1997-2025 Devart. All Rights Reserved.	Request Support	DAC Forum	Provide Feedback

5.5.1.2.2.4 Username Property

Holds the proxy username.

Class

TProxyOptions

Syntax

```
property Username: string;
```

Remarks

The Username property holds the proxy username.

© 1997-2025 Devart. All Rights Reserved.	Request Support	DAC Forum	Provide Feedback

5.5.2   Enumerations

Enumerations in the **CRVio** unit.

Enumerations

Name	Description

TIPVersion	Specifies Internet Protocol version.

Request Support   DAC Forum   Provide Feedback

5.5.2.1 TIPVersion Enumeration

Specifies Internet Protocol version.

Unit

`CRVio`

Syntax

```
TIPVersion = (ivIPv4, ivIPv6, ivIPBoth);
```

Values

Value	Meaning
ivIPBoth	Specifies that either IPv6 or IPv4 Internet Protocol version is used
ivIPv4	Specifies that the IPv4 Internet Protocol version is used
ivIPv6	Specifies that the IPv6 Internet Protocol version is used

Remarks

Note: When the TIPVersion property is set to **ivIPBoth** , a connection attempt is made via IPv6 if it is enabled in the operating system settings. If the connection attempt fails, a new connection attempt is made via IPv4.

See Also

- TMyConnectionOptions.IPVersion

Request Support   DAC Forum   Provide Feedback

5.6 DAAlerter

This unit contains the base class for the TMyAlerter component.

Classes

Name	Description
[TDAAlerter](#)	A base class that defines functionality for database event notification.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.6.1   Classes

Classes in the **DAAlerter** unit.

Classes

Name	Description
[TDAAlerter](#)	A base class that defines functionality for database event notification.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.6.1.1   TDAAlerter Class

A base class that defines functionality for database event notification.

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

Unit

[DAAlerter](#)

Syntax

```
TDAAlerter = class(TComponent);
```

Remarks

TDAAlerter is a base class that defines functionality for descendant classes support database event notification. Applications never use TDAAlerter objects directly. Instead they use descendants of TDAAlerter.

The TDAAlerter component allows you to register interest in and handle events posted by a

database server. Use TDAAlerter to handle events for responding to actions and database changes made by other applications. To get events, an application must register required events. To do this, set the Events property to the required events and call the Start method. When one of the registered events occurs OnEvent handler is called.

Request Support   **DAC Forum**   **Provide Feedback**

5.6.1.1.1 Members

TDAAlerter class overview.

Properties

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever connection opens.

Methods

Name	Description
SendEvent	Sends an event with Name and content Message.
Start	Starts waiting process.
Stop	Stops waiting process.

Events

Name	Description
OnError	Occurs if an exception occurs in waiting process

Request Support   **DAC Forum**   **Provide Feedback**

5.6.1.1.2 Properties

Properties of the **TDAAlerter** class.

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

Public

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever connection opens.

See Also

- TDAAlerter Class

- TDAAlerter Class Members

Request Support   DAC Forum   Provide Feedback

5.6.1.1.2.1 Active Property

Used to determine if TDAAlerter waits for messages.

Class

TDAAlerter

Syntax

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

Remarks

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

See Also

- Start

- **Stop**

5.6.1.1.2.2 AutoRegister Property

Used to automatically register events whenever connection opens.

Class

[TDAAlerter](#)

Syntax

```
property AutoRegister: boolean default False;
```

Remarks

Set the AutoRegister property to True to automatically register events whenever connection opens.

5.6.1.1.3 Methods

Methods of the **TDAAlerter** class.

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

Public

Name	Description
[SendEvent](#)	Sends an event with Name and content Message.
[Start](#)	Starts waiting process.
[Stop](#)	Stops waiting process.

See Also

- [TDAAlerter Class](#)

- [TDAAlerter Class Members](#)

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.6.1.1.3.1 SendEvent Method

Sends an event with Name and content Message.

Class

[TDAAlerter](#)

Syntax

```
procedure SendEvent(const EventName: string; const Message: string);
```

Parameters

EventName
  Holds the event name.

Message
  Holds the content Message of the event.

Remarks

Use SendEvent procedure to send an event with Name and content Message.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.6.1.1.3.2 Start Method

Starts waiting process.

Class

[TDAAlerter](#)

Syntax

```
procedure Start;
```

Remarks

Call the Start method to run waiting process. After starting TDAAlerter waits for messages with names defined by the Events property.

See Also

- [Stop](#)

- [Active](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.6.1.1.3.3 Stop Method

Stops waiting process.

Class

[TDAAlerter](#)

Syntax

```
procedure Stop;
```

Remarks

Call Stop method to end waiting process.

See Also

- [Start](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.6.1.1.4 Events

Events of the **TDAAlerter** class.

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

Public

Name	Description

OnError	Occurs if an exception occurs in waiting process

See Also

- TDAAlerter Class

- TDAAlerter Class Members

Request Support   DAC Forum   Provide Feedback

5.6.1.1.4.1 OnError Event

Occurs if an exception occurs in waiting process

Class

TDAAlerter

Syntax

```
property OnError: TAlerterErrorEvent;
```

Remarks

The OnError event occurs if an exception occurs in waiting process. Alerter stops in this case. The exception can be accessed using the E parameter.

Request Support   DAC Forum   Provide Feedback

5.7   DADump

This unit contains the base class for the TMyDump 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.

© 1997-2025
Devart. All Rights
Reserved.

Request Support   **DAC Forum**   **Provide Feedback**

5.7.1   Classes

Classes in the **DADump** unit.

Classes

Name	Description
[TDADump](#)	A base class that defines functionality for descendant classes that dump database objects to a script.
[TDADumpOptions](#)	This class allows setting up the behaviour of the TDADump class.

© 1997-2025
Devart. All Rights
Reserved.

Request Support   **DAC Forum**   **Provide Feedback**

5.7.1.1   TDADump Class

A base class that defines functionality for descendant classes that dump database objects to a script.

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

Unit

DADump

Syntax

```
TDADump = class(TComponent);
```

Remarks

TDADump is a base class that defines functionality for descendant classes that dump database objects to a script. Applications never use TDADump objects directly. Instead they use descendants of TDADump.

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

Request Support   DAC Forum   Provide Feedback

5.7.1.1.1 Members

TDADump class overview.

Properties

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Debug	Used to display the statement that is being executed and the values and types of its parameters.
Options	Used to specify the behaviour of a TDADump component.
[SQL](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 MySQL raises some error on TDADump.Restore.
OnRestoreProgress	Occurs to indicate the TDADump.Restore, TDADump.RestoreFromFile, or TDADump.RestoreFromStream method execution

	progress.

Request Support **DAC Forum** **Provide Feedback**

5.7.1.1.2 Properties

Properties of the **TDADump** class.

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

Public

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Options	Used to specify the behaviour of a TDADump component.

Published

Name	Description
Debug	Used to display the statement that is being executed and the values and types of its parameters.
SQL	Used to set or get the dump script.
TableNames	Used to set the names of the tables to dump.

See Also

- TDADump Class

- TDADump Class Members

Request Support **DAC Forum** **Provide Feedback**

5.7.1.1.2.1 Connection Property

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

Class

[TDADump](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

See Also

- [TCustomDAConnection](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.7.1.1.2.2 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

[TDADump](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

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

Note: If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDADataSet.Debug

- TCustomDASQL.Debug

Request Support   DAC Forum   Provide Feedback

5.7.1.1.2.3 Options Property

Used to specify the behaviour of a TDADump component.

Class

TDADump

Syntax

```
property Options: TDADumpOptions;
```

Remarks

Use the Options property to specify the behaviour of a TDADump component.

Descriptions of all options are in the table below.

Option Name	Description
AddDrop	Used to add DROP statements to a script before CREATE statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database

	object names in generated SQL statements.

Request Support   **DAC Forum**   **Provide Feedback**

5.7.1.1.2.4 SQL Property

Used to set or get the dump script.

Class

[TDADump](#)

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to get or set the dump script. The SQL property stores script that is executed by the [Restore](#) method. This property will store the result of [Backup](#) and [BackupQuery](#). At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- [Restore](#)

- [Backup](#)

- [BackupQuery](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.7.1.1.2.5 TableNames Property

Used to set the names of the tables to dump.

Class

[TDADump](#)

Syntax

```
property TableNames: string;
```

Remarks

Use the TableNames property to set the names of the tables to dump. Table names must be separated with semicolons. If the property is empty, the Backup method will dump all available tables.

See Also

- Backup

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3 Methods

Methods of the **TDADump** class.

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

Public

Name	Description
Backup	Dumps database objects to the TDADump.SQL property.
BackupQuery	Dumps the results of a particular query.
BackupToFile	Dumps database objects to the specified file.
BackupToStream	Dumps database objects to the stream.
Restore	Executes a script contained in the SQL property.
RestoreFromFile	Executes a script from a file.
RestoreFromStream	Executes a script received from the stream.

See Also

- TDADump Class

- TDADump Class Members

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.1 Backup Method

Dumps database objects to the SQL property.

Class

TDADump

Syntax

```
procedure Backup;
```

Remarks

Call the Backup method to dump database objects. The result script will be stored in the SQL property.

See Also

- SQL

- Restore

- BackupToFile

- BackupToStream

- BackupQuery

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.2 BackupQuery Method

Dumps the results of a particular query.

Class

TDADump

Syntax

```
procedure BackupQuery(const Query: string);
```

Parameters

Query

Holds a query used for data selection.

Remarks

Call the BackupQuery method to dump the results of a particular query. Query must be a valid select statement. If this query selects data from several tables, only data of the first table in the from list will be dumped.

See Also

- Restore

- Backup

- BackupToFile

- BackupToStream

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.3 BackupToFile Method

Dumps database objects to the specified file.

Class

TDADump

Syntax

```
procedure BackupToFile(const FileName: string; const Query:
string = '');
```

Parameters

FileName

Holds the file name to dump database objects to.

Query

Your query to receive the data for dumping.

Remarks

Call the BackupToFile method to dump database objects to the specified file.

See Also

- RestoreFromStream

- Backup

- BackupToStream

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.4 BackupToStream Method

Dumps database objects to the stream.

Class

TDADump

Syntax

```
procedure BackupToStream(Stream: TStream; const Query: string =
'');
```

Parameters

Stream
  Holds the stream to dump database objects to.

Query
  Your query to receive the data for dumping.

Remarks

Call the BackupToStream method to dump database objects to the stream.

See Also

- RestoreFromStream

- Backup

- BackupToFile

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.5 Restore Method

Executes a script contained in the SQL property.

Class

TDADump

Syntax

```
procedure Restore;
```

Remarks

Call the Restore method to execute a script contained in the SQL property.

See Also

- RestoreFromFile

- RestoreFromStream

- Backup

- SQL

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.6 RestoreFromFile Method

Executes a script from a file.

Class

TDADump

Syntax

```
procedure RestoreFromFile(const FileName: string);
overload;procedure RestoreFromFile(const FileName: string;
Encoding: TEncoding); overload;
```

Parameters

FileName

Holds the file name to execute a script from.

Remarks

Call the RestoreFromFile method to execute a script from the specified file.

See Also

- Restore

- RestoreFromStream

- BackupToFile

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.7 RestoreFromStream Method

Executes a script received from the stream.

Class

TDADump

Syntax

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

Parameters

Stream

Holds a stream to receive a script to be executed.

Remarks

Call the RestoreFromStream method to execute a script received from the stream.

See Also

- Restore

- RestoreFromFile

- BackupToStream

5.7.1.1.4 Events

Events of the **TDADump** class.

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

Published

Name	Description
OnBackupProgress	Occurs to indicate the TDADump.Backup, M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError	Occurs when MySQL 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

5.7.1.1.4.1 OnBackupProgress Event

Occurs to indicate the [Backup](), M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.

Class

[TDADump]()

Syntax

```
property OnBackupProgress: TDABackupProgressEvent;
```

Remarks

The OnBackupProgress event occurs several times during the dumping process of the [Backup](), M:Devart.Dac.TDADump.BackupToFile(System.String), or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution and indicates its progress. ObjectName parameter indicates the name of the currently dumping database object. ObjectNum shows the number of the current database object in the backup queue starting from zero. ObjectCount shows the quantity of database objects to dump. Percent parameter shows the current percentage of the current table data dumped, not the current percentage of the entire dump process.

See Also
- [Backup]()
- [BackupToFile]()
- [BackupToStream]()

© 1997-2025 Devart. All Rights Reserved.   [Request Support]()   [DAC Forum]()   [Provide Feedback]()

5.7.1.1.4.2 OnError Event

Occurs when MySQL raises some error on [Restore]().

Class

[TDADump]()

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when MySQL raises some error on Restore.

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaException.

Note: You should add the DAScript module to the 'uses' list to use the OnError event handler.

Request Support   DAC Forum   Provide Feedback

5.7.1.1.4.3 OnRestoreProgress Event

Occurs to indicate the Restore, RestoreFromFile, or RestoreFromStream method execution progress.

Class

TDADump

Syntax

```
property OnRestoreProgress: TDARestoreProgressEvent;
```

Remarks

The OnRestoreProgress event occurs several times during the dumping process of the Restore, RestoreFromFile, or RestoreFromStream method execution and indicates its progress. The Percent parameter of the OnRestoreProgress event handler indicates the percentage of the whole restore script execution.

See Also

- Restore

- RestoreFromFile

- RestoreFromStream

Request Support   DAC Forum   Provide Feedback

5.7.1.2 TDADumpOptions Class

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

For a list of all members of this type, see TDADumpOptions members.

Unit

DADump

Syntax

```
TDADumpOptions = class(TPersistent);
```

Request Support   DAC Forum   Provide Feedback

5.7.1.2.1 Members

TDADumpOptions class overview.

Properties

Name	Description
AddDrop	Used to add DROP statements to a script before CREATE statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

Request Support   DAC Forum   Provide Feedback

5.7.1.2.2 Properties

Properties of the **TDADumpOptions** class.

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

Published

Name	Description
AddDrop	Used to add DROP statements to a script before CREATE statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

See Also

- TDADumpOptions Class

- TDADumpOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.7.1.2.2.1 AddDrop Property

Used to add DROP statements to a script before CREATE statements.

Class

TDADumpOptions

Syntax

```
property AddDrop: boolean default True;
```

Remarks

Use the AddDrop property to add DROP statements to a script before CREATE statements.

Request Support   DAC Forum   Provide Feedback

5.7.1.2.2.2 CompleteInsert Property

Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.

Class

TDADumpOptions

Syntax

```
property CompleteInsert: boolean default False;
```

Remarks

If the CompleteInsert property is set to True, SQL query will include the field names, for example:

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

If False, it won't include the field names, for example:

```
INSERT INTO dept VALUES ('10', 'ACCOUNTING', 'NEW YORK');
```
Request Support   DAC Forum   Provide Feedback

5.7.1.2.2.3 GenerateHeader Property

Used to add a comment header to a script.

Class

TDADumpOptions

Syntax

```
property GenerateHeader: boolean default True;
```

Remarks

Use the GenerateHeader property to add a comment header to a script. It contains script generation date, DAC version, and some other information.

5.7.1.2.2.4 QuoteNames Property

Used for TDADump to quote all database object names in generated SQL statements.

Class

[TDADumpOptions](#)

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If the QuoteNames property is True, TDADump quotes all database object names in generated SQL statements.

5.7.2   Types

Types in the **DADump** unit.

Types

Name	Description
[TDABackupProgressEvent](#)	This type is used for the [TDADump.OnBackupProgress](#) event.
[TDARestoreProgressEvent](#)	This type is used for the [TDADump.OnRestoreProgress](#) event.

Reserved.

5.7.2.1 TDABackupProgressEvent Procedure Reference

This type is used for the TDADump.OnBackupProgress event.

Unit

DADump

Syntax

```
TDABackupProgressEvent = procedure (Sender: TObject; ObjectName:
string; ObjectNum: integer; ObjectCount: integer; Percent:
integer) of object;
```

Parameters

Sender
   An object that raised the event.

ObjectName
   The name of the currently dumping database object.

ObjectNum
   The number of the current database object in the backup queue starting from zero.

ObjectCount
   The quantity of database objects to dump.

Percent
   The current percentage of the current table data dumped.

Request Support   DAC Forum   Provide Feedback

5.7.2.2 TDARestoreProgressEvent Procedure Reference

This type is used for the TDADump.OnRestoreProgress event.

Unit

DADump

Syntax

```
TDARestoreProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender

  An object that raised the event.

Percent

  The percentage of the whole restore script execution.

Request Support   DAC Forum   Provide Feedback

5.8 DALoader

This unit contains the base class for the TMyLoader component.

Classes

Name	Description
TDAColumn	Represents the attributes for column loading.
TDAColumns	Holds a collection of TDAColumn objects.
TDALoader	This class allows loading external data into database.
TDALoaderOptions	Allows loading external data into database.

Types

Name	Description
TDAPutDataEvent	This type is used for the TDALoader.OnPutData event.
TGetColumnDataEvent	This type is used for the TDALoader.OnGetColumnData event.
TLoaderProgressEvent	This type is used for the TDALoader.OnProgress event.

Request Support   DAC Forum   Provide Feedback

5.8.1 Classes

Classes in the **DALoader** unit.

Classes

Name	Description
[TDAColumn](#)	Represents the attributes for column loading.
[TDAColumns](#)	Holds a collection of [TDAColumn](#) objects.
[TDALoader](#)	This class allows loading external data into database.
[TDALoaderOptions](#)	Allows loading external data into database.

Request Support   **DAC Forum**   **Provide Feedback**

5.8.1.1 TDAColumn Class

Represents the attributes for column loading.

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

Unit

`DALoader`

Syntax

```
TDAColumn = class(TCollectionItem);
```

Remarks

Each [TDALoader](#) uses [TDAColumns](#) to maintain a collection of TDAColumn objects. TDAColumn object represents the attributes for column loading. Every TDAColumn object corresponds to one of the table fields with the same name as its [TDAColumn.Name](#) property.

To create columns at design-time use the column editor of the [TDALoader](#) component.

See Also

- [TDALoader](#)

- TDAColumns

Request Support   **DAC Forum**   **Provide Feedback**

5.8.1.1.1 Members

TDAColumn class overview.

Properties

Name	Description
FieldType	Used to specify the types of values that will be loaded.
Name	Used to specify the field name of loading table.

Request Support   **DAC Forum**   **Provide Feedback**

5.8.1.1.2 Properties

Properties of the **TDAColumn** class.

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

Published

Name	Description
FieldType	Used to specify the types of values that will be loaded.
Name	Used to specify the field name of loading table.

See Also

- TDAColumn Class

- TDAColumn Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.8.1.1.2.1 FieldType Property

Used to specify the types of values that will be loaded.

Class

[TDAColumn](#)

Syntax

```
property FieldType: TFieldType default ftString;
```

Remarks

Use the FieldType property to specify the types of values that will be loaded. Field types for columns may not match data types for the corresponding fields in the database table. [TDALoader](#) will cast data values to the types of their fields.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.8.1.1.2.2 Name Property

Used to specify the field name of loading table.

Class

[TDAColumn](#)

Syntax

```
property Name: string;
```

Remarks

Each TDAColumn corresponds to one field of the loading table. Use the Name property to specify the name of this field.

See Also

* [FieldType](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.8.1.2 TDAColumns Class

Holds a collection of TDAColumn objects.

For a list of all members of this type, see TDAColumns members.

Unit

DALoader

Syntax

```
TDAColumns = class(TOwnedCollection);
```

Remarks

Each TDAColumns holds a collection of TDAColumn objects. TDAColumns maintains an index of the columns in its Items array. The Count property contains the number of columns in the collection. At design-time, use the Columns editor to add, remove, or modify columns.

See Also

- TDALoader

- TDAColumn

Request Support   DAC Forum   Provide Feedback

5.8.1.2.1 Members

TDAColumns class overview.

Properties

Name	Description
Items	Used to access individual columns.

Request Support   DAC Forum   Provide Feedback

5.8.1.2.2 Properties

Properties of the **TDAColumns** class.

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

Public

Name	Description
Items	Used to access individual columns.

See Also

- TDAColumns Class

- TDAColumns Class Members

© 1997-2025
Devart. All Rights
Reserved.   Request Support   DAC Forum   Provide Feedback

5.8.1.2.2.1 Items Property(Indexer)

Used to access individual columns.

Class

TDAColumns

Syntax

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

Parameters

Index
   Holds the Index of TDAColumn to refer to.

Remarks

Use the Items property to access individual columns. The value of the Index parameter corresponds to the Index property of TDAColumn.

See Also

- TDAColumn

Request Support   DAC Forum   Provide Feedback

5.8.1.3   TDALoader Class

This class allows loading external data into database.

For a list of all members of this type, see TDALoader members.

Unit

DALoader

Syntax

```
TDALoader = class(TComponent);
```

Remarks

TDALoader allows loading external data into database. To specify the name of loading table set the TDALoader.TableName property. Use the TDALoader.Columns property to access individual columns. Write the TDALoader.OnGetColumnData or TDALoader.OnPutData event handlers to read external data and pass it to the database. Call the TDALoader.Load method to start loading data.

See Also
- TMyLoader

Request Support   DAC Forum   Provide Feedback

5.8.1.3.1 Members

TDALoader class overview.

Properties

Name	Description
Columns	Used to add a TDAColumn object for each field that will

	be loaded.
[Connection](#)	property. 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.LoadFromData Set](#) method is needed.
[OnPutData](#)	Occurs when putting loading data by rows is needed.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.8.1.3.2 Properties

Properties of the **TDALoader** class.

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

Public

Name	Description
[Columns](#)	Used to add a [TDAColumn](#) object for each field that will be loaded.
[Connection](#)	property. 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](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.8.1.3.2.1 Columns Property

Used to add a [TDAColumn](#) object for each field that will be loaded.

Class

`TDALoader`

Syntax

```
property Columns: TDAColumns stored IsColumnsStored;
```

Remarks

Use the Columns property to add a [TDAColumn](#) object for each field that will be loaded.

See Also

- [TDAColumns](#)

Request Support   **DAC Forum**   **Provide Feedback**

Reserved.

5.8.1.3.2.2 Connection Property

property. 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

Request Support   DAC Forum   Provide Feedback

5.8.1.3.2.3 TableName Property

Used to specify the name of the table to which data will be loaded.

Class

TDALoader

Syntax

```
property TableName: string;
```

Remarks

Set the TableName property to specify the name of the table to which data will be loaded. Add TDAColumn objects to Columns for the fields that are needed to be loaded.

See Also

- TDAColumn

- TCustomDAConnection.GetTableNames

Request Support   DAC Forum   Provide Feedback

5.8.1.3.3 Methods

Methods of the **TDALoader** class.

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

Public

Name	Description
CreateColumns	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName.
Load	Starts loading data.
LoadFromDataSet	Loads data from the specified dataset.
PutColumnData	Overloaded. Puts the value of individual columns.

See Also

- TDALoader Class

- TDALoader Class Members

Request Support   DAC Forum   Provide Feedback

5.8.1.3.3.1 CreateColumns Method

Creates TDAColumn objects for all fields of the table with the same name as TableName.

Class

`TDALoader`

Syntax

```
procedure CreateColumns;
```

Remarks

Call the CreateColumns method to create TDAColumn objects for all fields of the table with the same name as TableName. If columns were created before, they will be recreated. You can call CreateColumns from the component popup menu at design-time. After you can customize column loading by setting properties of TDAColumn objects.

See Also

- TDAColumn

- TableName

Request Support   DAC Forum   Provide Feedback

5.8.1.3.3.2 Load Method

Starts loading data.

Class

TDALoader

Syntax

```
procedure Load; virtual;
```

Remarks

Call the Load method to start loading data. At first it is necessary to create columns and write one of the OnPutData or OnGetColumnData event handlers.

See Also

- OnGetColumnData

- OnPutData

Request Support   DAC Forum   Provide Feedback

5.8.1.3.3.3 LoadFromDataSet Method

Loads data from the specified dataset.

Class

[TDALoader](#)

Syntax

```
procedure LoadFromDataSet(DataSet: TDataSet);
```

Parameters

DataSet
  Holds the dataset to load data from.

Remarks

Call the LoadFromDataSet method to load data from the specified dataset. There is no need to create columns and write event handlers for [OnPutData](#) and [OnGetColumnData](#) before calling this method.

© 1997-2025
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.8.1.3.3.4 PutColumnData Method

Puts the value of individual columns.

Class

[TDALoader](#)

Overload List

Name	Description
[PutColumnData(Col: integer; Row: integer; **const** Value: variant)](#)	Puts the value of individual columns by the column index.
[PutColumnData(**const** ColName: **string**; Row: integer; **const** Value: variant)](#)	Puts the value of individual columns by the column name.

© 1997-2025
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Puts the value of individual columns by the column index.

Class

[TDALoader](#)

Syntax

```
procedure PutColumnData(Col: integer; Row: integer; const Value:
variant); overload; virtual;
```

Parameters

Col
  Holds the index of a loading column. The first column has index 0.

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

Value
  Holds the column value.

Remarks

Call the PutColumnData method to put the value of individual columns. The Col parameter indicates the index of loading column. The first column has index 0. The Row parameter indicates the number of the loading row. Row starts from 1.

This overloaded method works faster because it searches the right index by its index, not by the index name.

The value of a column should be assigned to the Value parameter.

See Also

- [TDALoader.OnPutData](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Puts the value of individual columns by the column name.

Class

[TDALoader](#)

Syntax

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

Parameters

ColName
  Hods the name of a loading column.

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

Value
  Holds the column value.

Request Support   **DAC Forum**   **Provide Feedback**

5.8.1.3.4 Events

Events of the **TDALoader** class.

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

Public

Name	Description
OnGetColumnData	Occurs when it is needed to put column values.
OnProgress	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData	Occurs when putting loading data by rows is needed.

See Also

- TDALoader Class

- TDALoader Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.8.1.3.4.1 OnGetColumnData Event

Occurs when it is needed to put column values.

Class

TDALoader

Syntax

```
property OnGetColumnData: TGetColumnDataEvent;
```

Remarks

Write the OnGetColumnData event handler to put column values. TDALoader calls the OnGetColumnData event handler for each column in the loop. Column points to a TDAColumn object that corresponds to the current loading column. Use its Name or Index property to identify what column is loading. The Row parameter indicates the current loading record. TDALoader increments the Row parameter when all the columns of the current record are loaded. The first row is 1. Set EOF to True to stop data loading. Fill the Value parameter by column values. To start loading call the Load method.

Another way to load data is using the OnPutData event.

Example

This handler loads 1000 rows.

```
procedure TfmMain.GetColumnData(Sender: TObject;
   Column: TDAColumn; Row: Integer; var Value: Variant;
   var EOF: Boolean);
begin
   if Row <= 1000 then begin
      case Column.Index of
         0: Value := Row;
         1: Value := Random(100);
         2: Value := Random*100;
         3: Value := 'abc01234567890123456789';
         4: Value := Date;
      else
         Value := Null;
      end;
   end
   else
      EOF := True;
end;
```

See Also

- OnPutData

- Load

　　Request Support　　DAC Forum　　Provide Feedback

5.8.1.3.4.2 OnProgress Event

Occurs if handling data loading progress of the LoadFromDataSet method is needed.

Class

`TDALoader`

Syntax

```
property OnProgress: TLoaderProgressEvent;
```

Remarks

Add a handler to this event if you want to handle data loading progress of the LoadFromDataSet method.

See Also

- LoadFromDataSet

　　Request Support　　DAC Forum　　Provide Feedback

5.8.1.3.4.3 OnPutData Event

Occurs when putting loading data by rows is needed.

Class

`TDALoader`

Syntax

```
property OnPutData: TDAPutDataEvent;
```

Remarks

Write the OnPutData event handler to put loading data by rows.

Note that rows should be loaded from the first in the ascending order. TMyLoader will flush data to MySQL when it is needed (see TMyLoader.RowsPerQuery).

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

Request Support   DAC Forum   Provide Feedback

5.8.1.4   TDALoaderOptions Class

Allows loading external data into database.

For a list of all members of this type, see TDALoaderOptions members.

Unit

DALoader

Syntax

```
TDALoaderOptions = class(TPersistent);
```

Request Support **DAC Forum** **Provide Feedback**

5.8.1.4.1 Members

TDALoaderOptions class overview.

Properties

Name	Description
UseBlankValues	Forces MyDAC to fill the buffer with null values after loading a row to the database.

Request Support **DAC Forum** **Provide Feedback**

5.8.1.4.2 Properties

Properties of the **TDALoaderOptions** class.

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

Public

Name	Description
UseBlankValues	Forces MyDAC to fill the buffer with null values after loading a row to the database.

See Also

- TDALoaderOptions Class

- TDALoaderOptions Class Members

Request Support **DAC Forum** **Provide Feedback**

5.8.1.4.2.1 UseBlankValues Property

Forces MyDAC to fill the buffer with null values after loading a row to the database.

Class

TDALoaderOptions

Syntax

```
property UseBlankValues: boolean default True;
```

Remarks

Used to force MyDAC to fill the buffer with null values after loading a row to the database.

Request Support   DAC Forum   Provide Feedback

5.8.2   Types

Types in the **DALoader** unit.

Types

Name	Description
TDAPutDataEvent	This type is used for the TDALoader.OnPutData event.
TGetColumnDataEvent	This type is used for the TDALoader.OnGetColumnData event.
TLoaderProgressEvent	This type is used for the TDALoader.OnProgress event.

Request Support   DAC Forum   Provide Feedback

5.8.2.1 TDAPutDataEvent Procedure Reference

This type is used for the TDALoader.OnPutData event.

Unit

DALoader

Syntax

```
TDAPutDataEvent = procedure (Sender: TDALoader) of object;
```

Parameters

Sender
   An object that raised the event.

Request Support   DAC Forum   Provide Feedback

5.8.2.2 TGetColumnDataEvent Procedure Reference

This type is used for the TDALoader.OnGetColumnData event.

Unit

DALoader

Syntax

```
TGetColumnDataEvent = procedure (Sender: TObject; Column:
TDAColumn; Row: integer; var Value: variant; var IsEOF: boolean)
of object;
```

Parameters

Sender
   An object that raised the event.

Column
   Points to TDAColumn object that corresponds to the current loading column.

Row
   Indicates the current loading record.

Value
   Holds column values.

IsEOF
   True, if data loading needs to be stopped.

5.8.2.3 TLoaderProgressEvent Procedure Reference

This type is used for the TDALoader.OnProgress event.

Unit

DALoader

Syntax

```
TLoaderProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender
  An object that raised the event.

Percent
  Percentage of the load operation progress.

5.9 DAScript

This unit contains the base class for the TMyScript component.

Classes

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

Types

Name	Description
[TAfterStatementExecuteEvent](#)	This type is used for the [TDAScript.AfterExecute](#) event.
[TBeforeStatementExecuteEvent](#)	This type is used for the [TDAScript.BeforeExecute](#) event.
[TOnErrorEvent](#)	This type is used for the [TDAScript.OnError](#) event.

Enumerations

Name	Description
[TErrorAction](#)	Indicates the action to take when the OnError handler exits.

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1 Classes

Classes in the **DAScript** unit.

Classes

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

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1 TDAScript Class

Makes it possible to execute several SQL statements one by one.

For a list of all members of this type, see TDAScript members.

Unit

DAScript

Syntax

```
TDAScript = class(TComponent);
```

Remarks

Often it is necessary to execute several SQL statements one by one. This can be performed using a lot of components such as TCustomDASQL descendants. Usually it isn't the best solution. With only one TDAScript descedant component you can execute several SQL statements as one. This sequence of statements is called script. To separate single statements use semicolon (;) or slash (/) and for statements that can contain semicolon, only slash. Note that slash must be the first character in line.

Errors that occur during execution can be processed in the TDAScript.OnError event handler. By default, on error TDAScript shows exception and continues execution.

See Also

- TCustomDASQL

Request Support   DAC Forum   Provide Feedback

5.9.1.1.1 Members

TDAScript class overview.

Properties

Name	Description
Connection	Used to specify the connection in which the script will be executed.

[DataSet](#)	Refers to a dataset that holds the result set of query execution.
[Debug](#)	Used to display the script execution and all its parameter values.
[Delimiter](#)	Used to set the delimiter string that separates script statements.
[EndLine](#)	Used to get the current statement last line number in a script.
[EndOffset](#)	Used to get the offset in the last line of the current statement.
[EndPos](#)	Used to get the end position of the current statement.
[Macros](#)	Used to change SQL script text in design- or run-time easily.
[SQL](#)	Used to get or set script text.
[StartLine](#)	Used to get the current statement start line number in a script.
[StartOffset](#)	Used to get the offset in the first line of the current statement.
[StartPos](#)	Used to get the start position of the current statement in a script.
[Statements](#)	Contains a list of statements obtained from the SQL property.

Methods

Name	Description
[BreakExec](#)	Stops script execution.
[ErrorOffset](#)	Used to get the offset of the statement if the Execute method raised an exception.
[Execute](#)	Executes a script.

[ExecuteFile](#)	Executes SQL statements contained in a file.
[ExecuteNext](#)	Executes the next statement in the script and then stops.
[ExecuteStream](#)	Executes SQL statements contained in a stream object.
[FindMacro](#)	Finds a macro with the specified name.
[MacroByName](#)	Finds a macro with the specified name.

Events

Name	Description
[AfterExecute](#)	Occurs after a SQL script execution.
[BeforeExecute](#)	Occurs when taking a specific action before executing the current SQL statement is needed.
[OnError](#)	Occurs when MySQL raises an error.

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1.2 Properties

Properties of the **TDAScript** class.

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

Public

Name	Description
[Connection](#)	Used to specify the connection in which the script will be executed.
[DataSet](#)	Refers to a dataset that holds the result set of query execution.

EndLine	Used to get the current statement last line number in a script.
EndOffset	Used to get the offset in the last line of the current statement.
EndPos	Used to get the end position of the current statement.
StartLine	Used to get the current statement start line number in a script.
StartOffset	Used to get the offset in the first line of the current statement.
StartPos	Used to get the start position of the current statement in a script.
Statements	Contains a list of statements obtained from the SQL property.

Published

Name	Description
Debug	Used to display the script execution and all its parameter values.
Delimiter	Used to set the delimiter string that separates script statements.
Macros	Used to change SQL script text in design- or run-time easily.
SQL	Used to get or set script text.

See Also

- TDAScript Class

- TDAScript Class Members

Request Support   DAC Forum   Provide Feedback

5.9.1.1.2.1 Connection Property

Used to specify the connection in which the script will be executed.

Class

TDAScript

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify the connection in which the script will be executed. If Connection is not connected, the Execute method calls the Connect method of Connection.

Set at design-time by selecting from the list of provided TCustomDAConnection objects.

At run-time, set the Connection property to reference an existing TCustomDAConnection object.

See Also

• TCustomDAConnection

Request Support   DAC Forum   Provide Feedback

5.9.1.1.2.2 DataSet Property

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

Class

TDAScript

Syntax

```
property DataSet: TCustomDADataSet;
```

Remarks

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

See Also

- [ExecuteNext](#)

- [Execute](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1.2.3 Debug Property

Used to display the script execution and all its parameter values.

Class

`TDAScript`

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

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

Note: If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1.2.4 Delimiter Property

Used to set the delimiter string that separates script statements.

Class

`TDAScript`

Syntax

```
property Delimiter: string stored IsDelimiterStored;
```

Remarks

Use the Delimiter property to set the delimiter string that separates script statements. By default it is semicolon (;). You can use slash (/) to separate statements that can contain semicolon if the Delimiter property's default value is semicolon. Note that slash must be the first character in line.

Request Support   DAC Forum   Provide Feedback

5.9.1.1.2.5 EndLine Property

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

Class

TDAScript

Syntax

```
property EndLine: Int64;
```

Remarks

Use the EndLine property to get the current statement last line number in a script.

Request Support   DAC Forum   Provide Feedback

5.9.1.1.2.6 EndOffset Property

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

Class

TDAScript

Syntax

```
property EndOffset: Int64;
```

Remarks

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

5.9.1.1.2.7 EndPos Property

Used to get the end position of the current statement.

Class

`TDAScript`

Syntax

```
property EndPos: Int64;
```

Remarks

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

5.9.1.1.2.8 Macros Property

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

Class

`TDAScript`

Syntax

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

Remarks

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

run-time. In time of opening query macro is replaced by its value.

See Also

- TMacro

- MacroByName

Request Support   DAC Forum   Provide Feedback

5.9.1.1.2.9 SQL Property

Used to get or set script text.

Class

TDAScript

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to get or set script text.

Request Support   DAC Forum   Provide Feedback

5.9.1.1.2.10 StartLine Property

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

Class

TDAScript

Syntax

```
property StartLine: Int64;
```

Remarks

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.1.2.11 StartOffset Property

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

Class

[TDAScript](#)

Syntax

```
property StartOffset: Int64;
```

Remarks

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.1.2.12 StartPos Property

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

Class

[TDAScript](#)

Syntax

```
property StartPos: Int64;
```

Remarks

Use the StartPos property to get the start position of the current statement (the position of the

first statement character) in a script.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.1.2.13 Statements Property

Contains a list of statements obtained from the SQL property.

Class

TDAScript

Syntax

```
property Statements: TDAStatements;
```

Remarks

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

For example, consider the following script:

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

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

Example

You can use the Statements property in the following way:

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

See Also

- TDAStatements

    Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1.3 Methods

Methods of the **TDAScript** class.

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

Public

Name	Description
BreakExec	Stops script execution.
ErrorOffset	Used to get the offset of the statement if the Execute method raised an exception.
Execute	Executes a script.
ExecuteFile	Executes SQL statements contained in a file.
ExecuteNext	Executes the next statement in the script and then stops.
ExecuteStream	Executes SQL statements contained in a stream object.
FindMacro	Finds a macro with the specified name.
MacroByName	Finds a macro with the specified name.

See Also

- TDAScript Class

- TDAScript Class Members

    Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1.3.1 BreakExec Method

Stops script execution.

Class

TDAScript

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

Request Support   DAC Forum   Provide Feedback

5.9.1.1.3.2 ErrorOffset Method

Used to get the offset of the statement if the Execute method raised an exception.

Class

TDAScript

Syntax

```
function ErrorOffset: Int64;
```

Return Value

offset of an error.

Remarks

Call the ErrorOffset method to get the offset of the statement if the Execute method raised an exception.

See Also

- OnError

Request Support   DAC Forum   Provide Feedback

5.9.1.1.3.3 Execute Method

Executes a script.

Class

[TDAScript](#)

Syntax

```
procedure Execute; virtual;
```

Remarks

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

See Also

- [ExecuteNext](#)

- [OnError](#)

- [ErrorOffset](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.1.3.4 ExecuteFile Method

Executes SQL statements contained in a file.

Class

[TDAScript](#)

Syntax

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

Parameters

FileName
  Holds the file name.

Remarks

Call the ExecuteFile method to execute SQL statements contained in a file. Script doesn't load full content into memory. Reading and execution is performed by blocks of 64k size. Therefore, it is optimal to use it for big files.

Request Support   DAC Forum   Provide Feedback

5.9.1.1.3.5 ExecuteNext Method

Executes the next statement in the script and then stops.

Class

`TDAScript`

Syntax

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

Return Value

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

Remarks

Use the ExecuteNext method to execute the next statement in the script statement and stop.

If MySQL raises an error, the OnError event occurs.

See Also
* Execute

* OnError

* ErrorOffset

Request Support   DAC Forum   Provide Feedback

5.9.1.1.3.6 ExecuteStream Method

Executes SQL statements contained in a stream object.

Class

`TDAScript`

Syntax

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

Parameters

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

Remarks

Call the ExecuteStream method to execute SQL statements contained in a stream object.

Reading from the stream and execution is performed by blocks of 64k size.

Request Support   DAC Forum   Provide Feedback

5.9.1.1.3.7 FindMacro Method

Finds a macro with the specified name.

Class

TDAScript

Syntax

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

Parameters

Name
  Holds the name of a macro to search for.

Return Value
  TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

See Also

- TMacro

- Macros

- MacroByName

Request Support   DAC Forum   Provide Feedback

5.9.1.1.3.8 MacroByName Method

Finds a macro with the specified name.

Class

TDAScript

Syntax

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

Parameters

Name
  Holds the name of a macro to search for.

Return Value
  TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

See Also

- TMacro

- Macros

- FindMacro

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1.4 Events

Events of the **TDAScript** class.

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

Published

Name	Description
AfterExecute	Occurs after a SQL script execution.
BeforeExecute	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError	Occurs when MySQL raises an error.

See Also

- TDAScript Class

- TDAScript Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.1.4.1 AfterExecute Event

Occurs after a SQL script execution.

Class

TDAScript

Syntax

```
property AfterExecute: TAfterStatementExecuteEvent;
```

Remarks

Occurs after a SQL script has been executed.

See Also

- Execute

Request Support   DAC Forum   Provide Feedback

5.9.1.1.4.2 BeforeExecute Event

Occurs when taking a specific action before executing the current SQL statement is needed.

Class

TDAScript

Syntax

```
property BeforeExecute: TBeforeStatementExecuteEvent;
```

Remarks

Write the BeforeExecute event handler to take specific action before executing the current SQL statement. SQL holds text of the current SQL statement. Write SQL to change the statement that will be executed. Set Omit to True to skip statement execution.

Request Support   DAC Forum   Provide Feedback

5.9.1.1.4.3 OnError Event

Occurs when MySQL raises an error.

Class

TDAScript

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

Occurs when MySQL 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

Request Support   DAC Forum   Provide Feedback

5.9.1.2   TDAStatement Class

This class has attributes and methods for controlling single SQL statement of a script.

For a list of all members of this type, see TDAStatement members.

Unit

DAScript

Syntax

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

Remarks

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

See Also
- TDAScript

- TDAStatements

Request Support   DAC Forum   Provide Feedback

5.9.1.2.1 Members

TDAStatement class overview.

Properties

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

Methods

Name	Description
Execute	Executes a statement.

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.2.2 Properties

Properties of the **TDAStatement** class.

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

Public

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

See Also

- [TDAStatement Class](#)

- [TDAStatement Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.2.2.1 EndLine Property

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

Class

`TDAStatement`

Syntax

```
property EndLine: integer;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.9.1.2.2.2 EndOffset Property

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

Class

TDAStatement

Syntax

```
property EndOffset: integer;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.9.1.2.2.3 EndPos Property

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

Class

TDAStatement

Syntax

```
property EndPos: integer;
```

Remarks

Use the EndPos property to get the end position of the statement (the position of the last

character in the statement) in a script.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.2.2.4 Omit Property

Used to avoid execution of a statement.

Class

[TDAStatement](#)

Syntax

```
property Omit: boolean;
```

Remarks

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.2.2.5 Params Property

Contains parasmeters for an SQL statement.

Class

[TDAStatement](#)

Syntax

```
property Params: TDAParams;
```

Remarks

Contains parameters for an SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically. Params is a zero-based array of parameter records. Index specifies the array element to access.

See Also

- TDAParam

Request Support   DAC Forum   Provide Feedback

5.9.1.2.2.6 Script Property

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

Class

TDAStatement

Syntax

```
property Script: TDAScript;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.9.1.2.2.7 SQL Property

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

Class

TDAStatement

Syntax

```
property SQL: string;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.9.1.2.2.8 StartLine Property

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

Class

TDAStatement

Syntax

```
property StartLine: integer;
```

Remarks

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

© 1997-2025
Devart. All Rights
Reserved.   Request Support   DAC Forum   Provide Feedback

5.9.1.2.2.9 StartOffset Property

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

Class

TDAStatement

Syntax

```
property StartOffset: integer;
```

Remarks

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

© 1997-2025
Devart. All Rights
Reserved.   Request Support   DAC Forum   Provide Feedback

5.9.1.2.2.10 StartPos Property

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

Class

TDAStatement

Syntax

```
property StartPos: integer;
```

Remarks

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

5.9.1.2.3 Methods

Methods of the **TDAStatement** class.

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

Public

Name	Description
Execute	Executes a statement.

See Also

- TDAStatement Class

- TDAStatement Class Members

5.9.1.2.3.1 Execute Method

Executes a statement.

Class

TDAStatement

Syntax

```
procedure Execute;
```

Remarks

Use the Execute method to execute a statement.

Request Support   DAC Forum   Provide Feedback

5.9.1.3   TDAStatements Class

Holds a collection of TDAStatement objects.

For a list of all members of this type, see TDAStatements members.

Unit

DAScript

Syntax

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

Remarks

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

See Also

- TDAScript

- TDAStatement

Request Support   DAC Forum   Provide Feedback

5.9.1.3.1 Members

TDAStatements class overview.

Properties

Name	Description
Items	Used to access separate script statements.

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.3.2 Properties

Properties of the **TDAStatements** class.

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

Public

Name	Description
Items	Used to access separate script statements.

See Also

- TDAStatements Class

- TDAStatements Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.9.1.3.2.1 Items Property(Indexer)

Used to access separate script statements.

Class

TDAStatements

Syntax

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

Parameters

Index
   Holds the index value.

Remarks

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

See Also

- TDAStatement

Request Support   DAC Forum   Provide Feedback

5.9.2   Types

Types in the **DAScript** unit.

Types

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

Request Support   DAC Forum   Provide Feedback

5.9.2.1   TAfterStatementExecuteEvent Procedure Reference

This type is used for the TDAScript.AfterExecute event.

Unit

DAScript

Syntax

```
TAfterStatementExecuteEvent = procedure (Sender: TObject; SQL:
string) of object;
```

Parameters

Sender

An object that raised the event.

SQL

Holds the passed SQL statement.

Request Support   DAC Forum   Provide Feedback

5.9.2.2   TBeforeStatementExecuteEvent Procedure Reference

This type is used for the TDAScript.BeforeExecute event.

Unit

DAScript

Syntax

```
TBeforeStatementExecuteEvent = procedure (Sender: TObject; var
SQL: string; var Omit: boolean) of object;
```

Parameters

Sender

An object that raised the event.

SQL

Holds the passed SQL statement.

Omit

True, if the statement execution should be skipped.

Request Support   DAC Forum   Provide Feedback

5.9.2.3   TOnErrorEvent Procedure Reference

This type is used for the TDAScript.OnError event.

Unit

DAScript

Syntax

```
TOnErrorEvent = procedure (Sender: TObject; E: Exception; SQL:
string; var Action: TErrorAction) of object;
```

Parameters

Sender

  An object that raised the event.

E

  The error code.

SQL

  Holds the passed SQL statement.

Action

  The action to take when the OnError handler exits.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.3 Enumerations

Enumerations in the **DAScript** unit.

Enumerations

Name	Description
[TErrorAction](#)	Indicates the action to take when the OnError handler exits.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.3.1 TErrorAction Enumeration

Indicates the action to take when the OnError handler exits.

Unit

[DAScript](#)

Syntax

```
TErrorAction = (eaAbort, eaFail, eaException, eaContinue);
```

Values

Value	Meaning
eaAbort	Abort execution without displaying an error message.
eaContinue	Continue execution.
eaException	In Delphi 6 and higher exception is handled by the Application.HandleException method.
eaFail	Abort execution and display an error message.

Request Support   DAC Forum   Provide Feedback

5.10   DASQLMonitor

This unit contains the base class for the TMySQLMonitor component.

Classes

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

Types

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

Enumerations

Name	Description
TDATraceFlag	Use TraceFlags to specify which database operations

	the monitor should track in an application at runtime.
[TMonitorOption](#)	Used to define where information from SQLMonitor will be dispalyed.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.10.1 Classes

Classes in the **DASQLMonitor** unit.

Classes

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

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.10.1.1 TCustomDASQLMonitor Class

A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.

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

Unit

[DASQLMonitor](#)

Syntax

```
TCustomDASQLMonitor = class(TComponent);
```

Remarks

TCustomDASQLMonitor is a base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively. TCustomDASQLMonitor provides two ways of displaying debug information. It monitors either by dialog window or by Borland's proprietary SQL Monitor. Furthermore to receive debug information use the [TCustomDASQLMonitor.OnSQL](#) event.

In applications use descendants of TCustomDASQLMonitor.

Request Support   **DAC Forum**   **Provide Feedback**

5.10.1.1.1 Members

TCustomDASQLMonitor class overview.

Properties

Name	Description
[Active](#)	Used to activate monitoring of SQL.
[DBMonitorOptions](#)	Used to set options for dbMonitor.
[Options](#)	Used to include the desired properties for TCustomDASQLMonitor.
[TraceFlags](#)	Used to specify which database operations the monitor should track in an application at runtime.

Events

Name	Description
[OnSQL](#)	Occurs when tracing of SQL activity on database components is needed.

Request Support   **DAC Forum**   **Provide Feedback**

5.10.1.1.2 Properties

Properties of the **TCustomDASQLMonitor** class.

For a complete list of the **TCustomDASQLMonitor** class members, see the

TCustomDASQLMonitor Members topic.

Public

Name	Description
Active	Used to activate monitoring of SQL.
DBMonitorOptions	Used to set options for dbMonitor.
Options	Used to include the desired properties for TCustomDASQLMonitor.
TraceFlags	Used to specify which database operations the monitor should track in an application at runtime.

See Also

- TCustomDASQLMonitor Class

- TCustomDASQLMonitor Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.1.2.1 Active Property

Used to activate monitoring of SQL.

Class

TCustomDASQLMonitor

Syntax

```
property Active: boolean default True;
```

Remarks

Set the Active property to True to activate monitoring of SQL.

See Also

- OnSQL

Request Support DAC Forum Provide Feedback

5.10.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

Class

`TCustomDASQLMonitor`

Syntax

```
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

Request Support DAC Forum Provide Feedback

5.10.1.1.2.3 Options Property

Used to include the desired properties for TCustomDASQLMonitor.

Class

`TCustomDASQLMonitor`

Syntax

```
property Options: TMonitorOptions default [moDialog,
moSQLMonitor, moDBMonitor, moCustom];
```

Remarks

Set Options to include the desired properties for TCustomDASQLMonitor.

See Also

- OnSQL

Request Support   **DAC Forum**   **Provide Feedback**

5.10.1.1.2.4 TraceFlags Property

Used to specify which database operations the monitor should track in an application at runtime.

Class

`TCustomDASQLMonitor`

Syntax

```
property TraceFlags: TDATraceFlags default [tfQPrepare,
tfQExecute, tfError, tfConnect, tfTransact, tfParams, tfMisc];
```

Remarks

Use the TraceFlags property to specify which database operations the monitor should track in an application at runtime.

See Also

- OnSQL

Request Support   **DAC Forum**   **Provide Feedback**

5.10.1.1.3 Events

Events of the **TCustomDASQLMonitor** class.

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

Public

Name	Description

[OnSQL](#)	Occurs when tracing of SQL activity on database components is needed.

See Also

- [TCustomDASQLMonitor Class](#)

- [TCustomDASQLMonitor Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.10.1.1.3.1 OnSQL Event

Occurs when tracing of SQL activity on database components is needed.

Class

`TCustomDASQLMonitor`

Syntax

```
property OnSQL: TOnSQLEvent;
```

Remarks

Write the OnSQL event handler to let an application trace SQL activity on database components. The Text parameter holds the detected SQL statement. Use the Flag parameter to make selective processing of SQL in the handler body.

See Also

- [TraceFlags](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.10.1.2 TDBMonitorOptions Class

This class holds options for dbMonitor.

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

Unit

[DASQLMonitor](#)

Syntax

```
TDBMonitorOptions = class(TPersistent);
```

Request Support   **DAC Forum**   **Provide Feedback**

5.10.1.2.1 Members

TDBMonitorOptions class overview.

Properties

Name	Description
[Host](#)	Used to set the host name or IP address of the computer where dbMonitor application runs.
[Port](#)	Used to set the port number for connecting to dbMonitor.
[ReconnectTimeout](#)	Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.
[SendTimeout](#)	Used to set timeout for sending events to dbMonitor.

Request Support   **DAC Forum**   **Provide Feedback**

5.10.1.2.2 Properties

Properties of the **TDBMonitorOptions** class.

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

Published

Name	Description
Host	Used to set the host name or IP address of the computer where dbMonitor application runs.
Port	Used to set the port number for connecting to dbMonitor.
ReconnectTimeout	Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.
SendTimeout	Used to set timeout for sending events to dbMonitor.

See Also

- TDBMonitorOptions Class

- TDBMonitorOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.2.2.1 Host Property

Used to set the host name or IP address of the computer where dbMonitor application runs.

Class

TDBMonitorOptions

Syntax

```
property Host: string;
```

Remarks

Use the Host property to set the host name or IP address of the computer where dbMonitor application runs.

dbMonitor supports remote monitoring. You can run dbMonitor on a different computer than monitored application runs. In this case you need to set the Host property to the corresponding computer name.

5.10.1.2.2.2 Port Property

Used to set the port number for connecting to dbMonitor.

Class

TDBMonitorOptions

Syntax

```
property Port: integer default DBMonitorPort;
```

Remarks

Use the Port property to set the port number for connecting to dbMonitor.

5.10.1.2.2.3 ReconnectTimeout Property

Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.

Class

TDBMonitorOptions

Syntax

```
property ReconnectTimeout: integer default
DefaultReconnectTimeout;
```

Remarks

Use the ReconnectTimeout property to set the minimum time (in milliseconds) that should be spent before allowing reconnecting to dbMonitor. If an error occurs when the component sends an event to dbMonitor (dbMonitor is not running), next events are ignored and the component does not restore the connection until ReconnectTimeout is over.

5.10.1.2.2.4 SendTimeout Property

Used to set timeout for sending events to dbMonitor.

Class

`TDBMonitorOptions`

Syntax

```
property SendTimeout: integer default DefaultSendTimeout;
```

Remarks

Use the SendTimeout property to set timeout (in milliseconds) for sending events to dbMonitor. If dbMonitor does not respond in the specified timeout, event is ignored.

Request Support   DAC Forum   Provide Feedback

5.10.2 Types

Types in the **DASQLMonitor** unit.

Types

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

Request Support   DAC Forum   Provide Feedback

5.10.2.1 TDATraceFlags Set

Represents the set of TDATraceFlag.

Unit

DASQLMonitor

Syntax

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

Request Support   DAC Forum   Provide Feedback

5.10.2.2 TMonitorOptions Set

Represents the set of TMonitorOption.

Unit

DASQLMonitor

Syntax

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

Request Support   DAC Forum   Provide Feedback

5.10.2.3 TOnSQLEvent Procedure Reference

This type is used for the TCustomDASQLMonitor.OnSQL event.

Unit

DASQLMonitor

Syntax

```
TOnSQLEvent = procedure (Sender: TObject; Text: string; Flag:
TDATraceFlag) of object;
```

Parameters

Sender

An object that raised the event.

Text

Holds the detected SQL statement.

Flag

Use the Flag parameter to make selective processing of SQL in the handler body.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.10.3 Enumerations

Enumerations in the **DASQLMonitor** unit.

Enumerations

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.10.3.1 TDATraceFlag Enumeration

Use TraceFlags to specify which database operations the monitor should track in an application at runtime.

Unit

[DASQLMonitor](#)

Syntax

```
TDATraceFlag = (tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt,
tfConnect, tfTransact, tfBlob, tfService, tfMisc, tfParams,
tfObjDestroy, tfPool);
```

Values

Value	Meaning
tfBlob	This option is declared for future use.
tfConnect	Establishing a connection.
tfError	Errors of query execution.
tfMisc	If this flag is set, then just before sending a query to the server, OnSQL event is called additionally. The difference from usual call is that the query is already completely decoded, i.e. parameters are quoted and included into the text of the query. If to use MySQL 4.1 protocol with preparing, a value of this flag will be ignored.
tfObjDestroy	Destroying of components.
tfParams	Representing parameter values for tfQPrepare and tfQExecute.
tfPool	Connection pool operations.
tfQExecute	Execution of the queries.
tfQFetch	This option is declared for future use.
tfQPrepare	Queries preparation.
tfService	This option is declared for future use.
tfStmt	This option is declared for future use.
tfTransact	Processing transactions.

Request Support   **DAC Forum**   **Provide Feedback**

5.10.3.2 TMonitorOption Enumeration

Used to define where information from SQLMonitor will be dispalyed.

Unit

`DASQLMonitor`

Syntax

```
TMonitorOption = (moDialog, moSQLMonitor, moDBMonitor, moCustom,
moHandled);
```

Values

Value	Meaning
moCustom	Monitoring of SQL for individual components is allowed. Set

	Debug properties in SQL-related components to True to let TCustomDASQLMonitor instance to monitor their behavior. Has effect when moDialog is included.
moDBMonitor	Debug information is displayed in DBMonitor.
moDialog	Debug information is displayed in debug window.
moHandled	Component handle is included into the event description string.
moSQLMonitor	Debug information is displayed in Borland SQL Monitor.

Request Support   DAC Forum   Provide Feedback

5.11   DBAccess

This unit contains base classes for most of the components.

Classes

Name	Description
EDAError	A base class for exceptions that are raised when an error occurs on the server side.
TCRDataSource	Provides an interface between a DAC dataset components and data-aware controls on a form.
TCustomConnectDialog	A base class for the connect dialog components.
TCustomDAConnection	A base class for components used to establish connections.
TCustomDADataSet	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
TCustomDASQL	A base class for components executing SQL statements that do not return result sets.
TCustomDAUpdateSQL	A base class for components that provide DML statements for more flexible control over data

	modifications.
[TDACondition](#)	Represents a condition from the [TDAConditions](#) list.
[TDAConditions](#)	Holds a collection of [TDACondition](#) objects.
[TDAConnectionOptions](#)	This class allows setting up the behaviour of the TDAConnection class.
[TDAConnectionSSLOptions](#)	This class is used to set up the SSL options.
[TDADataSetOptions](#)	This class allows setting up the behaviour of the TDADataSet class.
[TDAEncryption](#)	Used to specify the options of the data encryption in a dataset.
[TDAMapRule](#)	Class that formes rules for Data Type Mapping.
[TDAMapRules](#)	Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.
[TDAMetaData](#)	A class for retrieving metainformation of the specified database objects in the form of dataset.
[TDAParam](#)	A class that forms objects to represent the values of the [parameters set](#).
[TDAParams](#)	This class is used to manage a list of TDAParam objects for an object that uses field parameters.
[TDATransaction](#)	A base class that implements functionality for controlling transactions.
[TMacro](#)	Object that represents the value of a macro.
[TMacros](#)	Controls a list of TMacro objects for the [TCustomDASQL.Macros](#) or [TCustomDADataSet](#) components.

TPoolingOptions	This class allows setting up the behaviour of the connection pool.
TSmartFetchOptions	Smart fetch options are used to set up the behavior of the SmartFetch mode.

Types

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecute events.
TAfterFetchEvent	This type is used for the TCustomDADataSet.AfterFetch event.
TBeforeFetchEvent	This type is used for the TCustomDADataSet.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDAConnection.OnConnectionLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDAConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption.
TUpdateExecuteEvent	This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.

Enumerations

Name	Description
TLabelSet	Sets the languauge of labels in the connect dialog.

[TLockMode](#)	Specifies the lock mode.
[TRefreshOption](#)	Indicates when the editing record will be refreshed.
[TRetryMode](#)	Specifies the application behavior when connection is lost.

Variables

Name	Description
[BaseSQLOldBehavior](#)	After assigning SQL text and modifying it by [AddWhere](#), [DeleteWhere](#), and [SetOrderBy](#), all subsequent changes of the SQL property will not be reflected in the BaseSQL property.
[ChangeCursor](#)	When set to True allows data access components to change screen cursor for the execution time.
[SQLGeneratorCompatibility](#)	The value of the [TCustomDADataSet.BaseSQL](#) property is used to complete the refresh SQL statement, if the manually assigned [TCustomDAUpdateSQL.RefreshSQL](#) property contains only WHERE clause.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1 Classes

Classes in the **DBAccess** unit.

Classes

Name	Description
[EDAError](#)	A base class for exceptions that are raised when an error

	occurs on the server side.
[TCRDataSource](#)	Provides an interface between a DAC dataset components and data-aware controls on a form.
[TCustomConnectDialog](#)	A base class for the connect dialog components.
[TCustomDAConnection](#)	A base class for components used to establish connections.
[TCustomDADataSet](#)	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
[TCustomDASQL](#)	A base class for components executing SQL statements that do not return result sets.
[TCustomDAUpdateSQL](#)	A base class for components that provide DML statements for more flexible control over data modifications.
[TDACondition](#)	Represents a condition from the TDAConditions list.
[TDAConditions](#)	Holds a collection of TDACondition objects.
[TDAConnectionOptions](#)	This class allows setting up the behaviour of the TDAConnection class.
[TDAConnectionSSLOptions](#)	This class is used to set up the SSL options.
[TDADataSetOptions](#)	This class allows setting up the behaviour of the TDADataSet class.
[TDAEncryption](#)	Used to specify the options of the data encryption in a dataset.
[TDAMapRule](#)	Class that formes rules for Data Type Mapping.
[TDAMapRules](#)	Used for adding rules for DataSet fields mapping with both identifying by field

	name and by field type and Delphi field types.
[TDAMetaData](#)	A class for retrieving metainformation of the specified database objects in the form of dataset.
[TDAParam](#)	A class that forms objects to represent the values of the [parameters set](#).
[TDAParams](#)	This class is used to manage a list of TDAParam objects for an object that uses field parameters.
[TDATransaction](#)	A base class that implements functionality for controlling transactions.
[TMacro](#)	Object that represents the value of a macro.
[TMacros](#)	Controls a list of TMacro objects for the [TCustomDASQL.Macros](#) or [TCustomDADataSet](#) components.
[TPoolingOptions](#)	This class allows setting up the behaviour of the connection pool.
[TSmartFetchOptions](#)	Smart fetch options are used to set up the behavior of the SmartFetch mode.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.1 EDAError Class

A base class for exceptions that are raised when an error occurs on the server side.

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

Unit

DBAccess

Syntax

```
EDAError = class(EDatabaseError);
```

Remarks

EDAError is a base class for exceptions that are raised when an error occurs on the server side.

Request Support   DAC Forum   Provide Feedback

5.11.1.1.1 Members

EDAError class overview.

Properties

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

Request Support   DAC Forum   Provide Feedback

5.11.1.1.2 Properties

Properties of the **EDAError** class.

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

Public

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

See Also

- EDAError Class

- EDAError Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.1.2.1 Component Property

Contains the component that caused the error.

Class

EDAError

Syntax

```
property Component: TObject;
```

Remarks

The Component property contains the component that caused the error.

Request Support   DAC Forum   Provide Feedback

5.11.1.1.2.2 ErrorCode Property

Determines the error code returned by the server.

Class

EDAError

Syntax

```
property ErrorCode: integer;
```

Remarks

Use the ErrorCode property to determine the error code returned by MySQL. This value is always positive.

See MySQL server Reference Manual.

See Also

- EMyError

Request Support   DAC Forum   Provide Feedback

5.11.1.2 TCRDataSource Class

Provides an interface between a DAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TCRDataSource members.

Unit

DBAccess

Syntax

```
TCRDataSource = class(TDataSource);
```

Remarks

TCRDataSource provides an interface between a DAC dataset components and data-aware controls on a form.

TCRDataSource inherits its functionality directly from the TDataSource component.

At design time assign individual data-aware components' DataSource properties from their drop-down listboxes.

Request Support   DAC Forum   Provide Feedback

5.11.1.2.1 Members

TCRDataSource class overview.

Request Support   DAC Forum   Provide Feedback

5.11.1.3 TCustomConnectDialog Class

A base class for the connect dialog components.

For a list of all members of this type, see TCustomConnectDialog members.

Unit

DBAccess

Syntax

```
TCustomConnectDialog = class(TComponent);
```

Remarks

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name before connecting to a database. You can customize captions of buttons and labels by their properties.

© 1997-2025 Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback

5.11.1.3.1 Members

TCustomConnectDialog class overview.

Properties

Name	Description
CancelButton	Used to specify the label for the Cancel button.
Caption	Used to set the caption of dialog box.
ConnectButton	Used to specify the label for the Connect button.
DialogClass	Used to specify the class of the form that will be displayed to enter login information.
LabelSet	Used to set the language of buttons and labels captions.
PasswordLabel	Used to specify a prompt for password edit.
Retries	Used to indicate the number of retries of failed

	connections.
[SavePassword](#)	Used for the password to be displayed in ConnectDialog in asterisks.
[ServerLabel](#)	Used to specify a prompt for the server name edit.
[StoreLogInfo](#)	Used to specify whether the login information should be kept in system registry after a connection was established.
[UsernameLabel](#)	Used to specify a prompt for username edit.

Methods

Name	Description
[Execute](#)	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
[GetServerList](#)	Retrieves a list of available server names.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2 Properties

Properties of the **TCustomConnectDialog** class.

For a complete list of the **TCustomConnectDialog** class members, see the

[TCustomConnectDialog Members](#) topic.

Public

Name	Description
[CancelButton](#)	Used to specify the label for the Cancel button.
[Caption](#)	Used to set the caption of dialog box.
[ConnectButton](#)	Used to specify the label for the Connect button.

[DialogClass](#)	Used to specify the class of the form that will be displayed to enter login information.
[LabelSet](#)	Used to set the language of buttons and labels captions.
[PasswordLabel](#)	Used to specify a prompt for password edit.
[Retries](#)	Used to indicate the number of retries of failed connections.
[SavePassword](#)	Used for the password to be displayed in ConnectDialog in asterisks.
[ServerLabel](#)	Used to specify a prompt for the server name edit.
[StoreLogInfo](#)	Used to specify whether the login information should be kept in system registry after a connection was established.
[UsernameLabel](#)	Used to specify a prompt for username edit.

See Also

- [TCustomConnectDialog Class](#)

- [TCustomConnectDialog Class Members](#)

© 1997-2025 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2.1 CancelButton Property

Used to specify the label for the Cancel button.

Class

[TCustomConnectDialog](#)

Syntax

```
property CancelButton: string;
```

Remarks

Use the CancelButton property to specify the label for the Cancel button.

Request Support DAC Forum Provide Feedback

5.11.1.3.2.2 Caption Property

Used to set the caption of dialog box.

Class

TCustomConnectDialog

Syntax

```
property Caption: string;
```

Remarks

Use the Caption property to set the caption of dialog box.

Request Support DAC Forum Provide Feedback

5.11.1.3.2.3 ConnectButton Property

Used to specify the label for the Connect button.

Class

TCustomConnectDialog

Syntax

```
property ConnectButton: string;
```

Remarks

Use the ConnectButton property to specify the label for the Connect button.

Request Support DAC Forum Provide Feedback

5.11.1.3.2.4 DialogClass Property

Used to specify the class of the form that will be displayed to enter login information.

Class

[TCustomConnectDialog](#)

Syntax

```
property DialogClass: string;
```

Remarks

Use the DialogClass property to specify the class of the form that will be displayed to enter login information. When this property is blank, TCustomConnectDialog uses the default form - TConnectForm. You can write your own login form to enter login information and assign its class name to the DialogClass property. Each login form must have ConnectDialog: TCustomConnectDialog published property to access connection information. For details see the implementation of the connect form which sources are in the Lib subdirectory of the MyDAC installation directory.

See Also

- [GetServerList](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2.5 LabelSet Property

Used to set the language of buttons and labels captions.

Class

[TCustomConnectDialog](#)

Syntax

```
property LabelSet: TLabelSet default lsEnglish;
```

Remarks

Use the LabelSet property to set the language of labels and buttons captions.

The default value is lsEnglish.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class

TCustomConnectDialog

Syntax

```
property PasswordLabel: string;
```

Remarks

Use the PasswordLabel property to specify a prompt for password edit.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2.7 Retries Property

Used to indicate the number of retries of failed connections.

Class

TCustomConnectDialog

Syntax

```
property Retries: word default 3;
```

Remarks

Use the Retries property to determine the number of retries of failed connections.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2.8 SavePassword Property

Used for the password to be displayed in ConnectDialog in asterisks.

Class

[TCustomConnectDialog](#)

Syntax

```
property SavePassword: boolean default False;
```

Remarks

If True, and the Password property of the connection instance is assigned, the password in ConnectDialog is displayed in asterisks.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2.9 ServerLabel Property

Used to specify a prompt for the server name edit.

Class

[TCustomConnectDialog](#)

Syntax

```
property ServerLabel: string;
```

Remarks

Use the ServerLabel property to specify a prompt for the server name edit.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.3.2.10 StoreLogInfo Property

Used to specify whether the login information should be kept in system registry after a connection was established.

Class

TCustomConnectDialog

Syntax

```
property StoreLogInfo: boolean default True;
```

Remarks

Use the StoreLogInfo property to specify whether to keep login information in system registry after a connection was established using provided username, password and servername.

Set this property to True to store login information.

The default value is True.

Request Support   DAC Forum   Provide Feedback

5.11.1.3.2.11 UsernameLabel Property

Used to specify a prompt for username edit.

Class

TCustomConnectDialog

Syntax

```
property UsernameLabel: string;
```

Remarks

Use the UsernameLabel property to specify a prompt for username edit.

Request Support   DAC Forum   Provide Feedback

5.11.1.3.3 Methods

Methods of the **TCustomConnectDialog** class.

For a complete list of the **TCustomConnectDialog** class members, see the

[TCustomConnectDialog Members](#) topic.

Public

Name	Description
[Execute](#)	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
[GetServerList](#)	Retrieves a list of available server names.

See Also

- [TCustomConnectDialog Class](#)

- [TCustomConnectDialog Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.3.3.1 Execute Method

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.

Class

`TCustomConnectDialog`

Syntax

```
function Execute: boolean; virtual;
```

Return Value

True, if connected.

Remarks

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button. Returns True if connected. If user clicks Cancel, Execute returns False.

In the case of failed connection Execute offers to connect repeat [Retries](#) times.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.3.3.2 GetServerList Method

Retrieves a list of available server names.

Class

TCustomConnectDialog

Syntax

```
procedure GetServerList(List: TStrings); virtual;
```

Parameters

List

  Holds a list of available server names.

Remarks

Call the GetServerList method to retrieve a list of available server names. It is particularly relevant for writing custom login form.

See Also

- DialogClass

Request Support   DAC Forum   Provide Feedback

5.11.1.4 TCustomDAConnection Class

A base class for components used to establish connections.

For a list of all members of this type, see TCustomDAConnection members.

Unit

DBAccess

Syntax

```
TCustomDAConnection = class(TCustomConnection);
```

Remarks

TCustomDAConnection is a base class for components that establish connection with database, provide customised login support, and perform transaction control.

Do not create instances of TCustomDAConnection. To add a component that represents a connection to a source of data, use descendants of the TCustomDAConnection class.

Request Support   DAC Forum   Provide Feedback

5.11.1.4.1 Members

TCustomDAConnection class overview.

Properties

Name	Description
ConnectDialog	Allows to link a TCustomConnectDialog component.
ConnectString	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL	Allows customizing line breaks in string fields and parameters.
InTransaction	Indicates whether the transaction is active.
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the connection behavior.
Password	Serves to supply a password for login.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.
Server	Serves to supply the server

	name for login.
[Username](#)	Used to supply a user name for login.

Methods

Name	Description
[ApplyUpdates](#)	Overloaded. Applies changes in datasets.
[Commit](#)	Commits current transaction.
[Connect](#)	Establishes a connection to the server.
[CreateSQL](#)	Creates a component for queries execution.
[Disconnect](#)	Performs disconnect.
[ExecProc](#)	Allows to execute stored procedure or function providing its name and parameters.
[ExecProcEx](#)	Allows to execute a stored procedure or function.
[ExecSQL](#)	Executes a SQL statement with parameters.
[ExecSQLEx](#)	Executes any SQL statement outside the TQuery or TSQL components.
[GetDatabaseNames](#)	Returns a database list from the server.
[GetKeyFieldNames](#)	Provides a list of available key field names.
[GetStoredProcNames](#)	Returns a list of stored procedures from the server.
[GetTableNames](#)	Provides a list of available tables names.
[MonitorMessage](#)	Sends a specified message through the [TCustomDASQLMonitor](#) component.
[PerformConnect](#)	Establishes a connection to the server regardless of the LoginPrompt property.

Ping	Used to check state of connection to the server.
RemoveFromPool	Marks the connection that should not be returned to the pool after disconnect.
Rollback	Discards all current data changes and ends transaction.
StartTransaction	Begins a new user transaction.

Events

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2 Properties

Properties of the **TCustomDAConnection** class.

For a complete list of the **TCustomDAConnection** class members, see the

TCustomDAConnection Members topic.

Public

Name	Description
ConnectDialog	Allows to link a TCustomConnectDialog component.
ConnectString	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL	Allows customizing line breaks in string fields and parameters.

InTransaction	Indicates whether the transaction is active.
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the connection behavior.
Password	Serves to supply a password for login.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.
Server	Serves to supply the server name for login.
Username	Used to supply a user name for login.

See Also

- TCustomDAConnection Class

- TCustomDAConnection Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.1 ConnectDialog Property

Allows to link a TCustomConnectDialog component.

Class

`TCustomDAConnection`

Syntax

```
property ConnectDialog: TCustomConnectDialog;
```

Remarks

Use the ConnectDialog property to assign to connection a TCustomConnectDialog component.

See Also

- TCustomConnectDialog

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

`TCustomDAConnection`

Syntax

```
property ConnectString: string stored False;
```

Remarks

MyDAC recognizes an ODBC-like syntax in provider string property values. Within the string, elements are delimited by using a semicolon. Each element consists of a keyword, an equal sign character, and the value passed on initialization. For example:

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

Connection parameters

The following connection parameters can be used to customize connection:

Parameter Name	Description
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Pooling	Enables or disables using connection pool.
ConnectionLifeTime	Used to specify the maximum time during which an opened connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in

	connection pool.
[Validate Connection](#)	Used for a connection to be validated when it is returned from the pool.
[Server](#)	Serves to supply the server name for login.
[Username](#)	Used to supply a user name for login.
[Password](#)	Used to supply a user name for login.
[Database](#)	Used to set the name of the database to associate with TMyConnection component.
[Charset](#)	Used to set the character set that MyDAC uses to read and write character data.
[Protocol](#)	Used to specify the Network protocol of connection with MySQL server.
[UseUnicode](#)	Used to enable or disable Unicode support.
[Port](#)	Used to specify the port number for the connection.
[ConnectionTimeout](#)	Used to specify the amount of time before an attempt to make a connection is considered unsuccessful.
[Compress](#)	Used to apply compression on transferring data.
[Embedded](#)	Used to specify the server that will be used to connect.
[Direct](#)	Used to work without using MySQL client library (libmysql.dll).
[IPVersion](#)	Used to specify the version of the Internet Protocol.
P:Devart.Mydac.TMyConnectionSSLOptions.CACert	Holds the pathname to the certificate authority file.
P:Devart.Mydac.TMyConnectionSSLOptions.Cert	Holds the pathname to the certificate file.
P:Devart.Mydac.TMyConnectionSSLOptions.ChipherList	Holds the list of allowed ciphers to use for SSL encryption.
P:Devart.Mydac.TMyConnectionSSLOptions.Key	Holds the pathname to the key file.
[Username](#)	Holds the user name for HTTP authorization.
[Password](#)	Holds the password for HTTP authorization.
[URL](#)	Holds the url of the tunneling PHP script.
[Hostname](#)	Holds the host name or IP address to connect to proxy server.
[Username](#)	Holds the proxy server account name.
[Password](#)	Holds the password for the proxy server

	account.
[Port](#)	Used to specify the port number for TCP/IP connection with proxy server.

See Also

- [Password](#)

- [Username](#)

- [Server](#)

- [Connect](#)

© 1997-2025 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.2.3 ConvertEOL Property

Allows customizing line breaks in string fields and parameters.

Class

[TCustomDAConnection](#)

Syntax

```
property ConvertEOL: boolean default False;
```

Remarks

Affects the line break behavior in string fields and parameters. When fetching strings (including the 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.

© 1997-2025 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.2.4 InTransaction Property

Indicates whether the transaction is active.

Class

TCustomDAConnection

Syntax

```
property InTransaction: boolean;
```

Remarks

Examine the InTransaction property at runtime to determine whether user transaction is currently in progress. In other words InTransaction is set to True when user explicitly calls StartTransaction. Calling Commit or Rollback sets InTransaction to False. The value of the InTransaction property cannot be changed directly.

See Also

- StartTransaction

- Commit

- Rollback

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.5 LoginPrompt Property

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

Class

TCustomDAConnection

Syntax

```
property LoginPrompt default DefValLoginPrompt;
```

Remarks

Specifies whether a login dialog appears immediately before opening a new connection. If ConnectDialog is not specified, the default connect dialog will be shown. The connect dialog will appear only if the MyDacVcl unit appears to the uses clause.

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.6 Options Property

Specifies the connection behavior.

Class

`TCustomDAConnection`

Syntax

```
property Options: TDAConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of the connection.

Descriptions of all options are in the table below.

Option Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.
KeepDesignConnected	Prevents 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 the connection is lost.

See Also

- Disconnected Mode

- Working in an Unstable Network

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.7 Password Property

Serves to supply a password for login.

Class

`TCustomDAConnection`

Syntax

```
property Password: string stored False;
```

Remarks

Use the Password property to supply a password to handle server's request for a login.

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

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.8 Pooling Property

Enables or disables using connection pool.

Class

`TCustomDAConnection`

Syntax

```
property Pooling: boolean default DefValPooling;
```

Remarks

Normally, when TCustomDAConnection establishes connection with the server it takes server memory and time resources for allocating new server connection. For example, pooling can be very useful when using disconnect mode. If an application has wide user

activity that forces many connect/disconnect operations, it may spend a lot of time on creating connection and sending requests to the server. TCustomDAConnection has software pool which stores open connections with identical parameters.

Connection pool uses separate thread that validates the pool every 30 seconds. Pool validation consists of checking each connection in the pool. If a connection is broken due to a network problem or another reason, it is deleted from the pool. The validation procedure removes also connections that are not used for a long time even if they are valid from the pool.

Set Pooling to True to enable pooling. Specify correct values for PoolingOptions. Two connections belong to the same pool if they have identical values for the parameters: MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime , Server, Username, Password, TCustomMyConnection.Database, TCustomMyConnection.IsolationLevel, TMyConnection.Port, TMyConnection.IOHandler, TCustomMyConnection.ConnectionTimeout, TMyConnectionOptions.Compress, TMyConnectionOptions.Direct, TMyConnectionOptions.Embedded, TMyConnectionOptions.Protocol, TCustomMyConnectionOptions.Charset, TCustomMyConnectionOptions.UseUnicode, TCustomMyConnectionOptions.NumericType.

Note: Using Pooling := True can cause errors with working with temporary tables.

See Also
- Username

- Password

- PoolingOptions

- Connection Pooling

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.9 PoolingOptions Property

Specifies the behaviour of connection pool.

Class

`TCustomDAConnection`

Syntax

```
property PoolingOptions: TPoolingOptions;
```

Remarks

Set the properties of PoolingOptions to specify the behaviour of connection pool.

Descriptions of all options are in the table below.

Option Name	Description
ConnectionLifetime	Used to specify the maximum time during which an open connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in the connection pool.
PoolId	Used to specify an ID for a connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

- Pooling

Request Support   DAC Forum   Provide Feedback

5.11.1.4.2.10 Server Property

Serves to supply the server name for login.

Class

TCustomDAConnection

Syntax

```
property Server: string;
```

Remarks

Use the Server property to supply server name to handle server's request for a login. If this property is not set, MyDAC tries to connect to Localhost.

The Server property can be used only if [TMyConnectionOptions.Embedded](#) is set to False.

See Also

- [Username](#)

- [Password](#)

- [TMyConnection.Port](#)

© 1997-2025 Devart. All Rights Reserved.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.2.11 Username Property

Used to supply a user name for login.

Class

`TCustomDAConnection`

Syntax

```
property Username: string;
```

Remarks

Use the Username property to supply a user name to handle server's request for login. If this property is not set, MyDAC tries to connect with the empty 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](#)

© 1997-2025 Devart. All Rights Reserved.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.3 Methods

Methods of the **TCustomDAConnection** class.

For a complete list of the **TCustomDAConnection** class members, see the

TCustomDAConnection Members topic.

Public

Name	Description
ApplyUpdates	Overloaded. Applies changes in datasets.
Commit	Commits current transaction.
Connect	Establishes a connection to the server.
CreateSQL	Creates a component for queries execution.
Disconnect	Performs disconnect.
ExecProc	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx	Allows to execute a stored procedure or function.
ExecSQL	Executes a SQL statement with parameters.
ExecSQLEx	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames	Returns a database list from the server.
GetKeyFieldNames	Provides a list of available key field names.
GetStoredProcNames	Returns a list of stored procedures from the server.
GetTableNames	Provides a list of available tables names.
MonitorMessage	Sends a specified message through the TCustomDASQLMonitor component.

[PerformConnect](#)	Establishes a connection to the server regardless of the LoginPrompt property.
[Ping](#)	Used to check state of connection to the server.
[RemoveFromPool](#)	Marks the connection that should not be returned to the pool after disconnect.
[Rollback](#)	Discards all current data changes and ends transaction.
[StartTransaction](#)	Begins a new user transaction.

See Also

- [TCustomDAConnection Class](#)

- [TCustomDAConnection Class Members](#)

Request Support **DAC Forum** **Provide Feedback**

5.11.1.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class

[TCustomDAConnection](#)

Overload List

Name	Description
[ApplyUpdates](#)	Applies changes from all active datasets.
[ApplyUpdates(**const** DataSets: **array of** TCustomDADataSet)](#)	Applies changes from the specified datasets.

Request Support **DAC Forum** **Provide Feedback**

Applies changes from all active datasets.

Class

`TCustomDAConnection`

Syntax

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

Remarks

Call the ApplyUpdates method to write all pending cached updates from all active datasets attached to this connection to a database or from specific datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions, and clearing the cache when the operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than calling each individual dataset's ApplyUpdates method.

See Also
- TMemDataSet.CachedUpdates

- TMemDataSet.ApplyUpdates

Request Support   DAC Forum   Provide Feedback

Applies changes from the specified datasets.

Class

`TCustomDAConnection`

Syntax

```
procedure ApplyUpdates(const DataSets: array of
TCustomDADataSet); overload; virtual;
```

Parameters

DataSets
   A list of datasets changes in which are to be applied.

Remarks

Call the ApplyUpdates method to write all pending cached updates from the specified datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions and clearing the cache when operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than calling each individual dataset's ApplyUpdates method.

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.2 Commit Method

Commits current transaction.

Class

`TCustomDAConnection`

Syntax

```
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database and then ends the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also

- Rollback

- StartTransaction

- TCustomMyDataSet.FetchAll

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.3 Connect Method

Establishes a connection to the server.

Class

TCustomDAConnection

Syntax

```
procedure Connect; overload;procedure Connect(const
ConnectString: string); overload;
```

Remarks

Call the Connect method to establish a connection to the server. Connect sets the Connected property to True. If 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.

See Also

- Disconnect

- Username

- Password

- Server

- ConnectDialog

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.4 CreateSQL Method

Creates a component for queries execution.

Class

TCustomDAConnection

Syntax

```
function CreateSQL: TCustomDASQL; virtual;
```

Return Value

A new instance of the class.

Remarks

Call the CreateSQL to return a new instance of the TCustomDASQL class and associates it with this connection object. In the descendant classes this method should be overridden to create an appropriate descendant of the TCustomDASQL component.

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.5 Disconnect Method

Performs disconnect.

Class

TCustomDAConnection

Syntax

```
procedure Disconnect;
```

Remarks

Call the Disconnect method to drop a connection to database. Before the connection component is deactivated, all associated datasets are closed. Calling Disconnect is similar to setting the Connected property to False.

In most cases, closing a connection frees system resources allocated to the connection.

If user transaction is active, e.g. the InTransaction flag is set, calling to Disconnect 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

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.6 ExecProc Method

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

Class

TCustomDAConnection

Syntax

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

Parameters

Name
  Holds the name of the stored procedure or function.

Params
  Holds the parameters of the stored procedure or function.

Return Value
  the result of the stored procedure.

Remarks

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

Use the following Name value syntax for executing specific overloaded routine:
"StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded
stored procedure, while the second example executes the fifth overloaded procedure.

Assign parameters' values to the Params array in exactly the same order and number as they
appear in the stored procedure declaration. Out parameters of the procedure can be
accessed with the ParamByName procedure.

If the value of an input parameter was not included to the Params array, parameter default
value is taken. Only parameters at the end of the list can be unincluded to the Params array. If
the parameter has no default value, the NULL value is sent.

Note: Stored functions unlike stored procedures return result values that are obtained
internally through the RESULT parameter. You will no longer have to provide anonymous
value in the Params array to describe the result of the function. The stored function result is

obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For further examples of parameter usage see ExecSQL, ExecSQLEx.

Example

For example, having stored function declaration presented in Example 1), you may execute it and retrieve its result with commands presented in Example 2):

```
Example 1)
CREATE procedure MY_SUM (
      A INTEGER,
      B INTEGER)
RETURNS (
      RESULT INTEGER)
as
begin
   Result = a + b;
end;
Example 2)
Label1.Caption:= MyMyConnection1.ExecProc('My_Sum', [10, 20]);
Label2.Caption:= MyMyConnection1.ParamByName('Result').AsString;
```

See Also

- ExecProcEx

- ExecSQL

- ExecSQLEx

© 1997-2025
Devart. All Rights   Request Support   DAC Forum   Provide Feedback
Reserved.

5.11.1.4.3.7 ExecProcEx Method

Allows to execute a stored procedure or function.

Class

`TCustomDAConnection`

Syntax

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

Parameters

Name
  Holds the stored procedure name.

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

Return Value
  the result of the stored procedure.

Remarks

Allows to execute a stored procedure or function. Provide the stored procedure name and its parameters to the call of ExecProcEx.

Use the following Name value syntax for executing specific overloaded routine: "StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign pairs of parameters' names and values to a Params array so that every name comes before its corresponding value when an array is being indexed.

Out parameters of the procedure can be accessed with the ParamByName procedure. If the value for an input parameter was not included to the Params array, the parameter default value is taken. If the parameter has no default value, the NULL value is sent.

Note: Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. Stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For an example of parameters usage see ExecSQLEx.

Example

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

```
Connection.ExecProcEx('Some_Stored_Procedure', ['Param_Name1', 'Param_Value1
```

See Also

- [ExecSQL](#)

- [ExecSQLEx](#)

- [ExecProc](#)

　　[Request Support](#)　　　[DAC Forum](#)　　　　[Provide Feedback](#)

5.11.1.4.3.8 ExecSQL Method

Executes a SQL statement with parameters.

Class

`TCustomDAConnection`

Syntax

```
function ExecSQL(const Text: string): variant;
overload;function ExecSQL(const Text: string; const Params:
array of variant): variant; overload; virtual;
```

Parameters

Text
  a SQL statement to be executed.

Params
  Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

  Out parameter with the name Result will hold the result of function having data type dtString. Otherwise returns Null.

Remarks

Use the ExecSQL method to execute any SQL statement outside the [TCustomDADataSet](#) or [TCustomDASQL](#) components. Supply the Params array with the values of parameters arranged in the same order as they appear in a SQL statement which itself is passed to the Text string parameter.

Note: If a query doesn't have parameters (Params.Count = 0), this query will be executed faster.

See Also

- [ExecSQLEx](#)

- [ExecProc](#)

- [TCustomMyConnection.ExecSQL](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.3.9 ExecSQLEx Method

Executes any SQL statement outside the TQuery or TSQL components.

Class

`TCustomDAConnection`

Syntax

```
function ExecSQLEx(const Text: string; const Params: array of
variant): variant; virtual;
```

Parameters

Text
  a SQL statement to be executed.

Params
  Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

  Out parameter with the name Result will hold the result of a function having data type dtString. Otherwise returns Null.

Remarks

Call the ExecSQLEx method to execute any SQL statement outside the TQuery or TSQL components. Supply the Params array with values arranged in pairs of parameter name and its value. This way each parameter name in the array is found on even index values whereas parameter value is on odd index value but right after its parameter name. The parameter pairs must be arranged according to their occurrence in a SQL statement which itself is passed in the Text string parameter.

The Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the ExecSQLEx method.

Out parameter with the name Result will hold the result of a function having data type dtString. If neither of the parameters in the Text statement is named Result, ExecSQLEx will return Null.

To get the values of OUT parameters use the ParamByName function.

Example

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

See Also

*   ExecSQL

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.10 GetDatabaseNames Method

Returns a database list from the server.

Class

TCustomDAConnection

Syntax

```
procedure GetDatabaseNames(List: TStrings); virtual;
```

Parameters

List
    A TStrings descendant that will be filled with database names.

Remarks

Populates a string list with the names of databases.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetDatabaseNames.

See Also

*   GetTableNames

- [GetStoredProcNames](#)

- [TCustomMyConnection.Database](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.3.11 GetKeyFieldNames Method

Provides a list of available key field names.

Class

[TCustomDAConnection](#)

Syntax

```
procedure GetKeyFieldNames(const TableName: string; List:
TStrings); virtual;
```

Parameters

TableName
   Holds the table name

List
   The list of available key field names

Return Value

   Key field name

Remarks

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also
- [GetTableNames](#)

- [GetStoredProcNames](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.3.12 GetStoredProcNames Method

Returns a list of stored procedures from the server.

Class

[TCustomDAConnection](#)

Syntax

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

Parameters

List

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

AllProcs

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

Remarks

Call the GetStoredProcNames method to get the names of available stored procedures and functions. GetStoredProcNames populates a string list with the names of stored procs in the database. If AllProcs = True, the procedure returns to the List parameter the names of the stored procedures that belong to all schemas; otherwise, List will contain the names of functions that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetStoredProcNames.

See Also

- [GetDatabaseNames](#)

- [GetTableNames](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.3.13 GetTableNames Method

Provides a list of available tables names.

Class

[TCustomDAConnection](#)

Syntax

```
procedure GetTableNames(List: TStrings; AllTables: boolean =
False; OnlyTables: boolean = False); virtual;
```

Parameters

List
  A TStrings descendant that will be filled with table names.

AllTables
  True, if procedure returns all table names including the names of system tables to the List
  parameter.

OnlyTables

Remarks

Call the GetTableNames method to get the names of available tables. Populates a string list
with the names of tables in the database. If AllTables = True, procedure returns all table
names including the names of system tables to the List parameter, otherwise List will not
contain the names of system tables. If AllTables = True, the procedure returns to the List
parameter the names of the tables that belong to all schemas; otherwise, List will contain the
names of the tables that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by the
data produced by GetTableNames.

See Also

- [GetDatabaseNames](#)

- [GetStoredProcNames](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.3.14 MonitorMessage Method

Sends a specified message through the TCustomDASQLMonitor component.

Class

`TCustomDAConnection`

Syntax

```
procedure MonitorMessage(const Msg: string);
```

Parameters

Msg
  Message text that will be sent.

Remarks

Call the MonitorMessage method to output specified message via the
TCustomDASQLMonitor component.

See Also

- TCustomDASQLMonitor

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.15 PerformConnect Method

Establishes a connection to the server regardless of the LoginPrompt property.

Class

`TCustomDAConnection`

Syntax

```
procedure PerformConnect(Retry: boolean = False);
```

Parameters

Retry
  When the Retry argument is set to False (default), the BeforeConnect event handler is
  called before establishing the connection.

If Retry = True, BeforeConnect is not invoked.

Remarks

Unlike the Connect method, the PerformConnect method establishes a connection regardless of the LoginPrompt property. When PerformConnect is called with LoginPrompt = True, the Connection Dialog is not displayed, and the OnLogin event handler is not invoked. After establishing the connection, the AfterConnect event handler is triggered.

Note: PerformConnect does not handle exceptions. Exceptions should be handled as usual in the calling code.

See Also
* Connect

* LoginPrompt

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.16 Ping Method

Used to check state of connection to the server.

Class

TCustomDAConnection

Syntax

```
procedure Ping;
```

Remarks

The method is used for checking server connection state. Use the Ping method to prevent server from closing connection on exceeding the wait_timeout value of the MySQL server. For this, call the Ping method in intervals smaller than the server wait_timeout value.

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.17 RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

Class

`TCustomDAConnection`

Syntax

```
procedure RemoveFromPool;
```

Remarks

Call the RemoveFromPool method to mark the connection that should be deleted after disconnect instead of returning to the connection pool.

See Also

- Pooling

- PoolingOptions

Request Support   DAC Forum   Provide Feedback

5.11.1.4.3.18 Rollback Method

Discards all current data changes and ends transaction.

Class

`TCustomDAConnection`

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call the Rollback method to discard all updates, insertions, and deletions of data associated with the current transaction to the database server and then end the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also

- [Commit](#)

- [StartTransaction](#)

- [TCustomMyDataSet.FetchAll](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.3.19 StartTransaction Method

Begins a new user transaction.

Class

[TCustomDAConnection](#)

Syntax

```
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new user transaction against the database
server. Before calling StartTransaction, an application should check the status of the
[InTransaction](#) property. If InTransaction is True, indicating that a transaction is already in
progress, a subsequent call to StartTransaction without first calling [Commit](#) or [Rollback](#) to
end the current transaction raises EDatabaseError. Calling StartTransaction when
connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by
the server until an application calls Commit to save the changes, or Rollback to cancel them.

See Also
- [Commit](#)

- [Rollback](#)

- [InTransaction](#)

- [TCustomMyConnection.IsolationLevel](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.4.4 Events

Events of the **TCustomDAConnection** class.

For a complete list of the **TCustomDAConnection** class members, see the

TCustomDAConnection Members topic.

Public

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

See Also

- TCustomDAConnection Class

- TCustomDAConnection Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.4.4.1 OnConnectionLost Event

This event occurs when connection was lost.

Class

`TCustomDAConnection`

Syntax

```
property OnConnectionLost: TConnectionLostEvent;
```

Remarks

Write the OnConnectionLost event handler to process fatal errors and perform failover.

Note: To use the OnConnectionLost event handler, you should explicitly add the MemData unit to the 'uses' list and set the TCustomDAConnection.Options.LocalFailover property to True.

5.11.1.4.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

TCustomDAConnection

Syntax

```
property OnError: TDAConnectionErrorEvent;
```

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

5.11.1.5 TCustomDADataSet Class

Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.

For a list of all members of this type, see TCustomDADataSet members.

Unit

DBAccess

Syntax

```
TCustomDADataSet = class(TMemDataSet);
```

Remarks

TCustomDADataSet encapsulates general set of properties, events, and methods for working

with data accessed through various database engines. All database-specific features are supported by descendants of TCustomDADataSet.

Applications should not use TCustomDADataSet objects directly.

Inheritance Hierarchy

TMemDataSet

   TCustomDADataSet

   Request Support   DAC Forum   Provide Feedback

5.11.1.5.1 Members

TCustomDADataSet class overview.

Properties

Name	Description
BaseSQL	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Conditions	Used to add WHERE conditions to a query
Connection	Used to specify a connection object to use to connect to a data store.
DataTypeMap	Used to set data type mapping rules
Debug	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

[Disconnected](#)	Used to keep dataset opened after connection is closed.
[FetchRows](#)	Used to define the number of rows to be transferred across the network at the same time.
[FilterSQL](#)	Used to change the WHERE clause of SELECT statement and reopen a query.
[FinalSQL](#)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
[IndexFieldNames](#) (inherited from **[TMemDataSet](#)**)	Used to get or set the list of fields on which the recordset is sorted.
[IsQuery](#)	Used to check whether SQL statement returns rows.
[KeyExclusive](#) (inherited from **[TMemDataSet](#)**)	Specifies the upper and lower boundaries for a range.
[KeyFields](#)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
[LocalConstraints](#) (inherited from **[TMemDataSet](#)**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
[LocalUpdate](#) (inherited from **[TMemDataSet](#)**)	Used to prevent implicit update of rows on database server.
[MacroCount](#)	Used to get the number of macros associated with the Macros property.
[Macros](#)	Makes it possible to change SQL queries easily.
[MasterFields](#)	Used to specify the names

		of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
[MasterSource](#)		Used to specify the data source component which binds current dataset to the master one.
[Options](#)		Used to specify the behaviour of TCustomDADataSet object.
[ParamCheck](#)		Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
[ParamCount](#)		Used to indicate how many parameters are there in the Params property.
[Params](#)		Used to view and set parameter names, values, and data types dynamically.
[Prepared](#) (inherited from **[TMemDataSet](#)**)		Determines whether a query is prepared for execution or not.
[Ranged](#) (inherited from **[TMemDataSet](#)**)		Indicates whether a range is applied to a dataset.
[ReadOnly](#)		Used to prevent users from updating, inserting, or deleting data in the dataset.
[RefreshOptions](#)		Used to indicate when the editing record is refreshed.
[RowsAffected](#)		Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
[SQL](#)		Used to provide a SQL statement that a query component executes when its Open method is called.
[SQLDelete](#)		Used to specify a SQL statement that will be used

	when applying a deletion to a record.
[SQLInsert](#)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[SQLUpdate](#)	Used to specify a SQL statement that will be used when applying an update to a dataset.
[UniDirectional](#)	Used if an application does not need bidirectional access to records in the result set.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
[AddWhere](#)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
[ApplyRange](#) (inherited from **[TMemDataSet](#)**)	Applies a range to the dataset.

ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor	Shares data from another dataset.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching	Used to learn whether TCustomDADataSet is still

	fetching rows.
[FetchingAll](#)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
[FindKey](#)	Searches for a record which contains specified field values.
[FindMacro](#)	Finds a macro with the specified name.
[FindNearest](#)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
[FindParam](#)	Determines if a parameter with the specified name exists in a dataset.
[GetBlob](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
[GetDataType](#)	Returns internal field types defined in the MemData and accompanying modules.
[GetFieldObject](#)	Returns a multireference shared object from field.
[GetFieldPrecision](#)	Retrieves the precision of a number field.
[GetFieldScale](#)	Retrieves the scale of a number field.
[GetKeyFieldNames](#)	Provides a list of available key field names.
[GetOrderBy](#)	Retrieves an ORDER BY clause from a SQL statement.
[GotoCurrent](#)	Sets the current record in this dataset similar to the current record in another dataset.
[Locate](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Searches a dataset for a specific record and positions the cursor on

	it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a macro with the specified name.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.

SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved	Determines if the SQL property value was saved to the BaseSQL property.
UnLock	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

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

	insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.11.1.5.2 Properties

Properties of the **TCustomDADataSet** class.

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

Public

Name	Description
BaseSQL	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Conditions	Used to add WHERE conditions to a query
Connection	Used to specify a connection object to use to connect to a data store.
DataTypeMap	Used to set data type mapping rules
Debug	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields	Used to specify the fields that correspond to the

	foreign key fields from MasterFields when building master/detail relationship.
[Disconnected](#)	Used to keep dataset opened after connection is closed.
[FetchRows](#)	Used to define the number of rows to be transferred across the network at the same time.
[FilterSQL](#)	Used to change the WHERE clause of SELECT statement and reopen a query.
[FinalSQL](#)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
[IndexFieldNames](#) (inherited from **[TMemDataSet](#)**)	Used to get or set the list of fields on which the recordset is sorted.
[IsQuery](#)	Used to check whether SQL statement returns rows.
[KeyExclusive](#) (inherited from **[TMemDataSet](#)**)	Specifies the upper and lower boundaries for a range.
[KeyFields](#)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
[LocalConstraints](#) (inherited from **[TMemDataSet](#)**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
[LocalUpdate](#) (inherited from **[TMemDataSet](#)**)	Used to prevent implicit update of rows on database server.
[MacroCount](#)	Used to get the number of macros associated with the Macros property.

Macros	Makes it possible to change SQL queries easily.
MasterFields	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource	Used to specify the data source component which binds current dataset to the master one.
Options	Used to specify the behaviour of TCustomDADataSet object.
ParamCheck	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount	Used to indicate how many parameters are there in the Params property.
Params	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions	Used to indicate when the editing record is refreshed.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL	Used to provide a SQL statement that a query component executes when

	its Open method is called.
[SQLDelete](#)	Used to specify a SQL statement that will be used when applying a deletion to a record.
[SQLInsert](#)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[SQLUpdate](#)	Used to specify a SQL statement that will be used when applying an update to a dataset.
[UniDirectional](#)	Used if an application does not need bidirectional access to records in the result set.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

See Also

- [TCustomDADataSet Class](#)

- [TCustomDADataSet Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.5.2.1 BaseSQL Property

Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.

Class

[TCustomDADataSet](#)

Syntax

```
property BaseSQL: string;
```

Remarks

Use the BaseSQL property to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL, only macros are expanded. SQL text with all these changes can be returned by [FinalSQL](#).

See Also

- [FinalSQL](#)

- [AddWhere](#)

- [SaveSQL](#)

- [SQLSaved](#)

- [RestoreSQL](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

[TCustomDADataSet](#)

Syntax

```
property Conditions: TDAConditions stored False;
```

See Also

- TDAConditions

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.3 Connection Property

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

Class

TCustomDADataSet

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.4 DataTypeMap Property

Used to set data type mapping rules

Class

TCustomDADataSet

Syntax

```
property DataTypeMap: TDAMapRules stored IsMapRulesStored;
```

See Also

- TDAMapRules

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.5 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

`TCustomDADataSet`

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

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

Note: If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDASQL.Debug

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.6 DetailFields Property

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

Class

TCustomDADataSet

Syntax

```
property DetailFields: string;
```

Remarks

Use the DetailFields property to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship. DetailFields is a string containing one or more field names in the detail table. Separate field names with semicolons.

Use Field Link Designer to set the value in design time.

See Also

- MasterFields

- MasterSource

    Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.7 Disconnected Property

Used to keep dataset opened after connection is closed.

Class

TCustomDADataSet

Syntax

```
property Disconnected: boolean;
```

Remarks

Set the Disconnected property to True to keep dataset opened after connection is closed.

    Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.8 FetchRows Property

Used to define the number of rows to be transferred across the network at the same time.

Class

TCustomDADataSet

Syntax

```
property FetchRows: integer default 25;
```

Remarks

The number of rows that will be transferred across the network at the same time. This property can have a great impact on performance. So it is preferable to choose the optimal value of the FetchRows property for each SQL statement and software/hardware configuration experimentally.

The default value is 25.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.9 FilterSQL Property

Used to change the WHERE clause of SELECT statement and reopen a query.

Class

TCustomDADataSet

Syntax

```
property FilterSQL: string;
```

Remarks

The FilterSQL property is similar to the Filter property, but it changes the WHERE clause of SELECT statement and reopens query. Syntax is the same to the WHERE clause.

Note: the FilterSQL property adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

Example

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

See Also

- AddWhere

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.10 FinalSQL Property

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Class

TCustomDADataSet

Syntax

```
property FinalSQL: string;
```

Remarks

Use FinalSQL to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros. This is the exact statement that will be passed on to the database server.

See Also

- FinalSQL

- AddWhere

- SaveSQL

- SQLSaved

- RestoreSQL

- BaseSQL

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.11 IsQuery Property

Used to check whether SQL statement returns rows.

Class

TCustomDADataSet

Syntax

```
property IsQuery: boolean;
```

Remarks

After the TCustomDADataSet component is prepared, the IsQuery property returns True if SQL statement is a SELECT query.

Use the IsQuery property to check whether the SQL statement returns rows or not.

IsQuery is a read-only property. Reading IsQuery on unprepared dataset raises an exception.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.12 KeyFields Property

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.

Class

TCustomDADataSet

Syntax

```
property KeyFields: string;
```

Remarks

TCustomDADataset uses the KeyFields property to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database. For this feature KeyFields may hold a list of semicolon-delimited field names. If KeyFields is not defined before opening a dataset, TCustomDADataset uses the metainformation sent by the server together with data.

See Also

- SQLDelete

- SQLInsert

- SQLRefresh

- SQLUpdate

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.13 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

`TCustomDADataSet`

Syntax

```
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- Macros

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.14 Macros Property

Makes it possible to change SQL queries easily.

Class

`TCustomDADataSet`

Syntax

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

Remarks

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

Example

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

See Also

- TMacro

- MacroByName

- Params

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.15 MasterFields Property

Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

Class

TCustomDADataSet

Syntax

```
property MasterFields: string;
```

Remarks

Use the MasterFields property after setting the [MasterSource](#) property to specify the names of one or more fields that are used as foreign keys for this dataset when establishing detail/ master relationship between it and the dataset specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in these fields are used to select corresponding records in this table for display.

Use Field Link Designer to set the values at design time after setting the MasterSource property.

See Also

- [DetailFields](#)

- [MasterSource](#)

- [Master/Detail Relationships](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.2.16 MasterSource Property

Used to specify the data source component which binds current dataset to the master one.

Class

[TCustomDADataSet](#)

Syntax

```
property MasterSource: TDataSource;
```

Remarks

The MasterSource property specifies the data source component which binds current dataset to the master one.

TCustomDADataset uses MasterSource to extract foreign key fields values from the master dataset when building master/detail relationship between two datasets. MasterSource must point to another dataset; it cannot point to this dataset component.

When MasterSource is not **nil** dataset fills parameter values with corresponding field values from the current record of the master dataset.

Note: Do not set the DataSource property when building master/detail relationships. Although it points to the same object as the MasterSource property, it may lead to undesirable results.

See Also

- MasterFields

- DetailFields

- Master/Detail Relationships

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.17 Options Property

Used to specify the behaviour of TCustomDADataSet object.

Class

TCustomDADataSet

Syntax

```
property Options: TDADataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TCustomDADataSet object.

Descriptions of all options are in the table below.

Option Name	Description
AutoPrepare	Used to execute automatic Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the

	DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.
SetFieldsReadOnly	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
[UpdateAllFields](#)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
[UpdateBatchSize](#)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

See Also

- [Master/Detail Relationships](#)

- [TMemDataSet.CachedUpdates](#)

© 1997-2025 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.2.18 ParamCheck Property

Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Class

`TCustomDADataSet`

Syntax

```
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Set ParamCheck to True to let dataset automatically generate the Params property for the dataset based on a SQL statement.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of stored procedures which themselves will accept parameterized values. The default value is True.

See Also

- Params

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.19 ParamCount Property

Used to indicate how many parameters are there in the Params property.

Class

TCustomDADataSet

Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

See Also

- Params

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.20 Params Property

Used to view and set parameter names, values, and data types dynamically.

Class

TCustomDADataSet

Syntax

```
property Params: TDAParams stored False;
```

Remarks

Contains the parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

See Also

- ParamByName

- Macros

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.21 ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

Class

`TCustomDADataSet`

Syntax

```
property ReadOnly: boolean default False;
```

Remarks

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning that users can potentially alter data stored in the dataset.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True.

When ReadOnly is True, the dataset's CanModify property is False.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.22 RefreshOptions Property

Used to indicate when the editing record is refreshed.

Class

TCustomDADataSet

Syntax

```
property RefreshOptions: TRefreshOptions default [];
```

Remarks

Use the RefreshOptions property to determine when the editing record is refreshed.

Refresh is performed by the RefreshRecord method.

It queries the current record and replaces one in the dataset. Refresh record is useful when the table has triggers or the table fields have default values. Use roBeforeEdit to get actual data before editing.

The default value is [].

See Also

- RefreshRecord

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.23 Row sAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

TCustomDADataSet

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.24 SQL Property

Used to provide a SQL statement that a query component executes when its Open method is called.

Class

`TCustomDADataSet`

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a query component executes when its Open method is called. At the design time the SQL property can be edited by invoking the String List editor in Object Inspector.

When SQL is changed, TCustomDADataSet calls Close and UnPrepare.

See Also
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.25 SQLDelete Property

Used to specify a SQL statement that will be used when applying a deletion to a record.

Class

[TCustomDADataSet](#)

Syntax

```
property SQLDelete: TStrings;
```

Remarks

Use the SQLDelete property to specify the SQL statement that will be used when applying a deletion to a record. Statements can be parameterized queries.

To create a SQLDelete statement at design-time, use the query statements editor.

Example

```
DELETE FROM Orders
    WHERE
        OrderID = :Old_OrderID
```

See Also

- [SQL](#)

- [SQLInsert](#)

- [SQLUpdate](#)

- [SQLRefresh](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.2.26 SQLInsert Property

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Class

[TCustomDADataSet](#)

Syntax

```
property SQLInsert: TStrings;
```

Remarks

Use the SQLInsert property to specify the SQL statement that will be used when applying an insertion to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. Parameters prefixed with OLD_ allow using current values of fields prior to the actual operation.

To create a SQLInsert statement at design-time, use the query statements editor.

See Also

- SQL

- SQLUpdate

- SQLDelete

- SQLRefresh

© 1997-2025 Devart. All Rights Reserved. | Request Support | DAC Forum | Provide Feedback

5.11.1.5.2.27 SQLLock Property

Used to specify a SQL statement that will be used to perform a record lock.

Class

TCustomDADataSet

Syntax

```
property SQLLock: TStrings;
```

Remarks

Use the SQLLock property to specify a SQL statement that will be used to perform a record lock. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

To create a SQLLock statement at design-time, the use query statement editor.

See Also

- SQL

- SQLInsert

- SQLUpdate

- SQLDelete

- SQLRefresh

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.28 SQLRecCount Property

Used to specify the SQL statement that is used to get the record count when opening a dataset.

Class

`TCustomDADataSet`

Syntax

```
property SQLRecCount: TStrings;
```

Remarks

Use the SQLRecCount property to specify the SQL statement that is used to get the record count when opening a dataset. The SQL statement is used if the TDADataSetOptions.QueryRecCount property is True, and the TCustomDADataSet.FetchAll property is False. Is not used if the FetchAll property is True.

To create a SQLRecCount statement at design-time, use the query statements editor.

See Also

- SQLInsert

- SQLUpdate

- SQLDelete

- SQLRefresh

- TDADataSetOptions

- FetchingAll

Request Support   DAC Forum   Provide Feedback

5.11.1.5.2.29 SQLRefresh Property

Used to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Class

TCustomDADataSet

Syntax

```
property SQLRefresh: TStrings;
```

Remarks

Use the SQLRefresh property to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Different behavior is observed when the SQLRefresh property is assigned with a single WHERE clause that holds frequently altered search condition. In this case the WHERE clause from SQLRefresh is combined with the same clause of the SELECT statement in a SQL property and this final query is then sent to the database server.

To create a SQLRefresh statement at design-time, use the query statements editor.

Example

```
SELECT Shipname FROM Orders
    WHERE
        OrderID = :OrderID
```

See Also

- RefreshRecord

- SQL

- SQLInsert

- SQLUpdate

- SQLDelete

Request Support **DAC Forum** **Provide Feedback**

5.11.1.5.2.30 SQLUpdate Property

Used to specify a SQL statement that will be used when applying an update to a dataset.

Class

TCustomDADataSet

Syntax

```
property SQLUpdate: TStrings;
```

Remarks

Use the SQLUpdate property to specify a SQL statement that will be used when applying an update to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

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

Request Support **DAC Forum** **Provide Feedback**

5.11.1.5.2.31 UniDirectional Property

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

Class

`TCustomDADataSet`

Syntax

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

Remarks

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

Note: Pay attention to the specificity of using the FetchAll property=False

See Also

- TMyQuery.FetchAll

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3 Methods

Methods of the **TCustomDADataSet** class.

For a complete list of the **TCustomDADataSet** class members, see the TCustomDADataSet

[Members](#) topic.

Public

Name	Description
[AddWhere](#)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
[ApplyRange](#) (inherited from **[TMemDataSet](#)**)	Applies a range to the dataset.
[ApplyUpdates](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Writes dataset's pending cached updates to a database.
[BreakExec](#)	Breaks execution of the SQL statement on the server.
[CancelRange](#) (inherited from **[TMemDataSet](#)**)	Removes any ranges currently in effect for a dataset.
[CancelUpdates](#) (inherited from **[TMemDataSet](#)**)	Clears all pending cached updates from cache and restores dataset in its prior state.
[CloneCursor](#)	Shares data from another dataset.
[CommitUpdates](#) (inherited from **[TMemDataSet](#)**)	Clears the cached updates buffer.
[CreateBlobStream](#)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
[DeferredPost](#) (inherited from **[TMemDataSet](#)**)	Makes permanent changes to the database server.
[DeleteWhere](#)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
[EditRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Enables changing the ending value for an existing range.
[EditRangeStart](#) (inherited from **[TMemDataSet](#)**)	Enables changing the starting value for an existing range.

[Execute](#)	Overloaded. Executes a SQL statement on the server.
[Executing](#)	Indicates whether SQL statement is still being executed.
[Fetched](#)	Used to find out whether TCustomDADataSet has fetched all rows.
[Fetching](#)	Used to learn whether TCustomDADataSet is still fetching rows.
[FetchingAll](#)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
[FindKey](#)	Searches for a record which contains specified field values.
[FindMacro](#)	Finds a macro with the specified name.
[FindNearest](#)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
[FindParam](#)	Determines if a parameter with the specified name exists in a dataset.
[GetBlob](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
[GetDataType](#)	Returns internal field types defined in the MemData and accompanying modules.
[GetFieldObject](#)	Returns a multireference shared object from field.
[GetFieldPrecision](#)	Retrieves the precision of a number field.
[GetFieldScale](#)	Retrieves the scale of a number field.
[GetKeyFieldNames](#)	Provides a list of available

	key field names.
GetOrderBy	Retrieves an ORDER BY clause from a SQL statement.
GotoCurrent	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a macro with the specified name.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.

SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved	Determines if the SQL property value was saved to the BaseSQL property.
UnLock	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TCustomDADataSet Class

- TCustomDADataSet Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.1 AddWhere Method

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

Class

TCustomDADataSet

Syntax

```
procedure AddWhere(const Condition: string);
```

Parameters

Condition
  Holds the condition that will be added to the WHERE clause.

Remarks

Call the AddWhere method to add a condition to the WHERE clause of SELECT statement in the SQL property.

If SELECT has no WHERE clause, AddWhere creates it.

Note: the AddWhere method is implicitly called by RefreshRecord. The AddWhere method works for the SELECT statements only.

Note: the AddWhere method adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

See Also

- DeleteWhere

Request Support DAC Forum Provide Feedback

5.11.1.5.3.2 BreakExec Method

Breaks execution of the SQL statement on the server.

Class

TCustomDADataSet

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to break execution of the SQL statement on the server. Execution is broken by the KILL operator execution on server. It makes sense to call BreakExec only from another thread.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.3 CloneCursor Method

Shares data from another dataset.

Class

TCustomDADataSet

Syntax

```
procedure CloneCursor(Source: TMemDataSet; Reset: boolean = False;
KeepSettings: boolean = False); override;
```

Parameters

Source
  Specifies the dataset whose data will be shared.

Reset
  Defines how to set values for such events and properties: Filter, Filtered, FilterOptions, and OnFilterRecord; IndexName; MasterSource and MasterFields; ReadOnly; RemoteServer and ProviderName. If Reset is True, the properties are all set to their default values.

KeepSettings
  Outlines how to assign values for the same events and properties as those of the Reset parameter.

Remarks

Call CloneCursor to provide a dataset with data from another dataset. After calling the method, the internal data storage is identical to both datasets.

If Reset and KeepSettings are False, the values of the properties are configured to match the source dataset. If Reset is False and KeepSettings is True, the properties remain unmodified. In this scenario, the application needs to verify that the current indexes, filters, and related settings are consistent with the cloned data.

The CloneCursor method has the following restrictions:

* When the source dataset is closed, all cloned datasets will be closed too.

* The cloned dataset will have the read-only status.

* The source dataset must not be UniDirectional.

* It's not allowed to clone a dataset from another thread as the behavior will be undefined.

* Filter and IndexFieldNames must not be used in the cloned dataset.

* Applying Filter and/or IndexFieldNames in the source dataset will affect the contents of the cloned datasets.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.3.4 CreateBlobStream Method

Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

Class

[TCustomDADataSet](#)

Syntax

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode):
TStream; override;
```

Parameters

Field
  Holds the BLOB field for reading data from or writing data to from a stream.

Mode
  Holds the stream mode, for which the stream will be used.

Return Value
  The BLOB Stream.

Remarks

Call the CreateBlobStream method to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter. It must be a TBlobField component. You can specify whether the stream will be used for reading, writing, or updating the contents of the field with the Mode parameter.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.3.5 DeleteWhere Method

Removes WHERE clause from the SQL property and assigns the BaseSQL property.

Class

[TCustomDADataSet](#)

Syntax

```
procedure DeleteWhere;
```

Remarks

Call the DeleteWhere method to remove WHERE clause from the the SQL property and assign BaseSQL.

See Also

- [AddWhere](#)

- [BaseSQL](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.3.6 Execute Method

Executes a SQL statement on the server.

Class

[TCustomDADataSet](#)

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

Request Support   DAC Forum   Provide Feedback

Executes a SQL statement on the server.

Class

`TCustomDADataSet`

Syntax

```
procedure Execute; overload; virtual;
```

Remarks

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

Execute implicitly prepares SQL statement by calling the TCustomDADataSet.Prepare method if the TCustomDADataSet.Options option is set to True and the statement has not been prepared yet. To speed up the performance in case of multiple Execute calls, an application should call Prepare before calling the Execute method for the first time.

See Also

- TCustomDADataSet.AfterExecute

- TCustomDADataSet.Executing

- TCustomDADataSet.Prepare

Request Support   DAC Forum   Provide Feedback

Used to perform Batch operations .

Class

TCustomDADataSet

Syntax

```
procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

Iters
   Specifies the number of inserted rows.

Offset
   Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the Batch operations article for samples.

See Also

- Batch operations

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.7 Executing Method

Indicates whether SQL statement is still being executed.

Class

TCustomDADataSet

Syntax

```
function Executing: boolean;
```

Return Value

   True, if SQL statement is still being executed.

Remarks

Check Executing to learn whether TCustomDADataSet is still executing SQL statement.

5.11.1.5.3.8 Fetched Method

Used to find out whether TCustomDADataSet has fetched all rows.

Class

TCustomDADataSet

Syntax

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

Return Value

True, if all rows have been fetched.

Remarks

Call the Fetched method to find out whether TCustomDADataSet has fetched all rows.

See Also

- Fetching

5.11.1.5.3.9 Fetching Method

Used to learn whether TCustomDADataSet is still fetching rows.

Class

TCustomDADataSet

Syntax

```
function Fetching: boolean;
```

Return Value

True, if TCustomDADataSet is still fetching rows.

Remarks

Check Fetching to learn whether TCustomDADataSet is still fetching rows. Use the Fetching method if NonBlocking is True.

See Also

- Executing

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.10 FetchingAll Method

Used to learn whether TCustomDADataSet is fetching all rows to the end.

Class

TCustomDADataSet

Syntax

```
function FetchingAll: boolean;
```

Return Value

True, if TCustomDADataSet is fetching all rows to the end.

Remarks

Check FetchingAll to learn whether TCustomDADataSet is fetching all rows to the end.

See Also

- Executing

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.11 FindKey Method

Searches for a record which contains specified field values.

Class

TCustomDADataSet

Syntax

```
function FindKey(const KeyValues: array of System.TVarRec):
Boolean;
```

Parameters

KeyValues
   Holds a key.

Remarks

Call the FindKey method to search for a specific record in a dataset. KeyValues holds a comma-delimited array of field values, that is called a key.

This function is provided for BDE compatibility only. It is recommended to use functions TMemDataSet.Locate and TMemDataSet.LocateEx for the record search.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.12 FindMacro Method

Finds a macro with the specified name.

Class

TCustomDADataSet

Syntax

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

Parameters

Value
   Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

See Also

- TMacro

- Macros

- MacroByName

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.13 FindNearest Method

Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

Class

TCustomDADataSet

Syntax

```
procedure FindNearest(const KeyValues: array of System.TVarRec);
```

Parameters

KeyValues
  Holds the values of the record key fields to which the cursor should be moved.

Remarks

Call the FindNearest method to move the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, the cursor will not move.

This function is provided for BDE compatibility only. It is recommended to use functions TMemDataSet.Locate and TMemDataSet.LocateEx for the record search.

See Also

- TMemDataSet.Locate

- TMemDataSet.LocateEx

- FindKey

Request Support | DAC Forum | Provide Feedback

5.11.1.5.3.14 FindParam Method

Determines if a parameter with the specified name exists in a dataset.

Class

TCustomDADataSet

Syntax

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

Parameters

Value
  Holds the name of the param for which to search.

Return Value
  the TDAParam object for the specified Name. Otherwise it returns nil.

Remarks

Call the FindParam method to determine if a specified param component exists in a dataset. Name is the name of the param for which to search. If FindParam finds a param with a matching name, it returns a TDAParam object for the specified Name. Otherwise it returns nil.

See Also

- Params

- ParamByName

Request Support | DAC Forum | Provide Feedback

5.11.1.5.3.15 GetDataType Method

Returns internal field types defined in the MemData and accompanying modules.

Class

[TCustomDADataSet](#)

Syntax

```
function GetDataType(const FieldName: string): integer; virtual;
```

Parameters

FieldName
　Holds the name of the field.

Return Value

　internal field types defined in MemData and accompanying modules.

Remarks

Call the GetDataType method to return internal field types defined in the MemData and accompanying modules. Internal field data types extend the TFieldType type of VCL by specific database server data types. For example, ftString, ftFile, ftObject.

© 1997-2025
Devart. All Rights　　[Request Support](#)　　[DAC Forum](#)　　[Provide Feedback](#)
Reserved.

5.11.1.5.3.16 GetFieldObject Method

Returns a multireference shared object from field.

Class

[TCustomDADataSet](#)

Syntax

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

```
GetFieldObject(const FieldName: string): TSharedObject; overload;
```

Parameters

FieldName

Holds the field name.

Return Value

multireference shared object.

Remarks

Call the GetFieldObject method to return a multireference shared object from field. If field does not hold one of the TSharedObject descendants, GetFieldObject raises an exception.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.17 GetFieldPrecision Method

Retrieves the precision of a number field.

Class

TCustomDADataSet

Syntax

```
function GetFieldPrecision(const FieldName: string): integer;
```

Parameters

FieldName

Holds the existing field name.

Return Value

precision of number field.

Remarks

Call the GetFieldPrecision method to retrieve the precision of a number field. FieldName is the name of an existing field.

See Also

* GetFieldScale

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.18 GetFieldScale Method

Retrieves the scale of a number field.

Class

[TCustomDADataSet](#)

Syntax

```
function GetFieldScale(const FieldName: string): integer;
```

Parameters

FieldName
  Holds the existing field name.

Return Value

  the scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also

* [GetFieldPrecision](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.3.19 GetKeyFieldNames Method

Provides a list of available key field names.

Class

[TCustomDADataSet](#)

Syntax

```
procedure GetKeyFieldNames(List: TStrings);
```

Parameters

List
  The list of available key field names
Return Value
  Key field name

Remarks

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also

- TCustomDAConnection.GetTableNames

- TCustomDAConnection.GetStoredProcNames

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.20 GetOrderBy Method

Retrieves an ORDER BY clause from a SQL statement.

Class

TCustomDADataSet

Syntax

```
function GetOrderBy: string;
```

Return Value
  an ORDER BY clause from the SQL statement.

Remarks

Call the GetOrderBy method to retrieve an ORDER BY clause from a SQL statement.

Note: GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

See Also

- SetOrderBy

5.11.1.5.3.21 GotoCurrent Method

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

Class

`TCustomDADataSet`

Syntax

```
procedure GotoCurrent(DataSet: TCustomDADataSet);
```

Parameters

DataSet
  Holds the TCustomDADataSet descendant to synchronize the record position with.

Remarks

Call the GotoCurrent method to set the current record in this dataset similar to the current record in another dataset. The key fields in both these DataSets must be coincident.

See Also

- TMemDataSet.Locate

- TMemDataSet.LocateEx

5.11.1.5.3.22 Lock Method

Locks the current record.

Class

`TCustomDADataSet`

Syntax

```
procedure Lock; virtual;
```

Remarks

Call the Lock method to lock the current record by executing the statement that is defined in the SQLLock property.

The Lock method sets the savepoint with the name LOCK_ + <component_name>.

See Also

- UnLock

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.23 MacroByName Method

Finds a macro with the specified name.

Class

TCustomDADataSet

Syntax

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

Parameters

Value
  Holds the name of a macro to search for.

Return Value
  TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

Example

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

See Also

- TMacro

- Macros

- FindMacro

© 1997-2025
Devart. All Rights
Reserved. | Request Support | DAC Forum | Provide Feedback

5.11.1.5.3.24 ParamByName Method

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

Class

TCustomDADataSet

Syntax

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

Parameters

Value
  Holds the name of the parameter for which to retrieve information.

Return Value
  a TDAParam object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a TDAParam object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an

edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See Also
- Params

- FindParam

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.25 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDADataSet

Syntax

```
procedure Prepare; override;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The MySQL prepared protocol has certain server restrictions, and its work is not always stable. That is why it is advisable to perform test before using preparation in production versions of applications.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also
- TMemDataSet.Prepared

- TMemDataSet.UnPrepare

- Options

5.11.1.5.3.26 RefreshRecord Method

Actualizes field values for the current record.

Class

TCustomDADataSet

Syntax

```
procedure RefreshRecord;
```

Remarks

Call the RefreshRecord method to actualize field values for the current record.
RefreshRecord performs query to database and refetches new field values from the returned cursor.

See Also

- RefreshOptions

- SQLRefresh

5.11.1.5.3.27 RestoreSQL Method

Restores the SQL property modified by AddWhere and SetOrderBy.

Class

TCustomDADataSet

Syntax

```
procedure RestoreSQL;
```

Remarks

Call the RestoreSQL method to restore the SQL property modified by AddWhere and SetOrderBy.

See Also
- AddWhere

- SetOrderBy

- SaveSQL

- SQLSaved

Request Support DAC Forum Provide Feedback

5.11.1.5.3.28 SaveSQL Method

Saves the SQL property value to BaseSQL.

Class
TCustomDADataSet

Syntax
```
procedure SaveSQL;
```

Remarks
Call the SaveSQL method to save the SQL property value to the BaseSQL property.

See Also
- SQLSaved

- RestoreSQL

- BaseSQL

Request Support DAC Forum Provide Feedback

5.11.1.5.3.29 SetOrderBy Method

Builds an ORDER BY clause of a SELECT statement.

Class

TCustomDADataSet

Syntax

```
procedure SetOrderBy(const Fields: string);
```

Parameters

Fields
  Holds the names of the fields which will be added to the ORDER BY clause.

Remarks

Call the SetOrderBy method to build an ORDER BY clause of a SELECT statement. The fields are identified by the comma-delimited field names.

Note: The GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

Example

```
Query1.SetOrderBy('DeptNo;DName');
```

See Also

- GetOrderBy

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.30 SQLSaved Method

Determines if the SQL property value was saved to the BaseSQL property.

Class

TCustomDADataSet

Syntax

```
function SQLSaved: boolean;
```

Return Value

True, if the SQL property value was saved to the BaseSQL property.

Remarks

Call the SQLSaved method to know whether the SQL property value was saved to the BaseSQL property.

Request Support   DAC Forum   Provide Feedback

5.11.1.5.3.31 UnLock Method

Releases a record lock.

Class

TCustomDADataSet

Syntax

```
procedure UnLock;
```

Remarks

Call the Unlock method to release the record lock made by the Lock method before.

Unlock is performed by rolling back to the savepoint set by the Lock method.

See Also

* Lock

Request Support   DAC Forum   Provide Feedback

5.11.1.5.4 Events

Events of the **TCustomDADataSet** class.

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

Public

Name	Description
[AfterExecute](#)	Occurs after a component has executed a query to database.
[AfterFetch](#)	Occurs after dataset finishes fetching data from server.
[AfterUpdateExecute](#)	Occurs after executing insert, delete, update, lock and refresh operations.
[BeforeFetch](#)	Occurs before dataset is going to fetch block of records from the server.
[BeforeUpdateExecute](#)	Occurs before executing insert, delete, update, lock, and refresh operations.
[OnUpdateError](#) (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
[OnUpdateRecord](#) (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

See Also

- [TCustomDADataSet Class](#)

- [TCustomDADataSet Class Members](#)

© 1997-2025
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.4.1 AfterExecute Event

Occurs after a component has executed a query to database.

Class

`TCustomDADataSet`

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a component has executed a query to database.

See Also

- TCustomDADataSet.Execute

Request Support   DAC Forum   Provide Feedback

5.11.1.5.4.2 AfterFetch Event

Occurs after dataset finishes fetching data from server.

Class

TCustomDADataSet

Syntax

```
property AfterFetch: TAfterFetchEvent;
```

Remarks

The AfterFetch event occurs after dataset finishes fetching data from server.

See Also

- BeforeFetch

Request Support   DAC Forum   Provide Feedback

5.11.1.5.4.3 AfterUpdateExecute Event

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

Class

TCustomDADataSet

Syntax

```
property AfterUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs after executing insert, delete, update, lock, and refresh operations. You can use AfterUpdateExecute to set the parameters of corresponding statements.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.4.4 BeforeFetch Event

Occurs before dataset is going to fetch block of records from the server.

Class

[TCustomDADataSet](#)

Syntax

```
property BeforeFetch: TBeforeFetchEvent;
```

Remarks

The BeforeFetch event occurs every time before dataset is going to fetch a block of records from the server. Set Cancel to True to abort current fetch operation.

See Also

- [AfterFetch](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.5.4.5 BeforeUpdateExecute Event

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

Class

[TCustomDADataSet](#)

Syntax

```
property BeforeUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs before executing insert, delete, update, lock, and refresh operations. You can use BeforeUpdateExecute to set the parameters of corresponding statements.

See Also

- AfterUpdateExecute

Request Support   DAC Forum   Provide Feedback

5.11.1.6 TCustomDASQL Class

A base class for components executing SQL statements that do not return result sets.

For a list of all members of this type, see TCustomDASQL members.

Unit

DBAccess

Syntax

```
TCustomDASQL = class(TComponent);
```

Remarks

TCustomDASQL is a base class that defines functionality for descendant classes which access database using SQL statements. Applications never use TCustomDASQL objects directly. Instead they use descendants of TCustomDASQL.

Use TCustomDASQL when client application must execute SQL statement or call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

Request Support   DAC Forum   Provide Feedback

5.11.1.6.1 Members

TCustomDASQL class overview.

Properties

Name	Description
[ChangeCursor](#)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
[Connection](#)	Used to specify a connection object to use to connect to a data store.
[Debug](#)	Used to display the statement that is being executed and the values and types of its parameters.
[FinalSQL](#)	Used to return a SQL statement with expanded macros.
[MacroCount](#)	Used to get the number of macros associated with the Macros property.
[Macros](#)	Makes it possible to change SQL queries easily.
[ParamCheck](#)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
[ParamCount](#)	Indicates the number of parameters in the Params property.
[Params](#)	Used to contain parameters for a SQL statement.
[ParamValues](#)	Used to get or set the values of individual field parameters that are identified by name.
[Prepared](#)	Used to indicate whether a query is prepared for execution.
[RowsAffected](#)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

[SQL](#)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Methods

Name	Description
[BreakExec](#)	Breaks execution of an SQL satatement on the server.
[Execute](#)	Overloaded. Executes a SQL statement on the server.
[Executing](#)	Checks whether TCustomDASQL still executes a SQL statement.
[FindMacro](#)	Finds a macro with the specified name.
[FindParam](#)	Finds a parameter with the specified name.
[MacroByName](#)	Finds a macro with the specified name.
[ParamByName](#)	Finds a parameter with the specified name.
[Prepare](#)	Allocates, opens, and parses cursor for a query.
[UnPrepare](#)	Frees the resources allocated for a previously prepared query on the server and client sides.
[WaitExecuting](#)	Waits until TCustomDASQL executes a SQL statement.

Events

Name	Description
[AfterExecute](#)	Occurs after a SQL statement has been executed.

5.11.1.6.2 Properties

Properties of the **TCustomDASQL** class.

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

[Members](#) topic.

Public

Name	Description
[ChangeCursor](#)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
[Connection](#)	Used to specify a connection object to use to connect to a data store.
[Debug](#)	Used to display the statement that is being executed and the values and types of its parameters.
[FinalSQL](#)	Used to return a SQL statement with expanded macros.
[MacroCount](#)	Used to get the number of macros associated with the Macros property.
[Macros](#)	Makes it possible to change SQL queries easily.
[ParamCheck](#)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
[ParamCount](#)	Indicates the number of parameters in the Params property.
[Params](#)	Used to contain parameters for a SQL statement.
[ParamValues](#)	Used to get or set the values of individual field parameters that are

	identified by name.
[Prepared](#)	Used to indicate whether a query is prepared for execution.
[RowsAffected](#)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
[SQL](#)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

See Also

- [TCustomDASQL Class](#)

- [TCustomDASQL Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.6.2.1 ChangeCursor Property

Enables or disables changing screen cursor when executing commands in the NonBlocking mode.

Class

`TCustomDASQL`

Syntax

```
property ChangeCursor: boolean;
```

Remarks

Set the ChangeCursor property to False to prevent the screen cursor from changing to crSQLArrow when executing commands in the NonBlocking mode. The default value is True.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.6.2.2 Connection Property

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

Class

TCustomDASQL

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

Request Support   DAC Forum   Provide Feedback

5.11.1.6.2.3 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

TCustomDASQL

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

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

Note: If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

See Also
- TCustomDADataSet.Debug

Request Support   DAC Forum   Provide Feedback

5.11.1.6.2.4 FinalSQL Property

Used to return a SQL statement with expanded macros.

Class

`TCustomDASQL`

Syntax

```
property FinalSQL: string;
```

Remarks

Read the FinalSQL property to return a SQL statement with expanded macros. This is the exact statement that will be passed on to the database server.

Request Support   DAC Forum   Provide Feedback

5.11.1.6.2.5 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

`TCustomDASQL`

Syntax

```
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- Macros

Request Support   DAC Forum   Provide Feedback

5.11.1.6.2.6 Macros Property

Makes it possible to change SQL queries easily.

Class

TCustomDASQL

Syntax

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

Remarks

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

See Also

- TMacro

- MacroByName

- Params

Request Support   DAC Forum   Provide Feedback

5.11.1.6.2.7 ParamCheck Property

Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

Class

[TCustomDASQL](#)

Syntax

```
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

Set ParamCheck to True to let TCustomDASQL generate the Params property for the dataset based on a SQL statement automatically.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of the stored procedures that will accept parameterized values themselves. The default value is True.

See Also

- [Params](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.6.2.8 ParamCount Property

Indicates the number of parameters in the Params property.

Class

[TCustomDASQL](#)

Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

Request Support   DAC Forum   Provide Feedback

5.11.1.6.2.9 Params Property

Used to contain parameters for a SQL statement.

Class

TCustomDASQL

Syntax

```
property Params: TDAParams stored False;
```

Remarks

Access the Params property at runtime to view and set parameter names, values, and data types dynamically (at design-time use the Parameters editor to set parameter properties). Params is a zero-based array of parameter records. Index specifies the array element to access. An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

Example

Setting parameters at runtime:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with MySQL do
 begin
    SQL.Clear;
    SQL.Add('INSERT INTO Temp_Table(Id, Name)');
    SQL.Add('VALUES (:id, :Name)');
    ParamByName('Id').AsInteger := 55;
    Params[1].AsString := ' Green';
    Execute;
  end;
end;
```

See Also

- [TDAParam](#)

- [FindParam](#)

- [Macros](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.6.2.10 ParamValues Property(Indexer)

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

Class

`TCustomDASQL`

Syntax

```
property ParamValues[const ParamName: string]: Variant; default;
```

Parameters

ParamName
  Holds parameter names separated by semicolon.

Remarks

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

Setting ParamValues sets the Value property for each parameter listed in the ParamName string. Specify the values as Variants.

Getting ParamValues retrieves an array of variants, each of which represents the value of one of the named parameters.

Note: The Params array is generated implicitly if ParamCheck property is set to True. If ParamName includes a name that does not match any of the parameters in Items, an exception is raised.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.6.2.11 Prepared Property

Used to indicate whether a query is prepared for execution.

Class

[TCustomDASQL](#)

Syntax

```
property Prepared: boolean;
```

Remarks

Check the Prepared property to determine if a query is already prepared for execution. True means that the query has already been prepared. As a rule prepared queries are executed faster, but the preparation itself also takes some time. One of the proper cases for using preparation is parametrized queries that are executed several times.

See Also

- [Prepare](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.6.2.12 Row sAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

[TCustomDASQL](#)

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.6.2.13 SQL Property

Used to provide a SQL statement that a TCustomDASQL component executes when the
Execute method is called.

Class

`TCustomDASQL`

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a TCustomDASQL component
executes when the Execute method is called. At design time the SQL property can be edited
by invoking the String List editor in Object Inspector.

See Also

- FinalSQL

- TCustomDASQL.Execute

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.6.3 Methods

Methods of the **TCustomDASQL** class.

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

Public

Name	Description
BreakExec	Breaks execution of an SQL satatement on the server.

Execute	Overloaded. Executes a SQL statement on the server.
Executing	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro	Finds a macro with the specified name.
FindParam	Finds a parameter with the specified name.
MacroByName	Finds a macro with the specified name.
ParamByName	Finds a parameter with the specified name.
Prepare	Allocates, opens, and parses cursor for a query.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting	Waits until TCustomDASQL executes a SQL statement.

See Also

- TCustomDASQL Class

- TCustomDASQL Class Members

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.11.1.6.3.1 BreakExec Method

Breaks execution of an SQL satatement on the server.

Class

TCustomDASQL

Syntax

```
procedure BreakExec;
```

Remarks

Call the BreakExec method to break execution of an SQL statement on the server. It makes sense to call BreakExec only from another thread. Useful when NonBlocking is True.

See Also

- [TCustomDASQL.Execute](#)

- [TCustomDADataSet.BreakExec](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.6.3.2 Execute Method

Executes a SQL statement on the server.

Class

`TCustomDASQL`

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

Request Support   **DAC Forum**   **Provide Feedback**

Executes a SQL statement on the server.

Class

`TCustomDASQL`

Syntax

```
procedure Execute; overload; virtual;
```

Remarks

Call the Execute method to execute a SQL statement on the server. If the SQL statement has OUT parameters, use the TCustomDASQL.ParamByName method or the TCustomDASQL.Params property to get their values. Iters argument is ignored.

Request Support   DAC Forum   Provide Feedback

Used to perform Batch operations .

Class

`TCustomDASQL`

Syntax

```
procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

Iters
  Specifies the number of inserted rows.
Offset
  Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the Batch operations article for samples.

See Also

- Batch operations

Request Support   DAC Forum   Provide Feedback

5.11.1.6.3.3 Executing Method

Checks whether TCustomDASQL still executes a SQL statement.

Class

`TCustomDASQL`

Syntax

```
function Executing: boolean;
```

Return Value

True, if a SQL statement is still being executed by TCustomDASQL.

Remarks

Check Executing to find out whether TCustomDASQL still executes a SQL statement.

[Request Support](.)   [DAC Forum](.)   [Provide Feedback](.)

5.11.1.6.3.4 FindMacro Method

Finds a macro with the specified name.

Class

[TCustomDASQL](.)

Syntax

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

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the [TMacros.Items](.) property to avoid depending on the order of the items.

See Also

- [TMacro](.)

- [Macros](.)

- [MacroByName](.)

5.11.1.6.3.5 FindParam Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

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

Parameters

Value
   Holds the parameter name to search for.

Return Value

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

Remarks

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

See Also

* ParamByName

5.11.1.6.3.6 MacroByName Method

Finds a macro with the specified name.

Class

TCustomDASQL

Syntax

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

Parameters

Value
  Holds the name of a macro to search for.

Return Value
  TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

See Also

- TMacro

- Macros

- FindMacro

Request Support   DAC Forum   Provide Feedback

5.11.1.6.3.7 ParamByName Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

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

Parameters

Value
  Holds the name of the parameter to search for.

Return Value

a TDAParam object, if a match was found. Otherwise, an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the specified name. If no parameter with the specified name found, an exception is raised.

Example

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

See Also

- FindParam

Request Support   DAC Forum   Provide Feedback

5.11.1.6.3.8 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDASQL

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The MySQL prepared protocol has certain server restrictions, and its work is not always stable. That is why it is advisable to perform test before using preparation in production versions of applications.

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

5.11.1.6.3.9 UnPrepare Method

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

Class

`TCustomDASQL`

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free resources allocated for a previously prepared query on the server and client sides.

See Also

- Prepare

5.11.1.6.3.10 WaitExecuting Method

Waits until TCustomDASQL executes a SQL statement.

Class

`TCustomDASQL`

Syntax

```
function WaitExecuting(TimeOut: integer = 0): boolean;
```

Parameters

TimeOut

Holds the time in seconds to wait while TCustomDASQL executes the SQL statement. Zero means infinite time.

Return Value

True, if the execution of a SQL statement was completed in the preset time.

Remarks

Call the WaitExecuting method to wait until TCustomDASQL executes a SQL statement.

See Also

- [Executing](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.6.4 Events

Events of the **TCustomDASQL** class.

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

Public

Name	Description
AfterExecute	Occurs after a SQL statement has been executed.

See Also

- TCustomDASQL Class

- TCustomDASQL Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.6.4.1 AfterExecute Event

Occurs after a SQL statement has been executed.

Class

[TCustomDASQL](#)

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a SQL statement has been executed. This event may be used for descendant components which use multithreaded environment.

See Also

- [TCustomDASQL.Execute](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.7 TCustomDAUpdateSQL Class

A base class for components that provide DML statements for more flexible control over data modifications.

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

Unit

[DBAccess](#)

Syntax

```
TCustomDAUpdateSQL = class(TComponent);
```

Remarks

TCustomDAUpdateSQL is a base class for components that provide DML statements for more flexible control over data modifications. Besides providing BDE compatibility, this component allows to associate a separate component for each update command.

See Also

- TCustomMyDataSet.UpdateObject

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.7.1 Members

TCustomDAUpdateSQL class overview.

Properties

Name	Description
DataSet	Used to hold a reference to the TCustomDADataSet object that is being updated.
DeleteObject	Provides ability to perform advanced adjustment of the delete operations.
DeleteSQL	Used when deleting a record.
InsertObject	Provides ability to perform advanced adjustment of insert operations.
InsertSQL	Used when inserting a record.
LockObject	Provides ability to perform advanced adjustment of lock operations.
LockSQL	Used to lock the current record.
ModifyObject	Provides ability to perform advanced adjustment of modify operations.
ModifySQL	Used when updating a record.
RefreshObject	Provides ability to perform advanced adjustment of refresh operations.
RefreshSQL	Used to specify an SQL statement that will be used for refreshing the current record by

	[TCustomDADataSet.RefreshRecord](#) procedure.
[SQL](#)	Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Methods

Name	Description
[Apply](#)	Sets parameters for a SQL statement and executes it to update a record.
[ExecSQL](#)	Executes a SQL statement.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.7.2 Properties

Properties of the **TCustomDAUpdateSQL** class.

For a complete list of the **TCustomDAUpdateSQL** class members, see the

[TCustomDAUpdateSQL Members](#) topic.

Public

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

Published

Name	Description
[DeleteObject](#)	Provides ability to perform advanced adjustment of the delete operations.

[DeleteSQL](#)	Used when deleting a record.
[InsertObject](#)	Provides ability to perform advanced adjustment of insert operations.
[InsertSQL](#)	Used when inserting a record.
[LockObject](#)	Provides ability to perform advanced adjustment of lock operations.
[LockSQL](#)	Used to lock the current record.
[ModifyObject](#)	Provides ability to perform advanced adjustment of modify operations.
[ModifySQL](#)	Used when updating a record.
[RefreshObject](#)	Provides ability to perform advanced adjustment of refresh operations.
[RefreshSQL](#)	Used to specify an SQL statement that will be used for refreshing the current record by [TCustomDADataSet.RefreshRecord](#) procedure.

See Also

- [TCustomDAUpdateSQL Class](#)

- [TCustomDAUpdateSQL Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.7.2.1 DataSet Property

Used to hold a reference to the TCustomDADataSet object that is being updated.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property DataSet: TCustomDADataSet;
```

Remarks

The DataSet property holds a reference to the TCustomDADataSet object that is being updated. Generally it is not used directly.

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.2 DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class

TCustomDAUpdateSQL

Syntax

```
property DeleteObject: TComponent;
```

Remarks

Assign SQL component or a TCustomMyDataSet descendant to this property to perform advanced adjustment of the delete operations. In some cases this can give some additional performance. Use the same principle to set the SQL property of an object as for setting the DeleteSQL property.

See Also

- DeleteSQL

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.3 DeleteSQL Property

Used when deleting a record.

Class

TCustomDAUpdateSQL

Syntax

```
property DeleteSQL: TStrings;
```

Remarks

Set the DeleteSQL property to a DELETE statement to use when deleting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class

TCustomDAUpdateSQL

Syntax

```
property InsertObject: TComponent;
```

Remarks

Assign SQL component or TCustomMyDataSet 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

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.5 InsertSQL Property

Used when inserting a record.

Class

TCustomDAUpdateSQL

Syntax

```
property InsertSQL: TStrings;
```

Remarks

Set the InsertSQL property to an INSERT INTO statement to use when inserting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.6 LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

TCustomDAUpdateSQL

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomMyDataSet descendant to this property to perform advanced adjustment of lock operations. In some cases that can give some additional performance. Set the SQL property of an object in the same way as used for the LockSQL property.

See Also

- LockSQL

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.7 LockSQL Property

Used to lock the current record.

Class

TCustomDAUpdateSQL

Syntax

```
property LockSQL: TStrings;
```

Remarks

Use the LockSQL property to lock the current record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class

TCustomDAUpdateSQL

Syntax

```
property ModifyObject: TComponent;
```

Remarks

Assign a SQL component or TCustomMyDataSet 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

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.9 ModifySQL Property

Used when updating a record.

Class

`TCustomDAUpdateSQL`

Syntax

```
property ModifySQL: TStrings;
```

Remarks

Set ModifySQL to an UPDATE statement to use when updating a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

Request Support   DAC Forum   Provide Feedback

5.11.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

`TCustomDAUpdateSQL`

Syntax

```
property RefreshObject: TComponent;
```

Remarks

Assign a SQL component or TCustomMyDataSet 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

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.7.2.11 RefreshSQL Property

Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.

Class

TCustomDAUpdateSQL

Syntax

```
property RefreshSQL: TStrings;
```

Remarks

Use the RefreshSQL property to specify a SQL statement that will be used for refreshing the current record by the TCustomDADataSet.RefreshRecord procedure.

You can assign to SQLRefresh a WHERE clause only. In such a case it is added to SELECT defined by the SQL property by TCustomDADataSet.AddWhere.

To create a RefreshSQL statement at design time, use the query statements editor.

See Also

- TCustomDADataSet.RefreshRecord

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.7.2.12 SQL Property(Indexer)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

Parameters

UpdateKind
   Specifies which of update SQL statements to return.

Remarks

Returns a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties, depending on the value of the UpdateKind index.

Request Support   DAC Forum   Provide Feedback

5.11.1.7.3 Methods

Methods of the **TCustomDAUpdateSQL** class.

For a complete list of the **TCustomDAUpdateSQL** class members, see the

[TCustomDAUpdateSQL Members](#) topic.

Public

Name	Description
Apply	Sets parameters for a SQL statement and executes it to update a record.
ExecSQL	Executes a SQL statement.

See Also

- [TCustomDAUpdateSQL Class](#)

- [TCustomDAUpdateSQL Class Members](#)

Request Support   DAC Forum   Provide Feedback

5.11.1.7.3.1 Apply Method

Sets parameters for a SQL statement and executes it to update a record.

Class

TCustomDAUpdateSQL

Syntax

```
procedure Apply(UpdateKind: TUpdateKind); virtual;
```

Parameters

UpdateKind
  Specifies which of update SQL statements to execute.

Remarks

Call the Apply method to set parameters for a SQL statement and execute it to update a record. UpdateKind indicates which SQL statement to bind and execute.

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

Note: If a SQL statement does not contain parameters, it is more efficient to call ExecSQL instead of Apply.

See Also
- ExecSQL

Request Support   DAC Forum   Provide Feedback

5.11.1.7.3.2 ExecSQL Method

Executes a SQL statement.

Class

TCustomDAUpdateSQL

Syntax

```
procedure ExecSQL(UpdateKind: TUpdateKind);
```

Parameters

UpdateKind
  Specifies the kind of update statement to be executed.

Remarks

Call the ExecSQL method to execute a SQL statement, necessary for updating the records belonging to a read-only result set when cached updates is enabled. UpdateKind specifies the statement to execute.

ExecSQL is primarily intended for manually executing update statements from the OnUpdateRecord event handler.

Note: To both bind parameters and execute a statement, call Apply.

See Also

* Apply

Request Support   DAC Forum   Provide Feedback

5.11.1.8 TDACondition Class

Represents a condition from the TDAConditions list.

For a list of all members of this type, see TDACondition members.

Unit

DBAccess

Syntax

```
TDACondition = class(TCollectionItem);
```

Remarks

Manipulate conditions using TDAConditions.

See Also

* TDAConditions

Request Support   DAC Forum   Provide Feedback

5.11.1.8.1 Members

[TDACondition](#) class overview.

Properties

Name	Description
[Enabled](#)	Indicates whether the condition is enabled or not
[Name](#)	The name of the condition
[Value](#)	The value of the condition

Methods

Name	Description
[Disable](#)	Disables the condition
[Enable](#)	Enables the condition

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.8.2 Properties

Properties of the **TDACondition** class.

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

Published

Name	Description
[Enabled](#)	Indicates whether the condition is enabled or not
[Name](#)	The name of the condition
[Value](#)	The value of the condition

See Also

- [TDACondition Class](#)

- [TDACondition Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

[TDACondition](#)

Syntax

```
property Enabled: Boolean default True;
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.8.2.2 Name Property

The name of the condition

Class

[TDACondition](#)

Syntax

```
property Name: string;
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.8.2.3 Value Property

The value of the condition

Class

[TDACondition](#)

Syntax

```
property Value: string;
```

Request Support   DAC Forum   Provide Feedback

5.11.1.8.3 Methods

Methods of the **TDACondition** class.

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

Public

Name	Description
Disable	Disables the condition
Enable	Enables the condition

See Also

- TDACondition Class

- TDACondition Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.8.3.1 Disable Method

Disables the condition

Class

[TDACondition](#)

Syntax

```
procedure Disable;
```

5.11.1.8.3.2 Enable Method

Enables the condition

Class

`TDACondition`

Syntax

```
procedure Enable;
```

5.11.1.9 TDAConditions Class

Holds a collection of TDACondition objects.

For a list of all members of this type, see TDAConditions members.

Unit

`DBAccess`

Syntax

```
TDAConditions = class(TCollection);
```

Remarks

The given example code

```
UniTable1.Conditions.Add('1','JOB="MANAGER"');
UniTable1.Conditions.Add('2','SAL>2500');
UniTable1.Conditions.Enable;
UniTable1.Open;
```

will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
(SAL<2500)
```

5.11.1.9.1 Members

[TDAConditions](#) class overview.

Properties

Name	Description
[Condition](#)	Used to iterate through all the conditions.
[Enabled](#)	Indicates whether the condition is enabled
[Items](#)	Used to iterate through all conditions.
[Text](#)	The property returns condition names and values as CONDITION_NAME=COND ITION
[WhereSQL](#)	Returns the SQL WHERE condition added in the Conditions property.

Methods

Name	Description
[Add](#)	Overloaded. Adds a condition to the WHERE clause of the query.
[Delete](#)	Deletes the condition
[Disable](#)	Disables the condition
[Enable](#)	Enables the condition
[Find](#)	Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.
[Get](#)	Retrieving a TDACondition object by its name. If found, the TDACondition object is

	returned, otherwise - an exception is raised.
[IndexOf](#)	Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.
[Remove](#)	Removes the condition

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.9.2 Properties

Properties of the **TDAConditions** class.

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

Public

Name	Description
[Condition](#)	Used to iterate through all the conditions.
[Enabled](#)	Indicates whether the condition is enabled
[Items](#)	Used to iterate through all conditions.
[Text](#)	The property returns condition names and values as CONDITION_NAME=CONDITION
[WhereSQL](#)	Returns the SQL WHERE condition added in the Conditions property.

See Also

- [TDAConditions Class](#)

- [TDAConditions Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.9.2.1 Condition Property(Indexer)

Used to iterate through all the conditions.

Class

TDAConditions

Syntax

```
property Condition[Index: Integer]: TDACondition;
```

Parameters

Index

Request Support DAC Forum Provide Feedback

5.11.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

TDAConditions

Syntax

```
property Enabled: Boolean;
```

Request Support DAC Forum Provide Feedback

5.11.1.9.2.3 Items Property(Indexer)

Used to iterate through all conditions.

Class

TDAConditions

Syntax

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

Parameters

Index

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

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.11.1.9.2.4 Text Property

The property returns condition names and values as CONDITION_NAME=CONDITION

Class

TDAConditions

Syntax

```
property Text: string;
```

Request Support   DAC Forum   Provide Feedback

5.11.1.9.2.5 WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class

TDAConditions

Syntax

```
property WhereSQL: string;
```

Request Support   DAC Forum   Provide Feedback

5.11.1.9.3 Methods

Methods of the **TDAConditions** class.

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

Public

Name	Description
Add	Overloaded. Adds a condition to the WHERE clause of the query.
Delete	Deletes the condition
Disable	Disables the condition
Enable	Enables the condition
Find	Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.
Get	Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.
IndexOf	Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.
Remove	Removes the condition

See Also

- TDAConditions Class

- TDAConditions Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.9.3.1 Add Method

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Overload List

Name	Description
Add(**const** Value: **string**; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.
Add(**const** Name: **string**; **const** Value: **string**; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.

Request Support   DAC Forum   Provide Feedback

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Syntax

```
function Add(const Value: string; Enabled: Boolean = True):
TDACondition; overload;
```

Parameters

Value
   The value of the condition

Enabled
   Indicates that the condition is enabled

Remarks

If you want then to access the condition, you should use Add and its name in the Name parameter.

The given example code will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
```

```
and
(SAL<2500)
```

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Syntax

```
function Add(const Name: string; const Value: string; Enabled:
Boolean = True): TDACondition; overload;
```

Parameters

Name
   Sets the name of the condition

Value
   The value of the condition

Enabled
   Indicates that the condition is enabled

Remarks

The given example code will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
(SAL<2500)
```

5.11.1.9.3.2 Delete Method

Deletes the condition

Class

TDAConditions

Syntax

```
procedure Delete(Index: integer);
```

Parameters

Index

Index of the condition

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.9.3.3 Disable Method

Disables the condition

Class

`TDAConditions`

Syntax

```
procedure Disable;
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.9.3.4 Enable Method

Enables the condition

Class

`TDAConditions`

Syntax

```
procedure Enable;
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.9.3.5 Find Method

Search for TDACondition (the condition) by its name. If found, the TDACondition object is
returned, otherwise - nil.

Class

[TDAConditions](#)

Syntax

```
function Find(const Name: string): TDACondition;
```

Parameters

Name

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.9.3.6 Get Method

Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.

Class

[TDAConditions](#)

Syntax

```
function Get(const Name: string): TDACondition;
```

Parameters

Name

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.9.3.7 IndexOf Method

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

Class

[TDAConditions](#)

Syntax

```
function IndexOf(const Name: string): Integer;
```

Parameters

Name

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.9.3.8 Remove Method

Removes the condition

Class

[TDAConditions](#)

Syntax

```
procedure Remove(const Name: string);
```

Parameters

Name
  Specifies the name of the removed condition

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.10 TDAConnectionOptions Class

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

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

Unit

[DBAccess](#)

Syntax

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.10.1 Members

TDAConnectionOptions class overview.

Properties

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.
KeepDesignConnected	Prevents 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 the connection is lost.

Request Support   DAC Forum   Provide Feedback

5.11.1.10.2 Properties

Properties of the **TDAConnectionOptions** class.

For a complete list of the **TDAConnectionOptions** class members, see the

TDAConnectionOptions Members topic.

Public

Name	Description
DefaultSortType	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.
KeepDesignConnected	Prevents 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 the connection is lost.

Published

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.

See Also

- TDAConnectionOptions Class

- TDAConnectionOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.10.2.1 Allow ImplicitConnect Property

Specifies whether to allow or not implicit connection opening.

Class

TDAConnectionOptions

Syntax

```
property AllowImplicitConnect: boolean default True;
```

Remarks

Use the AllowImplicitConnect property to specify whether allow or not implicit connection opening.

If a closed connection has AllowImplicitConnect set to True and a dataset that uses the connection is opened, the connection is opened implicitly to allow opening the dataset.

If a closed connection has AllowImplicitConnect set to False and a dataset that uses the connection is opened, an exception is raised.

The default value is True.

Request Support   DAC Forum   Provide Feedback

5.11.1.10.2.2 DefaultSortType Property

Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set 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.

Request Support   DAC Forum   Provide Feedback

5.11.1.10.2.3 DisconnectedMode Property

Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.

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.

Request Support   DAC Forum   Provide Feedback

5.11.1.10.2.4 KeepDesignConnected Property

Prevents an application from establishing a connection at the time of startup.

Class

TDAConnectionOptions

Syntax

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

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.11.1.10.2.5 LocalFailover Property

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

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.

5.11.1.11 TDAConnectionSSLOptions Class

This class is used to set up the SSL options.

For a list of all members of this type, see TDAConnectionSSLOptions members.

Unit

DBAccess

Syntax

```
TDAConnectionSSLOptions = class(TPersistent);
```

5.11.1.11.1 Members

TDAConnectionSSLOptions class overview.

Properties

Name	Description
[CACert](#)	Holds the path to the certificate authority file.
[Cert](#)	Holds the path to the client certificate.
[CipherList](#)	Holds the list of allowed SSL ciphers.
[Key](#)	Holds the path to the private client key.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.11.2 Properties

Properties of the **TDAConnectionSSLOptions** class.

For a complete list of the **TDAConnectionSSLOptions** class members, see the

[TDAConnectionSSLOptions Members](#) topic.

Published

Name	Description
[CACert](#)	Holds the path to the certificate authority file.
[Cert](#)	Holds the path to the client certificate.
[CipherList](#)	Holds the list of allowed SSL ciphers.
[Key](#)	Holds the path to the private client key.

See Also

- [TDAConnectionSSLOptions Class](#)

- [TDAConnectionSSLOptions Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.11.2.1 CACert Property

Holds the path to the certificate authority file.

Class

TDAConnectionSSLOptions

Syntax

```
property CACert: string;
```

Remarks

Use the CACert property to specify the path to the certificate authority file.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.11.2.2 Cert Property

Holds the path to the client certificate.

Class

TDAConnectionSSLOptions

Syntax

```
property Cert: string;
```

Remarks

Use the Cert property to specify the path to the client certificate.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.11.2.3 CipherList Property

Holds the list of allowed SSL ciphers.

Class

TDAConnectionSSLOptions

Syntax

```
property CipherList: string;
```

Remarks

Use the CipherList property to specify the list of allowed SSL ciphers.

Request Support DAC Forum Provide Feedback

5.11.1.11.2.4 Key Property

Holds the path to the private client key.

Class

TDAConnectionSSLOptions

Syntax

```
property Key: string;
```

Remarks

Use the Key property to specify the path to the private client key.

Request Support DAC Forum Provide Feedback

5.11.1.12 TDADataSetOptions Class

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

For a list of all members of this type, see TDADataSetOptions members.

Unit

DBAccess

Syntax

```
TDADataSetOptions = class(TPersistent);
```

Request Support DAC Forum Provide Feedback

5.11.1.12.1 Members

TDADataSetOptions class overview.

Properties

Name	Description
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataSet to use

	local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.

[SetFieldsReadOnly](#)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
[UpdateAllFields](#)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
[UpdateBatchSize](#)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.12.2 Properties

Properties of the **TDADataSetOptions** class.

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

Public

Name	Description
[AutoPrepare](#)	Used to execute automatic [TCustomDADataSet.Prepare](#) on the query execution.
[CacheCalcFields](#)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
[CompressBlobMode](#)	Used to store values of the BLOB fields in compressed form.

[DefaultValues](#)	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
[DetailDelay](#)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
[FieldsOrigin](#)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
[FlatBuffers](#)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
[InsertAllSetFields](#)	Used to include all set dataset fields in the generated INSERT statement
[LocalMasterDetail](#)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
[LongStrings](#)	Used to represent string fields with the length that is greater than 255 as TStringField.
[MasterFieldsNullable](#)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
[NumberRange](#)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate

	values.
[QueryRecCount](#)	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
[QuoteNames](#)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
[RemoveOnRefresh](#)	Used for a dataset to locally remove a record that can not be found on the server.
[RequiredFields](#)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
[ReturnParams](#)	Used to return the new value of fields to dataset after insert or update.
[SetFieldsReadOnly](#)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
[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

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.1 AutoPrepare Property

Used to execute automatic TCustomDADataSet.Prepare on the query execution.

Class

TDADataSetOptions

Syntax

```
property AutoPrepare: boolean default False;
```

Remarks

Use the AutoPrepare property to execute automatic TCustomDADataSet.Prepare on the query execution. Makes sense for cases when a query will be executed several times, for example, in Master/Detail relationships.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.2 CacheCalcFields Property

Used to enable caching of the TField.Calculated and TField.Lookup fields.

Class

TDADataSetOptions

Syntax

```
property CacheCalcFields: boolean default False;
```

Remarks

Use the CacheCalcFields property to enable caching of the TField.Calculated and TField.Lookup fields. It can be useful for reducing CPU usage for calculated fields. Using caching of calculated and lookup fields increases memory usage on the client side.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.3 CompressBlobMode Property

Used to store values of the BLOB fields in compressed form.

Class

TDADataSetOptions

Syntax

```
property CompressBlobMode: TCompressBlobMode default cbNone;
```

Remarks

Use the CompressBlobMode property to store values of the BLOB fields in compressed form. Add the MemData unit to uses list to use this option. Compression rate greatly depends on stored data, for example, usually graphic data compresses badly unlike text.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.4 DefaultValues Property

Used to request default values/expressions from the server and assign them to the DefaultExpression property.

Class

TDADataSetOptions

Syntax

```
property DefaultValues: boolean default False;
```

Remarks

If True, the default values/expressions are requested from the server and assigned to the DefaultExpression property of TField objects replacing already existent values.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.5 DetailDelay Property

Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.

Class

TDADataSetOptions

Syntax

```
property DetailDelay: integer default 0;
```

Remarks

Use the DetailDelay property to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset. If DetailDelay is 0 (the default value) then refreshing of detail dataset occurs immediately. The DetailDelay option should be used for detail dataset.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.6 FieldsOrigin Property

Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.

Class

TDADataSetOptions

Syntax

```
property FieldsOrigin: boolean;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.7 FlatBuffers Property

Used to control how a dataset treats data of the ftString and ftVarBytes fields.

Class

TDADataSetOptions

Syntax

```
property FlatBuffers: boolean default False;
```

Remarks

Use the FlatBuffers property to control how a dataset treats data of the ftString and ftVarBytes fields. When set to True, all data fetched from the server is stored in record pdata without unused tails.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.8 InsertAllSetFields Property

Used to include all set dataset fields in the generated INSERT statement

Class

TDADataSetOptions

Syntax

```
property InsertAllSetFields: boolean default False;
```

Remarks

If True, all set dataset fields, including those set to NULL explicitly, will be included in the

generated INSERT statements. Otherwise, only set fields containing not NULL values will be included to the generated INSERT statement.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.9 LocalMasterDetail Property

Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.

Class

TDADataSetOptions

Syntax

```
property LocalMasterDetail: boolean default False;
```

Remarks

If True, for detail dataset in master-detail relationship TCustomDADataSet uses local filtering for establishing master/detail relationship and does not refer to the server. Otherwise detail dataset performs query each time a record is selected in master dataset. This option is useful for reducing server calls number, server resources economy. It can be useful for slow connection. The TMemDataSet.CachedUpdates mode can be used for detail dataset only when this option is set to true. Setting the LocalMasterDetail option to True is not recommended when detail table contains too many rows, because when it is set to False, only records that correspond to the current record in master dataset are fetched.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.10 LongStrings Property

Used to represent string fields with the length that is greater than 255 as TStringField.

Class

TDADataSetOptions

Syntax

```
property LongStrings: boolean default True;
```

Remarks

Use the LongStrings property to represent string fields with the length that is greater than 255 as TStringField, not as TMemoField.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.11 MasterFieldsNullable Property

Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).

Class

TDADataSetOptions

Syntax

```
property MasterFieldsNullable: boolean default False;
```

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

TDADataSetOptions

Syntax

```
property NumberRange: boolean default False;
```

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.13 QueryRecCount Property

Used for TCustomDADataSet to perform additional query to get the record count for this
SELECT, so the RecordCount property reflects the actual number of records.

Class

TDADataSetOptions

Syntax

```
property QueryRecCount: boolean default False;
```

Remarks

If True, and the FetchAll property is False, TCustomDADataSet performs additional query to
get the record count for this SELECT, so the RecordCount property reflects the actual
number of records. Does not have any effect if the FetchAll property is True.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.14 QuoteNames Property

Used for TCustomDADataSet to quote all database object names in autogenerated SQL
statements such as update SQL.

Class

TDADataSetOptions

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If True, TCustomDADataSet quotes all database object names in autogenerated SQL
statements such as update SQL.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.15 RemoveOnRefresh Property

Used for a dataset to locally remove a record that can not be found on the server.

Class

[TDADataSetOptions](#)

Syntax

```
property RemoveOnRefresh: boolean default True;
```

Remarks

When the RefreshRecord procedure can't find necessary record on the server and RemoveOnRefresh is set to True, dataset removes the record locally. Usually RefreshRecord can't find necessary record when someone else dropped the record or changed the key value of it.

This option makes sense only if the StrictUpdate option is set to False. If the StrictUpdate option is True, error will be generated regardless of the RemoveOnRefresh option value.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.12.2.16 RequiredFields Property

Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.

Class

[TDADataSetOptions](#)

Syntax

```
property RequiredFields: boolean default True;
```

Remarks

If True, TCustomDADataSet sets the Required property of the TField objects for the NOT

NULL fields. It is useful when table has a trigger which updates the NOT NULL fields.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.17 ReturnParams Property

Used to return the new value of fields to dataset after insert or update.

Class

TDADataSetOptions

Syntax

```
property ReturnParams: boolean default False;
```

Remarks

Use the ReturnParams property to return the new value of fields to dataset after insert or update. The actual value of field after insert or update may be different from the value stored in the local memory if the table has a trigger. When ReturnParams is True, OUT parameters of the SQLInsert and SQLUpdate statements is assigned to the corresponding fields.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.18 SetFieldsReadOnly Property

Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

Class

TDADataSetOptions

Syntax

```
property SetFieldsReadOnly: boolean default True;
```

Remarks

If True, dataset sets the ReadOnly property to True for all fields that do not belong to

UpdatingTable or can not be updated. Set this option for datasets that use automatic generation of the update SQL statements only.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.12.2.19 StrictUpdate Property

Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.

Class

[TDADataSetOptions](#)

Syntax

```
property StrictUpdate: boolean default True;
```

Remarks

If True, TCustomDADataSet raises an exception when the number of updated or deleted records is not equal 1. Setting this option also causes the exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you execute SQL query, that doesn't return resultset.

Note: There can be problems if this option is set to True and triggers for UPDATE, DELETE, REFRESH commands that are defined for the table. So it is recommended to disable (set to False) this option with triggers.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.12.2.20 UpdateAllFields Property

Used to include all dataset fields in the generated UPDATE and INSERT statements.

Class

[TDADataSetOptions](#)

Syntax

```
property UpdateAllFields: boolean default False;
```

Remarks

If True, all dataset fields will be included in the generated UPDATE and INSERT statements. Unspecified fields will have NULL value in the INSERT statements. Otherwise, only updated fields will be included to the generated update statements.

Request Support   DAC Forum   Provide Feedback

5.11.1.12.2.21 UpdateBatchSize Property

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Class

TDADataSetOptions

Syntax

```
property UpdateBatchSize: Integer default 1;
```

Remarks

Use the UpdateBatchSize property to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch. Takes effect only when updating dataset in the TMemDataSet.CachedUpdates mode. The default value is 1.

Request Support   DAC Forum   Provide Feedback

5.11.1.13 TDAEncryption Class

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

For a list of all members of this type, see TDAEncryption members.

Unit

DBAccess

Syntax

```
TDAEncryption = class(TPersistent);
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.11.1.13.1 Members

TDAEncryption class overview.

Properties

Name	Description
Encryptor	Used to specify the encryptor class that will perform the data encryption.
Fields	Used to set field names for which encryption will be performed.

Request Support   DAC Forum   Provide Feedback

5.11.1.13.2 Properties

Properties of the **TDAEncryption** class.

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

Public

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

Published

Name	Description
[Fields](#)	Used to set field names for which encryption will be performed.

See Also

- [TDAEncryption Class](#)

- [TDAEncryption Class Members](#)

Request Support   DAC Forum   Provide Feedback

5.11.1.13.2.1 Encryptor Property

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

Class

[TDAEncryption](#)

Syntax

```
property Encryptor: TCREncryptor;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.11.1.13.2.2 Fields Property

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

Class

[TDAEncryption](#)

Syntax

```
property Fields: string;
```

Remarks

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

Request Support DAC Forum Provide Feedback

5.11.1.14 TDAMapRule Class

Class that formes rules for Data Type Mapping.

For a list of all members of this type, see TDAMapRule members.

Unit

DBAccess

Syntax

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

Remarks

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

Inheritance Hierarchy

TMapRule

  TDAMapRule

Request Support DAC Forum Provide Feedback

5.11.1.14.1 Members

TDAMapRule class overview.

Properties

Name	Description
DBLengthMax	Maximum DB field length, until which the rule is applied.
DBLengthMin	Minimum DB field length, starting from which the rule is applied.
DBScaleMax	Maximum DB field scale, until which the rule is applied to the specified DB field.
DBScaleMin	Minimum DB field Scale, starting from which the rule is applied to the specified DB field.
DBType	DB field type, that the rule is applied to.
FieldLength	The resultant field length in Delphi.
FieldName	DataSet field name, for which the rule is applied.
FieldScale	The resultant field Scale in Delphi.
FieldType	Delphi field type, that the specified DB type or DataSet field will be mapped to.
IgnoreErrors	Ignoring errors when converting data from DB to Delphi type.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.14.2 Properties

Properties of the **TDAMapRule** class.

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

Published

Name	Description

DBLengthMax	Maximum DB field length, until which the rule is applied.
DBLengthMin	Minimum DB field length, starting from which the rule is applied.
DBScaleMax	Maximum DB field scale, until which the rule is applied to the specified DB field.
DBScaleMin	Minimum DB field Scale, starting from which the rule is applied to the specified DB field.
DBType	DB field type, that the rule is applied to.
FieldLength	The resultant field length in Delphi.
FieldName	DataSet field name, for which the rule is applied.
FieldScale	The resultant field Scale in Delphi.
FieldType	Delphi field type, that the specified DB type or DataSet field will be mapped to.
IgnoreErrors	Ignoring errors when converting data from DB to Delphi type.

See Also

- TDAMapRule Class

- TDAMapRule Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.1 DBLengthMax Property

Maximum DB field length, until which the rule is applied.

Class

TDAMapRule

Syntax

```
property DBLengthMax: Integer default rlAny;
```

Remarks

Setting maximum DB field length, until which the rule is applied to the specified DB field.

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class

TDAMapRule

Syntax

```
property DBLengthMin: Integer default rlAny;
```

Remarks

Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.3 DBScaleMax Property

Maximum DB field scale, until which the rule is applied to the specified DB field.

Class

TDAMapRule

Syntax

```
property DBScaleMax: Integer default rlAny;
```

Remarks

Setting maximum DB field scale, until which the rule is applied to the specified DB field.

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.4 DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class

TDAMapRule

Syntax

```
property DBScaleMin: Integer default rlAny;
```

Remarks

Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.5 DBType Property

DB field type, that the rule is applied to.

Class

TDAMapRule

Syntax

```
property DBType: Word default dtUnknown;
```

Remarks

Setting DB field type, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields of the specified type in all DataSets related to this Connection.

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.6 FieldLength Property

The resultant field length in Delphi.

Class

TDAMapRule

Syntax

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

Remarks

Setting the Delphi field length after conversion.

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.7 FieldName Property

DataSet field name, for which the rule is applied.

Class

TDAMapRule

Syntax

```
property FieldName: string;
```

Remarks

Specifies the DataSet field name, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields with such name in DataSets related to this Connection.

Request Support   DAC Forum   Provide Feedback

5.11.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class

TDAMapRule

Syntax

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

Remarks

Setting the Delphi field Scale after conversion.

Request Support DAC Forum Provide Feedback

5.11.1.14.2.9 FieldType Property

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

Class

TDAMapRule

Syntax

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

Remarks

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

Request Support DAC Forum Provide Feedback

5.11.1.14.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class

[TDAMapRule](#)

Syntax

```
property IgnoreErrors: Boolean default False;
```

Remarks

Allows to ignore errors while data conversion in case if data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.15 TDAMapRules Class

Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.

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

Unit

[DBAccess](#)

Syntax

```
TDAMapRules = class(TMapRules);
```

Inheritance Hierarchy

TMapRules

   TDAMapRules

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.15.1 Members

[**TDAMapRules**](#) class overview.

Properties

Name	Description
[IgnoreInvalidRules](#)	Used to avoid raising exception on mapping rules that can't be applied.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.15.2 Properties

Properties of the **TDAMapRules** class.

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

Published

Name	Description
[IgnoreInvalidRules](#)	Used to avoid raising exception on mapping rules that can't be applied.

See Also

- [TDAMapRules Class](#)

- [TDAMapRules Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.15.2.1 IgnoreInvalidRules Property

Used to avoid raising exception on mapping rules that can't be applied.

Class

[TDAMapRules](#)

Syntax

```
property IgnoreInvalidRules: boolean default False;
```

Remarks

Allows to ignore errors (not to raise exception) during data conversion in case if the data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

Note: In order to ignore errors occurring during data conversion, use the TDAMapRule.IgnoreErrors property

See Also

- TDAMapRule.IgnoreErrors

Request Support   DAC Forum   Provide Feedback

5.11.1.16 TDAMetaData Class

A class for retrieving metainformation of the specified database objects in the form of dataset.

For a list of all members of this type, see TDAMetaData members.

Unit

`DBAccess`

Syntax

```
TDAMetaData = class(TMemDataSet);
```

Remarks

TDAMetaData is a TDataSet descendant standing for retrieving metainformation of the specified database objects in the form of dataset. First of all you need to specify which kind of metainformation you want to see. For this you need to assign the TDAMetaData.MetaDataKind property. Provide one or more conditions in the TDAMetaData.Restrictions property to diminish the size of the resultset and get only information you are interested in.

Use the TDAMetaData.GetMetaDataKinds method to get the full list of supported kinds of meta data. With the TDAMetaData.GetRestrictions method you can find out what restrictions are applicable to the specified MetaDataKind.

Example

The code below demonstrates how to get information about columns of the 'emp' table:

```
MetaData.Connection := Connection;
MetaData.MetaDataKind := 'Columns';
MetaData.Restrictions.Values['TABLE_NAME'] := 'Emp';
MetaData.Open;
```

Inheritance Hierarchy

[TMemDataSet](#)

   TDAMetaData

See Also

- [TDAMetaData.MetaDataKind](#)

- [TDAMetaData.Restrictions](#)

- [TDAMetaData.GetMetaDataKinds](#)

- [TDAMetaData.GetRestrictions](#)

[Request Support](#)　　　[DAC Forum](#)　　　[Provide Feedback](#)

5.11.1.16.1 Members

[**TDAMetaData**](#) class overview.

Properties

Name	Description
[CachedUpdates](#) (inherited from [**TMemDataSet**](#))	Used to enable or disable the use of cached updates for a dataset.
[Connection](#)	Used to specify a connection object to use to connect to a data store.
[IndexFieldNames](#) (inherited from [**TMemDataSet**](#))	Used to get or set the list of fields on which the recordset is sorted.
[KeyExclusive](#) (inherited from [**TMemDataSet**](#))	Specifies the upper and lower boundaries for a range.

LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MetaDataKind	Used to specify which kind of metainformation to show.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Restrictions	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.

EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds	Used to get values acceptable in the MetaDataKind property.
GetRestrictions	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

5.11.1.16.2 Properties

Properties of the **TDAMetaData** class.

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

Public

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Connection	Used to specify a connection object to use to connect to a data store.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MetaDataKind	Used to specify which kind of metainformation to show.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Restrictions	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.

UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

See Also

- TDAMetaData Class

- TDAMetaData Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.16.2.1 Connection Property

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

Class

TDAMetaData

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object to use to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, set the Connection property to reference an instanciated TCustomDAConnection object.

Request Support   DAC Forum   Provide Feedback

5.11.1.16.2.2 MetaDataKind Property

Used to specify which kind of metainformation to show.

Class

TDAMetaData

Syntax

```
property MetaDataKind: string;
```

Remarks

This string property specifies which kind of metainformation to show. The value of this property should be assigned before activating the component. If MetaDataKind equals to an empty string (the default value), the full value list that this property accepts will be shown.

They are described in the table below:

MetaDataKind	Description
Columns	show metainformation about columns of existing tables
Constraints	show metainformation about the constraints defined in the database
IndexColumns	show metainformation about indexed columns
Indexes	show metainformation about indexes in a database
MetaDataKinds	show the acceptable values of this property. You will get the same result if the MetadataKind property is an empty string
ProcedurePara meters	show metainformation about parameters of existing procedures
Procedures	show metainformation about existing procedures
Restrictions	generates a dataset that describes which restrictions are applicable to each MetaDataKind
Tables	show metainformation about existing tables
Databases	show metainformation about existing databases

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

See Also

- Restrictions

Request Support   DAC Forum   Provide Feedback

5.11.1.16.2.3 Restrictions Property

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

Class

TDAMetaData

Syntax

```
property Restrictions: TStrings;
```

Remarks

Use the Restriction list to provide one or more conditions restricting the list of objects to be described. To see the full list of restrictions and to which metadata kinds they are applicable, you should assign the Restrictions value to the MetaDataKind property and view the result.

See Also

- MetaDataKind

Request Support   DAC Forum   Provide Feedback

5.11.1.16.3 Methods

Methods of the **TDAMetaData** class.

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

Public

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.

DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds	Used to get values acceptable in the MetaDataKind property.
GetRestrictions	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file

	or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TDAMetaData Class

- TDAMetaData Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.16.3.1 GetMetaDataKinds Method

Used to get values acceptable in the MetaDataKind property.

Class

TDAMetaData

Syntax

```
procedure GetMetaDataKinds(List: TStrings);
```

Parameters

List
  Holds the object that will be filled with metadata kinds (restrictions).

Remarks

Call the GetMetaDataKinds method to get values acceptable in the MetaDataKind property.

The List parameter will be cleared and then filled with values.

See Also

* MetaDataKind

Request Support   DAC Forum   Provide Feedback

5.11.1.16.3.2 GetRestrictions Method

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

Class

TDAMetaData

Syntax

```
procedure GetRestrictions(List: TStrings; const MetaDataKind:
string);
```

Parameters

List
  Holds the object that will be filled with metadata kinds (restrictions).
MetaDataKind
  Holds the metadata kind for which restrictions are returned.

Remarks

Call the GetRestrictions method to find out which restrictions are applicable to a certain

MetaDataKind. The List parameter will be cleared and then filled with values.

See Also

- Restrictions

- GetMetaDataKinds

Request Support   DAC Forum   Provide Feedback

5.11.1.17 TDAParam Class

A class that forms objects to represent the values of the parameters set.

For a list of all members of this type, see TDAParam members.

Unit

DBAccess

Syntax

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

Remarks

Use the properties of TDAParam to set the value of a parameter. Objects that use parameters create TDAParam objects to represent these parameters. For example, TDAParam objects are used by TCustomDASQL, TCustomDADataSet.

TDAParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding and the way the field is displayed, edited, or calculated, that are not needed in a TDAParam object. Conversely, TDAParam includes properties that indicate how the field value is passed as a parameter.

See Also

- TCustomDADataSet

- TCustomDASQL

- TDAParams

5.11.1.17.1 Members

TDAParam class overview.

Properties

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

[IsNull](#)	Used to indicate whether the value assigned to a parameter is NULL.
[ParamType](#)	Used to indicate the type of use for a parameter.
[Size](#)	Specifies the size of a string type parameter.
[Value](#)	Used to represent the value of the parameter as Variant.

Methods

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

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.17.2   Properties

Properties of the **TDAParam** class.

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

Public

Name	Description
[AsBlob](#)	Used to set and read the value of the BLOB parameter as string.

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

Published

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

See Also

- TDAParam Class

- TDAParam Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.17.2.1 AsBlob Property

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

Class

TDAParam

Syntax

```
property AsBlob: TBlobData;
```

Remarks

Use the AsBlob property to set and read the value of the BLOB parameter as string. Setting AsBlob will set the DataType property to ftBlob.

Request Support   DAC Forum   Provide Feedback

5.11.1.17.2.2 AsBlobRef Property

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

Class

TDAParam

Syntax

```
property AsBlobRef: TBlob;
```

Remarks

Use the AsBlobRef property to set and read the value of the BLOB parameter as a TBlob

object. Setting AsBlobRef will set the DataType property to ftBlob.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.17.2.3 AsFloat Property

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

Class

[TDAParam](#)

Syntax

```
property AsFloat: double;
```

Remarks

Use the AsFloat property to assign the value for a float field to the parameter. Setting AsFloat will set the DataType property to dtFloat.

Read the AsFloat property to determine the value that was assigned to an output parameter, represented as Double. The value of the parameter will be converted to the Double value if possible.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.17.2.4 AsInteger Property

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

Class

[TDAParam](#)

Syntax

```
property AsInteger: LongInt;
```

Remarks

Use the AsInteger property to assign the value for an integer field to the parameter. Setting

AsInteger will set the DataType property to dtInteger.

Read the AsInteger property to determine the value that was assigned to an output parameter, represented as a 32-bit integer. The value of the parameter will be converted to the Integer value if possible.

Request Support   DAC Forum   Provide Feedback

5.11.1.17.2.5 AsLargeInt Property

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

Class

TDAParam

Syntax

```
property AsLargeInt: Int64;
```

Remarks

Set the AsLargeInt property to assign the value for an Int64 field to the parameter. Setting AsLargeInt will set the DataType property to dtLargeint.

Read the AsLargeInt property to determine the value that was assigned to an output parameter, represented as a 64-bit integer. The value of the parameter will be converted to the Int64 value if possible.

Request Support   DAC Forum   Provide Feedback

5.11.1.17.2.6 AsMemo Property

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

Class

TDAParam

Syntax

```
property AsMemo: string;
```

Remarks

Use the AsMemo property to assign the value for a memo field to the parameter. Setting AsMemo will set the DataType property to ftMemo.

Request Support   DAC Forum   Provide Feedback

5.11.1.17.2.7 AsMemoRef Property

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

Class

TDAParam

Syntax

```
property AsMemoRef: TBlob;
```

Remarks

Use the AsMemoRef property to set and read the value of the memo parameter as a TBlob object. Setting AsMemoRef will set the DataType property to ftMemo.

Request Support   DAC Forum   Provide Feedback

5.11.1.17.2.8 AsSQLTimeStamp Property

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

Class

TDAParam

Syntax

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

5.11.1.17.2.9 AsString Property

Used to assign the string value to the parameter.

Class

TDAParam

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to the parameter. Setting AsString will set the DataType property to ftString.

Read the AsString property to determine the value that was assigned to an output parameter represented as a string. The value of the parameter will be converted to a string.

5.11.1.17.2.10 AsWideString Property

Used to assign the Unicode string value to the parameter.

Class

TDAParam

Syntax

```
property AsWideString: string;
```

Remarks

Set AsWideString to assign the Unicode string value to the parameter. Setting AsWideString will set the DataType property to ftWideString.

Read the AsWideString property to determine the value that was assigned to an output

parameter, represented as a Unicode string. The value of the parameter will be converted to a Unicode string.

　　Request Support　　DAC Forum　　Provide Feedback

5.11.1.17.2.11 DataType Property

Indicates the data type of the parameter.

Class

TDAParam

Syntax

```
property DataType: TFieldType stored IsDataTypeStored;
```

Remarks

DataType is set automatically when a value is assigned to a parameter. Do not set DataType for bound fields, as this may cause the assigned value to be misinterpreted.

Read DataType to learn the type of data that was assigned to the parameter. Every possible value of DataType corresponds to the type of a database field.

　　Request Support　　DAC Forum　　Provide Feedback

5.11.1.17.2.12 IsNull Property

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

Class

TDAParam

Syntax

```
property IsNull: boolean;
```

Remarks

Use the IsNull property to indicate whether the value assigned to a parameter is NULL.

5.11.1.17.2.13 ParamType Property

Used to indicate the type of use for a parameter.

Class

TDAParam

Syntax

```
property ParamType default DB . ptUnknown;
```

Remarks

Objects that use TDAParam objects to represent field parameters set ParamType to indicate the type of use for a parameter.

To learn the description of TParamType refer to Delphi Help.

5.11.1.17.2.14 Size Property

Specifies the size of a string type parameter.

Class

TDAParam

Syntax

```
property Size: integer default 0;
```

Remarks

Use the Size property to indicate the maximum number of characters the parameter may contain. Use the Size property only for Output parameters of the **ftString, ftFixedChar, ftBytes, ftVarBytes,** or **ftWideString** type.

5.11.1.17.2.15 Value Property

Used to represent the value of the parameter as Variant.

Class

[TDAParam](#)

Syntax

```
property Value: variant stored IsValueStored;
```

Remarks

The Value property represents the value of the parameter as Variant.

Use Value in generic code that manipulates the values of parameters without the need to know the field type the parameter represent.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.17.3 Methods

Methods of the **TDAParam** class.

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

Public

Name	Description
[AssignField](#)	Assigns field name and field value to a param.
[AssignFieldValue](#)	Assigns the specified field properties and value to a parameter.
[LoadFromFile](#)	Places the content of a specified file into a TDAParam object.
[LoadFromStream](#)	Places the content from a stream into a TDAParam object.

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

See Also

- TDAParam Class

- TDAParam Class Members

Request Support   DAC Forum   Provide Feedback

5.11.1.17.3.1 AssignField Method

Assigns field name and field value to a param.

Class

TDAParam

Syntax

```
procedure AssignField(Field: TField);
```

Parameters

Field
  Holds the field which name and value should be assigned to the param.

Remarks

Call the AssignField method to assign field name and field value to a param.

Request Support   DAC Forum   Provide Feedback

5.11.1.17.3.2 AssignFieldValue Method

Assigns the specified field properties and value to a parameter.

Class

TDAParam

Syntax

```
procedure AssignFieldValue(Field: TField; const Value: Variant);
virtual;
```

Parameters

Field

Holds the field the properties of which will be assigned to the parameter.

Value

Holds the value for the parameter.

Remarks

Call the AssignFieldValue method to assign the specified field properties and value to a
parameter.

5.11.1.17.3.3 LoadFromFile Method

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

Class

TDAParam

Syntax

```
procedure LoadFromFile(const FileName: string; BlobType:
TBlobType);
```

Parameters

FileName

Holds the name of the file.

BlobType

Holds a value that modifies the DataType property so that this TDAParam object now holds
the BLOB value.

Remarks

Use the LoadFromFile method to place the content of a file specified by FileName into a
TDAParam object. The BlobType value modifies the DataType property so that this
TDAParam object now holds the BLOB value.

See Also

- LoadFromStream

Request Support   DAC Forum   Provide Feedback

5.11.1.17.3.4 LoadFromStream Method

Places the content from a stream into a TDAParam object.

Class

TDAParam

Syntax

```
procedure LoadFromStream(Stream: TStream; BlobType: TBlobType);
virtual;
```

Parameters

Stream

Holds the stream to copy content from.

BlobType

Holds a value that modifies the DataType property so that this TDAParam object now holds the BLOB value.

Remarks

Call the LoadFromStream method to place the content from a stream into a TDAParam object. The BlobType value modifies the DataType property so that this TDAParam object now holds the BLOB value.

See Also

- LoadFromFile

Request Support   DAC Forum   Provide Feedback

5.11.1.17.3.5 SetBlobData Method

Writes the data from a specified buffer to BLOB.

Class

[TDAParam](#)

Overload List

Name	Description
[SetBlobData(Buffer: TValueBuffer)](#)	Writes the data from a specified buffer to BLOB.
[SetBlobData(Buffer: IntPtr; Size: Integer)](#)	Writes the data from a specified buffer to BLOB.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

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.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

Writes the data from a specified buffer to BLOB.

Class

[TDAParam](#)

Syntax

```
procedure SetBlobData(Buffer: IntPtr; Size: Integer); overload;
```

Parameters

Buffer

  Holds the pointer to data.

Size

  Holds the number of bytes to read from the buffer.

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.11.1.18 TDAParams Class

This class is used to manage a list of TDAParam objects for an object that uses field

parameters.

For a list of all members of this type, see TDAParams members.

Unit

DBAccess

Syntax

```
TDAParams = class(TParams);
```

Remarks

Use TDAParams to manage a list of TDAParam objects for an object that uses field

parameters. For example, TCustomDADataSet objects and TCustomDASQL objects use

TDAParams objects to create and access their parameters.

See Also

- TCustomDADataSet.Params

- TCustomDASQL.Params

- TDAParam

Request Support   DAC Forum   Provide Feedback

5.11.1.18.1 Members

TDAParams class overview.

Properties

Name	Description
Items	Used to interate through all parameters.

Methods

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

Request Support   DAC Forum   Provide Feedback

5.11.1.18.2 Properties

Properties of the **TDAParams** class.

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

Public

Name	Description
Items	Used to interate through all parameters.

See Also

- TDAParams Class

- TDAParams Class Members

Request Support   DAC Forum   Provide Feedback

Reserved.

5.11.1.18.2.1 Items Property(Indexer)

Used to interate through all parameters.

Class

TDAParams

Syntax

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

Parameters

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

Remarks

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

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.11.1.18.3 Methods

Methods of the **TDAParams** class.

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

Public

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

See Also

- TDAParams Class

- [TDAParams Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.18.3.1 FindParam Method

Searches for a parameter with the specified name.

Class

[TDAParams](#)

Syntax

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

Parameters

Value
  Holds the parameter name.

Return Value
  a parameter, if a match was found. Nil otherwise.

Remarks

Use the FindParam method to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.18.3.2 ParamByName Method

Searches for a parameter with the specified name.

Class

[TDAParams](#)

Syntax

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

Parameters

Value
  Holds the parameter name.

Return Value
  a parameter, if the match was found. otherwise an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the name passed in Value. If a match was found, ParamByName returns the parameter. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindParam method.

Request Support   DAC Forum   Provide Feedback

5.11.1.19 TDATransaction Class

A base class that implements functionality for controlling transactions.

For a list of all members of this type, see TDATransaction members.

Unit

DBAccess

Syntax

```
TDATransaction = class(TComponent);
```

Remarks

TDATransaction is a base class for components implementing functionality for managing transactions.

Do not create instances of TDATransaction. Use descendants of the TDATransaction class instead.

Request Support   DAC Forum   Provide Feedback

Devart. All Rights
Reserved.

5.11.1.19.1 Members

TDATransaction class overview.

Properties

Name	Description
Active	Used to determine if the transaction is active.
DefaultCloseAction	Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Methods

Name	Description
Commit	Commits the current transaction.
Rollback	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction	Begins a new transaction.

Events

Name	Description
OnCommit	Occurs after the transaction has been successfully committed.
OnCommitRetaining	Occurs after CommitRetaining has been executed.
OnError	Used to process errors that occur during executing a transaction.

[OnRollback](#)	Occurs after the transaction has been successfully rolled back.
[OnRollbackRetaining](#)	Occurs after RollbackRetaining has been executed.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.19.2 Properties

Properties of the **TDATransaction** class.

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

Public

Name	Description
[Active](#)	Used to determine if the transaction is active.
[DefaultCloseAction](#)	Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

See Also

- [TDATransaction Class](#)

- [TDATransaction Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.19.2.1 Active Property

Used to determine if the transaction is active.

Class

TDATransaction

Syntax

```
property Active: boolean;
```

Remarks

Indicates whether the transaction is active. This property is read-only.

Request Support   DAC Forum   Provide Feedback

5.11.1.19.2.2 DefaultCloseAction Property

Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Class

TDATransaction

Syntax

```
property DefaultCloseAction: TCRTransactionAction default
taRollback;
```

Remarks

Use DefaultCloseAction to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Request Support   DAC Forum   Provide Feedback

5.11.1.19.3 Methods

Methods of the **TDATransaction** class.

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

Public

Name	Description
[Commit](#)	Commits the current transaction.
[Rollback](#)	Discards all modifications of data associated with the current transaction and ends the transaction.
[StartTransaction](#)	Begins a new transaction.

See Also

- [TDATransaction Class](#)

- [TDATransaction Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.19.3.1 Commit Method

Commits the current transaction.

Class

`TDATransaction`

Syntax

```
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit the current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database, and then finishes the transaction.

See Also

- [Rollback](#)

- [StartTransaction](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.19.3.2 Rollback Method

Discards all modifications of data associated with the current transaction and ends the
transaction.

Class

TDATransaction

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call Rollback to cancel all data modifications made within the current transaction to the
database server, and finish the transaction.

See Also

- Commit

- StartTransaction

Request Support   DAC Forum   Provide Feedback

5.11.1.19.3.3 StartTransaction Method

Begins a new transaction.

Class

TDATransaction

Syntax

```
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new transaction against the database server.
Before calling StartTransaction, an application should check the Active property. If
TDATransaction.Active is True, indicating that a transaction is already in progress, a

subsequent call to StartTransaction will raise EDatabaseError. An active transaction must be finished by call to [Commit](#) or [Rollback](#) before call to StartTransaction. Call to StartTransaction when connection is closed also will raise EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until the application calls [Commit](#) to save the changes, or [Rollback](#) to cancel them.

See Also

- [Commit](#)

- [Rollback](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.19.4 Events

Events of the **TDATransaction** class.

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

Public

Name	Description
[OnCommit](#)	Occurs after the transaction has been successfully committed.
[OnCommitRetaining](#)	Occurs after CommitRetaining has been executed.
[OnError](#)	Used to process errors that occur during executing a transaction.
[OnRollback](#)	Occurs after the transaction has been successfully rolled back.
[OnRollbackRetaining](#)	Occurs after RollbackRetaining has been executed.

See Also

- [TDATransaction Class](#)

- [TDATransaction Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.19.4.1 OnCommit Event

Occurs after the transaction has been successfully committed.

Class

`TDATransaction`

Syntax

```
property OnCommit: TNotifyEvent;
```

Remarks

The OnCommit event fires when the M:Devart.Dac.TDATransaction.Commit method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.MyDac.TMyTransaction.CommitRetaining() method execution, the [OnCommitRetaining](#) event is used. When an error occurs during commit, the [OnError](#) event fires.

See Also
- [Commit](#)

- [OnError](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.19.4.2 OnCommitRetaining Event

Occurs after CommitRetaining has been executed.

Class

`TDATransaction`

Syntax

```
property OnCommitRetaining: TNotifyEvent;
```

Remarks

The OnCommitRetaining event fires when the CommitRetaining method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.Dac.TDATransaction.Commit method execution, the OnCommit event is used.

When an error occurs during commit, the OnError event fired.

See Also

- Commit

- OnCommit

- OnError

Request Support   DAC Forum   Provide Feedback

5.11.1.19.4.3 OnError Event

Used to process errors that occur during executing a transaction.

Class

TDATransaction

Syntax

```
property OnError: TDATransactionErrorEvent;
```

Remarks

Add a handler to the OnError event to process errors that occur during executing a transaction control statements such as Commit, Rollback. Check the E parameter to get the error code.

See Also

- Commit

- **Rollback**

- **StartTransaction**

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.19.4.4 OnRollback Event

Occurs after the transaction has been successfully rolled back.

Class

`TDATransaction`

Syntax

```
property OnRollback: TNotifyEvent;
```

Remarks

The OnRollback event fires when the M:Devart.Dac.TDATransaction.Rollback method is executed, just after the transaction is successfully rolled back. In order to respond to the M:Devart.MyDac.TMyTransaction.RollbackRetaining() method execution, the **OnRollbackRetaining** event is used.

When an error occurs during rollback, the **OnError** event fired.

See Also
- **Rollback**

- **OnError**

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.19.4.5 OnRollbackRetaining Event

Occurs after RollbackRetaining has been executed.

Class

`TDATransaction`

Syntax

```
property OnRollbackRetaining: TNotifyEvent;
```

Remarks

The OnRollbackRetaining event fires when the RollbackRetaining method is executed, just after the transaction is successfully rolled back. In order to respond to the Rollback method execution, the OnRollback event is used. When an error occurs during rollback, the OnError event fired.

See Also

- Rollback

- OnRollback

- OnError

Request Support   DAC Forum   Provide Feedback

5.11.1.20 TMacro Class

Object that represents the value of a macro.

For a list of all members of this type, see TMacro members.

Unit

DBAccess

Syntax

```
TMacro = class(TCollectionItem);
```

Remarks

TMacro object represents the value of a macro. Macro is a variable that holds string value. You just insert **&** MacroName in a SQL query text and change the value of macro by the Macro property editor at design time or the Value property at run time. At the time of opening query macro is replaced by its value.

If by any reason it is not convenient for you to use the ' **&** ' symbol as a character of macro

replacement, change the value of the MacroChar variable.

See Also

- TMacros

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.20.1 Members

TMacro class overview.

Properties

Name	Description
Active	Used to determine if the macro should be expanded.
AsDateTime	Used to set the TDataTime value to a macro.
AsFloat	Used to set the float value to a macro.
AsInteger	Used to set the integer value to a macro.
AsString	Used to assign the string value to a macro.
Name	Used to identify a particular macro.
Value	Used to set the value to a macro.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.20.2 Properties

Properties of the **TMacro** class.

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

Public

Name	Description

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

Published

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

See Also

- TMacro Class

- TMacro Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.20.2.1 Active Property

Used to determine if the macro should be expanded.

Class

TMacro

Syntax

```
property Active: boolean default True;
```

Remarks

When set to True, the macro will be expanded, otherwise macro definition is replaced by null

string. You can use the Active property to modify the SQL property.

The default value is True.

Example

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

Request Support   DAC Forum   Provide Feedback

5.11.1.20.2.2 AsDateTime Property

Used to set the TDataTime value to a macro.

Class

TMacro

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.11.1.20.2.3 AsFloat Property

Used to set the float value to a macro.

Class

TMacro

Syntax

```
property AsFloat: double;
```

Remarks

Use the AsFloat property to set the float value to a macro.

Request Support   DAC Forum   Provide Feedback

5.11.1.20.2.4 AsInteger Property

Used to set the integer value to a macro.

Class

TMacro

Syntax

```
property AsInteger: integer;
```

Remarks

Use the AsInteger property to set the integer value to a macro.

Request Support   DAC Forum   Provide Feedback

5.11.1.20.2.5 AsString Property

Used to assign the string value to a macro.

Class

TMacro

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to a macro. Read the AsString property to determine the value of macro represented as a string.

Request Support   DAC Forum   Provide Feedback

5.11.1.20.2.6 Name Property

Used to identify a particular macro.

Class

[TMacro](#)

Syntax

```
property Name: string;
```

Remarks

Use the Name property to identify a particular macro.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.20.2.7 Value Property

Used to set the value to a macro.

Class

[TMacro](#)

Syntax

```
property Value: string;
```

Remarks

Use the Value property to set the value to a macro.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.21 TMacros Class

Controls a list of TMacro objects for the [TCustomDASQL.Macros](#) or [TCustomDADataSet](#) components.

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

Unit

[DBAccess](#)

Syntax

```
TMacros = class(TCollection);
```

Remarks

Use TMacros to manage a list of TMacro objects for the [TCustomDASQL](#) or [TCustomDADataSet](#) components.

See Also

- [TMacro](#)

[Request Support](#)　　[DAC Forum](#)　　[Provide Feedback](#)

5.11.1.21.1　Members

[TMacros](#) class overview.

Properties

Name	Description
[Items](#)	Used to iterate through all the macros parameters.

Methods

Name	Description
[AssignValues](#)	Copies the macros values and properties from the specified source.
[Expand](#)	Changes the macros in the passed SQL statement to their values.
[FindMacro](#)	Finds a macro with the specified name.
[IsEqual](#)	Compares itself with another TMacro object.

[MacroByName](#)	Used to search for a macro with the specified name.
[Scan](#)	Creates a macros from the passed SQL statement.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.21.2 Properties

Properties of the **TMacros** class.

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

Public

Name	Description
[Items](#)	Used to iterate through all the macros parameters.

See Also

- [TMacros Class](#)

- [TMacros Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.21.2.1 Items Property(Indexer)

Used to iterate through all the macros parameters.

Class

[TMacros](#)

Syntax

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

Parameters

Index

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

Remarks

Use the Items property to iterate through all macros parameters. Index identifies the index in the range 0..Count - 1.

    Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.21.3 Methods

Methods of the **TMacros** class.

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

Public

Name	Description
AssignValues	Copies the macros values and properties from the specified source.
Expand	Changes the macros in the passed SQL statement to their values.
FindMacro	Finds a macro with the specified name.
IsEqual	Compares itself with another TMacro object.
MacroByName	Used to search for a macro with the specified name.
Scan	Creates a macros from the passed SQL statement.

See Also

- TMacros Class

- TMacros Class Members

    Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.21.3.1 AssignValues Method

Copies the macros values and properties from the specified source.

Class

TMacros

Syntax

```
procedure AssignValues(Value: TMacros);
```

Parameters

Value
  Holds the source to copy the macros values and properties from.

Remarks

The Assign method copies the macros values and properties from the specified source.

Macros are not recreated. Only the values of macros with matching names are assigned.

Request Support   DAC Forum   Provide Feedback

5.11.1.21.3.2 Expand Method

Changes the macros in the passed SQL statement to their values.

Class

TMacros

Syntax

```
procedure Expand(var SQL: string);
```

Parameters

SQL
  Holds the passed SQL statement.

Remarks

Call the Expand method to change the macros in the passed SQL statement to their values.

Request Support   DAC Forum   Provide Feedback

Reserved.

5.11.1.21.3.3 FindMacro Method

Finds a macro with the specified name.

Class

TMacros

Syntax

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

Parameters

Value
   Holds the value of a macro to search for.

Return Value
   TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the Items property to avoid depending on the order of the items.

Request Support   DAC Forum   Provide Feedback

5.11.1.21.3.4 IsEqual Method

Compares itself with another TMacro object.

Class

TMacros

Syntax

```
function IsEqual(Value: TMacros): boolean;
```

Parameters

Value
   Holds the values of TMacro objects.

Return Value

True, if the number of TMacro objects and the values of all TMacro objects are equal.

Remarks

Call the IsEqual method to compare itself with another TMacro object. Returns True if the number of TMacro objects and the values of all TMacro objects are equal.

Request Support   DAC Forum   Provide Feedback

5.11.1.21.3.5 MacroByName Method

Used to search for a macro with the specified name.

Class

TMacros

Syntax

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

Parameters

Value

Holds a name of the macro to search for.

Return Value

TMacro object, if a macro with specified name was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Value. If a match is found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the Items property to avoid depending on the order of the items.

To locate a macro by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

Request Support   DAC Forum   Provide Feedback

5.11.1.21.3.6 Scan Method

Creates a macros from the passed SQL statement.

Class

TMacros

Syntax

```
procedure Scan(const SQL: string);
```

Parameters

SQL
   Holds the passed SQL statement.

Remarks

Call the Scan method to create a macros from the passed SQL statement. On that all existing TMacro objects are cleared.

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.11.1.22 TPoolingOptions Class

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

For a list of all members of this type, see TPoolingOptions members.

Unit

DBAccess

Syntax

```
TPoolingOptions = class(TPersistent);
```

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.11.1.22.1 Members

TPoolingOptions class overview.

Properties

Name	Description
[ConnectionLifetime](#)	Used to specify the maximum time during which an open connection can be used by connection pool.
[MaxPoolSize](#)	Used to specify the maximum number of connections that can be opened in connection pool.
[MinPoolSize](#)	Used to specify the minimum number of connections that can be opened in the connection pool.
[PoolId](#)	Used to specify an ID for a connection pool.
[Validate](#)	Used for a connection to be validated when it is returned from the pool.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.22.2 Properties

Properties of the **TPoolingOptions** class.

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

Published

Name	Description
[ConnectionLifetime](#)	Used to specify the maximum time during which an open connection can be used by connection pool.
[MaxPoolSize](#)	Used to specify the maximum number of connections that can be opened in connection pool.

[MinPoolSize](#)	Used to specify the minimum number of connections that can be opened in the connection pool.
[PoolId](#)	Used to specify an ID for a connection pool.
[Validate](#)	Used for a connection to be validated when it is returned from the pool.

See Also

- [TPoolingOptions Class](#)

- [TPoolingOptions Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.22.2.1 ConnectionLifetime Property

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

Class

`TPoolingOptions`

Syntax

```
property ConnectionLifetime: integer default
DefValConnectionLifetime;
```

Remarks

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Reserved.

5.11.1.22.2.2 MaxPoolSize Property

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

Class

`TPoolingOptions`

Syntax

```
property MaxPoolSize: integer default DefValMaxPoolSize;
```

Remarks

Specifies the maximum number of connections that can be opened in connection pool. Once this value is reached, no more connections are opened. The valid values are 1 and higher.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.22.2.3 MinPoolSize Property

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

Class

`TPoolingOptions`

Syntax

```
property MinPoolSize: integer default DefValMinPoolSize;
```

Remarks

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.11.1.22.2.4 PoolId Property

Used to specify an ID for a connection pool.

Class

TPoolingOptions

Syntax

```
property PoolId: Integer default DefValPoolId;
```

Remarks

Use the PoolId property to make a group of connections use a specific connection pool.

Request Support   DAC Forum   Provide Feedback

5.11.1.22.2.5 Validate Property

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

Class

TPoolingOptions

Syntax

```
property Validate: boolean default DefValValidate;
```

Remarks

If the Validate property is set to True, connection will be validated when it is returned from the pool. By default this option is set to False and pool does not validate connection when it is returned to be used by a TCustomDAConnection component.

Request Support   DAC Forum   Provide Feedback

5.11.1.23 TSmartFetchOptions Class

Smart fetch options are used to set up the behavior of the SmartFetch mode.

For a list of all members of this type, see TSmartFetchOptions members.

Unit

[DBAccess](#)

Syntax

```
TSmartFetchOptions = class(TPersistent);
```

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.23.1 Members

TSmartFetchOptions class overview.

Properties

Name	Description
[Enabled](#)	Sets SmartFetch mode enabled or not.
[LiveBlock](#)	Used to minimize memory consumption.
[PrefetchedFields](#)	List of fields additional to key fields that will be read from the database on dataset open.
[SQLGetKeyValues](#)	SQL query for the read key and prefetched fields from the database.

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.23.2 Properties

Properties of the **TSmartFetchOptions** class.

For a complete list of the **TSmartFetchOptions** class members, see the

[TSmartFetchOptions Members](#) topic.

Published

Name	Description

Enabled	Sets SmartFetch mode enabled or not.
LiveBlock	Used to minimize memory consumption.
PrefetchedFields	List of fields additional to key fields that will be read from the database on dataset open.
SQLGetKeyValues	SQL query for the read key and prefetched fields from the database.

See Also

- TSmartFetchOptions Class

- TSmartFetchOptions Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.23.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

Class

`TSmartFetchOptions`

Syntax

```
property Enabled: Boolean default False;
```

Request Support   **DAC Forum**   **Provide Feedback**

5.11.1.23.2.2 LiveBlock Property

Used to minimize memory consumption.

Class

`TSmartFetchOptions`

Syntax

```
property LiveBlock: Boolean default True;
```

Remarks

If LiveBlock is True, then on navigating through a dataset forward or backward, memory will be allocated for records count defined in the the FetchRows propety, and no additional memory will be allocated. But if you return records that were read from the database before, they will be read from the database again, because when you left block with these records, memory was free. So the LiveBlock mode minimizes memory consumption, but can decrease performance, because it can lead to repeated data reading from the database.

The default value of LiveBlock is False.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.11.1.23.2.3 PrefetchedFields Property

List of fields additional to key fields that will be read from the database on dataset open.

Class

TSmartFetchOptions

Syntax

```
property PrefetchedFields: string;
```

Remarks

If you are going to use locate, filter or sort by some fields, then these fields should be added to the prefetched fields list to avoid excessive reading from the database.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.11.1.23.2.4 SQLGetKeyValues Property

SQL query for the read key and prefetched fields from the database.

Class

TSmartFetchOptions

Syntax

```
property SQLGetKeyValues: TStrings;
```

Remarks

SQLGetKeyValues is used when the basic SQL query is complex and the query for reading the key and prefetched fields can't be generated automatically.

| **Request Support** | **DAC Forum** | **Provide Feedback**

5.11.2 Types

Types in the **DBAccess** unit.

Types

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecute events.
TAfterFetchEvent	This type is used for the TCustomDADataSet.AfterFetch event.
TBeforeFetchEvent	This type is used for the TCustomDADataSet.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDAConnection.OnConnectionLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDAConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption.

	This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.
[TUpdateExecuteEvent](#)	

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.11.2.1 TAfterExecuteEvent Procedure Reference

This type is used for the [TCustomDADataSet.AfterExecute](#) and [TCustomDASQL.AfterExecute](#) events.

Unit

[DBAccess](#)

Syntax

```
TAfterExecuteEvent = procedure (Sender: TObject; Result: boolean)
of object;
```

Parameters

Sender

An object that raised the event.

Result

The result is True if SQL statement is executed successfully. False otherwise.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.11.2.2 TAfterFetchEvent Procedure Reference

This type is used for the [TCustomDADataSet.AfterFetch](#) event.

Unit

[DBAccess](#)

Syntax

```
TAfterFetchEvent = procedure (DataSet: TCustomDADataSet) of
```

```
object;
```

Parameters

DataSet

  Holds the TCustomDADataSet descendant to synchronize the record position with.

  Request Support   DAC Forum   Provide Feedback

5.11.2.3 TBeforeFetchEvent Procedure Reference

This type is used for the TCustomDADataSet.BeforeFetch event.

Unit

DBAccess

Syntax

```
TBeforeFetchEvent = procedure (DataSet: TCustomDADataSet; var
Cancel: boolean) of object;
```

Parameters

DataSet

  Holds the TCustomDADataSet descendant to synchronize the record position with.

Cancel

  True, if the current fetch operation should be aborted.

  Request Support   DAC Forum   Provide Feedback

5.11.2.4 TConnectionLostEvent Procedure Reference

This type is used for the TCustomDAConnection.OnConnectionLost event.

Unit

DBAccess

Syntax

```
TConnectionLostEvent = procedure (Sender: TObject; Component:
TComponent; ConnLostCause: TConnLostCause; var RetryMode:
TRetryMode) of object;
```

Parameters

Sender

   An object that raised the event.

Component

ConnLostCause

   The reason of the connection loss.

RetryMode

   The application behavior when connection is lost.

5.11.2.5 TDAConnectionErrorEvent Procedure Reference

This type is used for the TCustomDAConnection.OnError event.

Unit

DBAccess

Syntax

```
TDAConnectionErrorEvent = procedure (Sender: TObject; E: EDAError;
var Fail: boolean) of object;
```

Parameters

Sender

   An object that raised the event.

E

   The error information.

Fail

   False, if an error dialog should be prevented from being displayed and EAbort exception
   should be raised to cancel current operation .

5.11.2.6 TDATransactionErrorEvent Procedure Reference

This type is used for the TDATransaction.OnError event.

Unit

DBAccess

Syntax

```
TDATransactionErrorEvent = procedure (Sender: TObject; E:
EDAError; var Fail: boolean) of object;
```

Parameters

Sender

An object that raised the event.

E

The error code.

Fail

False, if an error dialog should be prevented from being displayed and EAbort exception to cancel the current operation should be raised.

Request Support   DAC Forum   Provide Feedback

5.11.2.7 TRefreshOptions Set

Represents the set of TRefreshOption.

Unit

DBAccess

Syntax

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

Request Support   DAC Forum   Provide Feedback

5.11.2.8 TUpdateExecuteEvent Procedure Reference

This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.

Unit

DBAccess

Syntax

```
TUpdateExecuteEvent = procedure (Sender: TDataSet; StatementTypes:
TStatementTypes; Params: TDAParams) of object;
```

Parameters

Sender

  An object that raised the event.

StatementTypes

  Holds the type of the SQL statement being executed.

Params

  Holds the parameters with which the SQL statement will be executed.

Request Support   DAC Forum   Provide Feedback

5.11.3 Enumerations

Enumerations in the **DBAccess** unit.

Enumerations

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

Request Support   DAC Forum   Provide Feedback

5.11.3.1 TLabelSet Enumeration

Sets the languauge of labels in the connect dialog.

Unit

DBAccess

Syntax

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

Values

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

Request Support   DAC Forum   Provide Feedback

5.11.3.2 TLockMode Enumeration

Specifies the lock mode.

Unit

DBAccess

Syntax

```
TLockMode = (lmNone, lmPessimistic, lmOptimistic);
```

Values

Value	Meaning
lmNone	No locking occurs. This mode should only be used in single user applications. The default value.
lmOptimistic	Locking occurs when the user posts an edited record, then the lock is released. Locking is done by the RefreshRecord method.
lmPessimistic	Locking occurs when the user starts editing a record. The lock is released after the user has posted or canceled the changes.

Request Support   DAC Forum   Provide Feedback

5.11.3.3 TRefreshOption Enumeration

Indicates when the editing record will be refreshed.

Unit

DBAccess

Syntax

```
TRefreshOption = (roAfterInsert, roAfterUpdate, roBeforeEdit);
```

Values

Value	Meaning
roAfterInsert	Refresh is performed after inserting.
roAfterUpdate	Refresh is performed after updating.
roBeforeEdit	Refresh is performed by Edit method.

Request Support   DAC Forum   Provide Feedback

5.11.3.4 TRetryMode Enumeration

Specifies the application behavior when connection is lost.

Unit

DBAccess

Syntax

```
TRetryMode = (rmRaise, rmReconnect, rmReconnectExecute);
```

Values

Value	Meaning
rmRaise	An exception is raised.
rmReconnect	Reconnect is performed and then exception is raised.
rmReconnectExecute	Reconnect is performed and abortive operation is reexecuted. Exception is not raised.

5.11.4 Variables

Variables in the **DBAccess** unit.

Variables

Name	Description
BaseSQLOldBehavior	After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.
ChangeCursor	When set to True allows data access components to change screen cursor for the execution time.
SQLGeneratorCompatibility	The value of the TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.

5.11.4.1 BaseSQLOldBehavior Variable

After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.

Unit

`DBAccess`

Syntax

```
BaseSQLOldBehavior: boolean = False;
```

Remarks

The [BaseSQL](#) property is similar to the SQL property, but it does not store changes made by the [AddWhere](#), [DeleteWhere](#), and [SetOrderBy](#) methods. After assigning SQL text and modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the BaseSQL property. This behavior was changed in MyDAC 4.00.2.8. To restore old behavior, set the BaseSQLOldBehavior variable to True.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.11.4.2 ChangeCursor Variable

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

Unit

[DBAccess](#)

Syntax

```
ChangeCursor: boolean = True;
```

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.11.4.3 SQLGeneratorCompatibility Variable

The value of the [TCustomDADataSet.BaseSQL](#) property is used to complete the refresh SQL statement, if the manually assigned [TCustomDAUpdateSQL.RefreshSQL](#) property contains only WHERE clause.

Unit

[DBAccess](#)

Syntax

```
SQLGeneratorCompatibility: boolean = False;
```

Remarks

If the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause, MyDAC 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 MyDAC 5.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

Request Support   DAC Forum   Provide Feedback

5.12 MemData

This unit contains classes for storing data in memory.

Classes

Name	Description
TAttribute	TAttribute is not used in MyDAC.
TBlob	Holds large object value for field and parameter dtBlob, dtMemo data types.
TCompressedBlob	Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.
TDBObject	A base class for classes that work with user-defined data types that have attributes.
TMemData	Implements in-memory database.
TObjectType	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
TCompressBlobMode	Specifies when the values should be compressed and the way they should be stored.
TConnLostCause	Specifies the cause of the connection loss.
TDANumericType	Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.
TLocateExOption	Allows to set additional search parameters which will be used by the LocateEx method.
TSortType	Specifies a sort type for string fields.
TUpdateRecKind	Indicates records for which the ApplyUpdates method will be performed.

Request Support   DAC Forum   Provide Feedback

5.12.1 Classes

Classes in the **MemData** unit.

Classes

Name	Description

TAttribute	TAttribute is not used in MyDAC.
TBlob	Holds large object value for field and parameter dtBlob, dtMemo data types.
TCompressedBlob	Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.
TDBObject	A base class for classes that work with user-defined data types that have attributes.
TMemData	Implements in-memory database.
TObjectType	This class is not used.
TSharedObject	A base class that allows to simplify memory management for object referenced by several other objects.

Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.1 TAttribute Class

TAttribute is not used in MyDAC.

For a list of all members of this type, see TAttribute members.

Unit

MemData

Syntax

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

Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.1.1 Members

TAttribute class overview.

Properties

Name	Description
[AttributeNo](#)	Returns an attribute's ordinal position in object.
[DataSize](#)	Returns the size of an attribute value in internal representation.
[DataType](#)	Returns the type of data that was assigned to the Attribute.
[Length](#)	Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.
[ObjectType](#)	Returns a TObjectType object for an object attribute.
[Offset](#)	Returns an offset of the attribute value in internal representation.
[Owner](#)	Indicates TObjectType that uses the attribute to represent one of its attributes.
[Scale](#)	Returns the scale of dtFloat and dtInteger attributes.
[Size](#)	Returns the size of an attribute value in external representation.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.12.1.1.2 Properties

Properties of the **TAttribute** class.

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

Public

Name	Description

[AttributeNo](#)	Returns an attribute's ordinal position in object.
[DataSize](#)	Returns the size of an attribute value in internal representation.
[DataType](#)	Returns the type of data that was assigned to the Attribute.
[Length](#)	Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.
[ObjectType](#)	Returns a TObjectType object for an object attribute.
[Offset](#)	Returns an offset of the attribute value in internal representation.
[Owner](#)	Indicates TObjectType that uses the attribute to represent one of its attributes.
[Scale](#)	Returns the scale of dtFloat and dtInteger attributes.
[Size](#)	Returns the size of an attribute value in external representation.

See Also

- [TAttribute Class](#)

- [TAttribute Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.12.1.1.2.1 AttributeNo Property

Returns an attribute's ordinal position in object.

Class

[TAttribute](#)

Syntax

```
property AttributeNo: Word;
```

Remarks

Use the AttributeNo property to learn an attribute's ordinal position in object, where 1 is the first field.

See Also

* TObjectType.Attributes

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.2 DataSize Property

Returns the size of an attribute value in internal representation.

Class

TAttribute

Syntax

```
property DataSize: Integer;
```

Remarks

Use the DataSize property to learn the size of an attribute value in internal representation.

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.3 DataType Property

Returns the type of data that was assigned to the Attribute.

Class

TAttribute

Syntax

```
property DataType: Word;
```

Remarks

Use the DataType property to discover the type of data that was assigned to the Attribute.

Possible values: dtDate, dtFloat, dtInteger, dtString, dtObject.

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.4 Length Property

Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.

Class

TAttribute

Syntax

```
property Length: Word;
```

Remarks

Use the Length property to learn the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.

See Also

- Scale

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.5 ObjectType Property

Returns a TObjectType object for an object attribute.

Class

TAttribute

Syntax

```
property ObjectType: TObjectType;
```

Remarks

Use the ObjectType property to return a TObjectType object for an object attribute.

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.6 Offset Property

Returns an offset of the attribute value in internal representation.

Class

TAttribute

Syntax

```
property Offset: Integer;
```

Remarks

Use the DataSize property to learn an offset of the attribute value in internal representation.

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.7 Owner Property

Indicates TObjectType that uses the attribute to represent one of its attributes.

Class

TAttribute

Syntax

```
property Owner: TObjectType;
```

Remarks

Check the value of the Owner property to determine TObjectType that uses the attribute to

represent one of its attributes. Applications should not assign the Owner property directly.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.12.1.1.2.8 Scale Property

Returns the scale of dtFloat and dtInteger attributes.

Class

TAttribute

Syntax

```
property Scale: Word;
```

Remarks

Use the Scale property to learn the scale of dtFloat and dtInteger attributes.

See Also

- [Length](#)

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.12.1.1.2.9 Size Property

Returns the size of an attribute value in external representation.

Class

TAttribute

Syntax

```
property Size: Integer;
```

Remarks

Read Size to learn the size of an attribute value in external representation.

For example:

dtDate	8 (sizeof(TDateTime)
dtFloat	8 (sizeof(Double))
dtInteger	4 (sizeof(Integer))

See Also

- DataSize

Request Support   DAC Forum   Provide Feedback

5.12.1.2 TBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types.

For a list of all members of this type, see TBlob members.

Unit

MemData

Syntax

```
TBlob = class(TSharedObject);
```

Remarks

Object TBlob holds large object value for the field and parameter dtBlob, dtMemo, dtWideMemo data types.

Inheritance Hierarchy

TSharedObject

  TBlob

See Also

- TMemDataSet.GetBlob

Request Support   DAC Forum   Provide Feedback

Reserved.

5.12.1.2.1 Members

TBlob class overview.

Properties

Name	Description
AsString	Used to manipulate BLOB value as string.
AsWideString	Used to manipulate BLOB value as Unicode string.
IsUnicode	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Sets BLOB value from another TBlob object.
Clear	Deletes the current value in TBlob object.
LoadFromFile	Loads the contents of a file into a TBlob object.
LoadFromStream	Copies the contents of a stream into the TBlob object.
Read	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from **TSharedObject**)	Decrements the reference count.

SaveToFile	Saves the contents of the TBlob object to a file.
SaveToStream	Copies the contents of a TBlob object to a stream.
Truncate	Sets new TBlob size and discards all data over it.
Write	Stores a raw sequence of bytes into a TBlob object.

Request Support   DAC Forum   Provide Feedback

5.12.1.2.2 Properties

Properties of the **TBlob** class.

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

Public

Name	Description
AsString	Used to manipulate BLOB value as string.
AsWideString	Used to manipulate BLOB value as Unicode string.
IsUnicode	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of the TBlob value in bytes.

See Also

- TBlob Class

- TBlob Class Members

Request Support   DAC Forum   Provide Feedback

5.12.1.2.2.1 AsString Property

Used to manipulate BLOB value as string.

Class

[TBlob](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to manipulate BLOB value as string.

See Also

- [Assign](#)

- [AsWideString](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.12.1.2.2.2 AsWideString Property

Used to manipulate BLOB value as Unicode string.

Class

[TBlob](#)

Syntax

```
property AsWideString: string;
```

Remarks

Use the AsWideString property to manipulate BLOB value as Unicode string.

See Also

- [Assign](#)

- [AsString](#)

Request Support   DAC Forum   Provide Feedback

5.12.1.2.2.3 IsUnicode Property

Gives choice of making TBlob store and process data in Unicode format or not.

Class

TBlob

Syntax

```
property IsUnicode: boolean;
```

Remarks

Set IsUnicode to True if you want TBlob to store and process data in Unicode format.

Note: changing this property raises an exception if TBlob is not empty.

Request Support   DAC Forum   Provide Feedback

5.12.1.2.2.4 Size Property

Used to learn the size of the TBlob value in bytes.

Class

TBlob

Syntax

```
property Size: Cardinal;
```

Remarks

Use the Size property to find out the size of the TBlob value in bytes.

Request Support   DAC Forum   Provide Feedback

5.12.1.2.3 Methods

Methods of the **TBlob** class.

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

Public

Name	Description
[AddRef](#) (inherited from **[TSharedObject](#)**)	Increments the reference count for the number of references dependent on the TSharedObject object.
[Assign](#)	Sets BLOB value from another TBlob object.
[Clear](#)	Deletes the current value in TBlob object.
[LoadFromFile](#)	Loads the contents of a file into a TBlob object.
[LoadFromStream](#)	Copies the contents of a stream into the TBlob object.
[Read](#)	Acquires a raw sequence of bytes from the data stored in TBlob.
[Release](#) (inherited from **[TSharedObject](#)**)	Decrements the reference count.
[SaveToFile](#)	Saves the contents of the TBlob object to a file.
[SaveToStream](#)	Copies the contents of a TBlob object to a stream.
[Truncate](#)	Sets new TBlob size and discards all data over it.
[Write](#)	Stores a raw sequence of bytes into a TBlob object.

See Also

- [TBlob Class](#)

- [TBlob Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.2.3.1 Assign Method

Sets BLOB value from another TBlob object.

Class

TBlob

Syntax

```
procedure Assign(Source: TBlob);
```

Parameters

Source
  Holds the BLOB from which the value to the current object will be assigned.

Remarks

Call the Assign method to set BLOB value from another TBlob object.

See Also

- LoadFromStream

- AsString

- AsWideString

Request Support   DAC Forum   Provide Feedback

5.12.1.2.3.2 Clear Method

Deletes the current value in TBlob object.

Class

TBlob

Syntax

```
procedure Clear; virtual;
```

Remarks

Call the Clear method to delete the current value in TBlob object.

5.12.1.2.3.3 LoadFromFile Method

Loads the contents of a file into a TBlob object.

Class

[TBlob]

Syntax

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

Parameters

FileName
   Holds the name of the file from which the TBlob value is loaded.

Remarks

Call the LoadFromFile method to load the contents of a file into a TBlob object. Specify the
name of the file to load into the field as the value of the FileName parameter.

See Also

- SaveToFile

5.12.1.2.3.4 LoadFromStream Method

Copies the contents of a stream into the TBlob object.

Class

[TBlob]

Syntax

```
procedure LoadFromStream(Stream: TStream); virtual;
```

Parameters

Stream

Holds the specified stream from which the field's value is copied.

Remarks

Call the LoadFromStream method to copy the contents of a stream into the TBlob object. Specify the stream from which the field's value is copied as the value of the Stream parameter.

See Also

- SaveToStream

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.12.1.2.3.5 Read Method

Acquires a raw sequence of bytes from the data stored in TBlob.

Class

TBlob

Syntax

```
function Read(Position: Cardinal; Count: Cardinal; Dest: IntPtr):
Cardinal; virtual;
```

Parameters

Position
  Holds the starting point of the byte sequence.

Count
  Holds the size of the sequence in bytes.

Dest
  Holds a pointer to the memory area where to store the sequence.

Return Value
  Actually read byte count if the sequence crosses object size limit.

Remarks

Call the Read method to acquire a raw sequence of bytes from the data stored in TBlob.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Dest parameter is a pointer to the memory area where to store the sequence.

If the sequence crosses object size limit, function will return actually read byte count.

See Also

- Write

Request Support   DAC Forum   Provide Feedback

5.12.1.2.3.6 SaveToFile Method

Saves the contents of the TBlob object to a file.

Class

TBlob

Syntax

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

Parameters

FileName
  Holds a string that contains the name of the file.

Remarks

Call the SaveToFile method to save the contents of the TBlob object to a file. Specify the name of the file as the value of the FileName parameter.

See Also

- LoadFromFile

Request Support   DAC Forum   Provide Feedback

5.12.1.2.3.7 SaveToStream Method

Copies the contents of a TBlob object to a stream.

Class

TBlob

Syntax

```
procedure SaveToStream(Stream: TStream); virtual;
```

Parameters

Stream
  Holds the name of the stream.

Remarks

Call the SaveToStream method to copy the contents of a TBlob object to a stream. Specify the name of the stream to which the field's value is saved as the value of the Stream parameter.

See Also

• LoadFromStream

Request Support   DAC Forum   Provide Feedback

5.12.1.2.3.8 Truncate Method

Sets new TBlob size and discards all data over it.

Class

TBlob

Syntax

```
procedure Truncate(NewSize: Cardinal); virtual;
```

Parameters

NewSize
  Holds the new size of TBlob.

Remarks

Call the Truncate method to set new TBlob size and discard all data over it. If NewSize is greater or equal TBlob.Size, it does nothing.

Request Support   DAC Forum   Provide Feedback

5.12.1.2.3.9 Write Method

Stores a raw sequence of bytes into a TBlob object.

Class

[TBlob](#)

Syntax

```
procedure Write(Position: Cardinal; Count: Cardinal; Source:
IntPtr); virtual;
```

Parameters

Position
  Holds the starting point of the byte sequence.

Count
  Holds the size of the sequence in bytes.

Source
  Holds a pointer to a source memory area.

Remarks

Call the Write method to store a raw sequence of bytes into a TBlob object.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Source parameter is a pointer to a source memory area.

If the value of the Position parameter crosses current size limit of TBlob object, source data will be appended to the object data.

See Also

- [Read](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.12.1.3 TCompressedBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.

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

Unit

MemData

Syntax

```
TCompressedBlob = class(TBlob);
```

Remarks

TCompressedBlob is a descendant of the TBlob class. It holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data. For more information about using BLOB compression see TCustomDADataSet.Options.

Note: Internal compression functions are available in CodeGear Delphi 2007 for Win32, Borland Developer Studio 2006, Borland Delphi 2005, and Borland Delphi 7. To use BLOB compression under Borland Delphi 6 and Borland C++ Builder you should use your own compression functions. To use them set the CompressProc and UncompressProc variables declared in the MemUtils unit.

Example

```
type
    TCompressProc = function(dest: IntPtr; destLen: IntPtr; const source: I
    TUncompressProc = function(dest: IntPtr; destlen: IntPtr; source: IntPtr
var
    CompressProc: TCompressProc;
    UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

TSharedObject

  TBlob

    TCompressedBlob

See Also

- TBlob

- TMemDataSet.GetBlob

- TCustomDADataSet.Options

Request Support   DAC Forum   Provide Feedback

Reserved.

5.12.1.3.1 Members

TCompressedBlob class overview.

Properties

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Compressed	Used to indicate if the Blob is compressed.
CompressedSize	Used to indicate compressed size of the Blob data.
IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from **TBlob**)	Sets BLOB value from another TBlob object.
Clear (inherited from **TBlob**)	Deletes the current value in TBlob object.
LoadFromFile (inherited from **TBlob**)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from **TBlob**)	Copies the contents of a stream into the TBlob object.

Read (inherited from **TBlob**)	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile (inherited from **TBlob**)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from **TBlob**)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from **TBlob**)	Sets new TBlob size and discards all data over it.
Write (inherited from **TBlob**)	Stores a raw sequence of bytes into a TBlob object.

Request Support   DAC Forum   Provide Feedback

5.12.1.3.2 Properties

Properties of the **TCompressedBlob** class.

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

Public

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Compressed	Used to indicate if the Blob is compressed.
CompressedSize	Used to indicate compressed size of the Blob data.
IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

See Also

- TCompressedBlob Class

- TCompressedBlob Class Members

    Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.3.2.1 Compressed Property

Used to indicate if the Blob is compressed.

Class

TCompressedBlob

Syntax

```
property Compressed: boolean;
```

Remarks

Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

    Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.3.2.2 CompressedSize Property

Used to indicate compressed size of the Blob data.

Class

TCompressedBlob

Syntax

```
property CompressedSize: Cardinal;
```

Remarks

Indicates compressed size of the Blob data.

Request Support   DAC Forum   Provide Feedback

5.12.1.4 TDBObject Class

A base class for classes that work with user-defined data types that have attributes.

For a list of all members of this type, see TDBObject members.

Unit

MemData

Syntax

```
TDBObject = class(TSharedObject);
```

Remarks

TDBObject is a base class for classes that work with user-defined data types that have attributes.

Inheritance Hierarchy

TSharedObject

  TDBObject

Request Support   DAC Forum   Provide Feedback

5.12.1.4.1 Members

TDBObject class overview.

Properties

Name	Description
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a

	TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

Request Support   DAC Forum   Provide Feedback

5.12.1.5 TMemData Class

Implements in-memory database.

For a list of all members of this type, see TMemData members.

Unit

`MemData`

Syntax

```
TMemData = class(TData);
```

Inheritance Hierarchy

TData

   TMemData

Request Support   DAC Forum   Provide Feedback

5.12.1.5.1 Members

TMemData class overview.

Request Support   DAC Forum   Provide Feedback

5.12.1.6 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

TSharedObject

TObjectType

Request Support   DAC Forum   Provide Feedback

5.12.1.6.1 Members

TObjectType class overview.

Properties

Name	Description
AttributeCount	Used to indicate the number of attributes of type.
Attributes	Used to access separate attributes.
DataType	Used to indicate the type of object dtObject, dtArray or dtTable.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of an object instance.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

Request Support   DAC Forum   Provide Feedback

5.12.1.6.2 Properties

Properties of the **TObjectType** class.

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

Public

Name	Description
AttributeCount	Used to indicate the number of attributes of type.
Attributes	Used to access separate attributes.
DataType	Used to indicate the type of object dtObject, dtArray or dtTable.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of an object instance.

See Also

- TObjectType Class

- TObjectType Class Members

5.12.1.6.2.1 AttributeCount Property

Used to indicate the number of attributes of type.

Class

[TObjectType](#)

Syntax

```
property AttributeCount: Integer;
```

Remarks

Use the AttributeCount property to determine the number of attributes of type.

5.12.1.6.2.2 Attributes Property(Indexer)

Used to access separate attributes.

Class

[TObjectType](#)

Syntax

```
property Attributes[Index: integer]: TAttribute;
```

Parameters

Index
  Holds the attribute's ordinal position.

Remarks

Use the Attributes property to access individual attributes. The value of the Index parameter corresponds to the AttributeNo property of TAttribute.

See Also

- [TAttribute](#)

- [FindAttribute](#)

    Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.6.2.3 DataType Property

Used to indicate the type of object dtObject, dtArray or dtTable.

Class

`TObjectType`

Syntax

```
property DataType: Word;
```

Remarks

Use the DataType property to determine the type of object dtObject, dtArray or dtTable.

    Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.6.2.4 Size Property

Used to learn the size of an object instance.

Class

`TObjectType`

Syntax

```
property Size: Integer;
```

Remarks

Use the Size property to find out the size of an object instance. Size is a sum of all attribute sizes.

See Also

- TAttribute.Size

Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.6.3 Methods

Methods of the **TObjectType** class.

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

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

See Also

- TObjectType Class

- TObjectType Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.6.3.1 FindAttribute Method

Indicates whether a specified Attribute component is referenced in the TAttributes object.

Class

`TObjectType`

Syntax

```
function FindAttribute(const Name: string): TAttribute; virtual;
```

Parameters

Name

   Holds the name of the attribute to search for.

Return Value

   TAttribute, if an attribute with a matching name was found. Nil Otherwise.

Remarks

Call FindAttribute to determine if a specified Attribute component is referenced in the TAttributes object. Name is the name of the Attribute for which to search. If FindAttribute finds an Attribute with a matching name, it returns the TAttribute. Otherwise it returns nil.

See Also

- TAttribute

- Attributes

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.12.1.7 TSharedObject Class

A base class that allows to simplify memory management for object referenced by several other objects.

For a list of all members of this type, see TSharedObject members.

Unit

MemData

Syntax

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

Remarks

TSharedObject allows to simplify memory management for object referenced by several other objects. TSharedObject holds a count of references to itself. When any object (referer object) is going to use TSharedObject, it calls the TSharedObject.AddRef method. Referer

object has to call the TSharedObject.Release method after using TSharedObject.

See Also

- TBlob

- TObjectType

Request Support   DAC Forum   Provide Feedback

5.12.1.7.1 Members

TSharedObject class overview.

Properties

Name	Description
RefCount	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef	Increments the reference count for the number of references dependent on the TSharedObject object.
Release	Decrements the reference count.

Request Support   DAC Forum   Provide Feedback

5.12.1.7.2 Properties

Properties of the **TSharedObject** class.

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

Public

Name	Description
[RefCount](#)	Used to return the count of reference to a TSharedObject object.

See Also

- [TSharedObject Class](#)

- [TSharedObject Class Members](#)

Request Support   DAC Forum   Provide Feedback

5.12.1.7.2.1 RefCount Property

Used to return the count of reference to a TSharedObject object.

Class

TSharedObject

Syntax

```
property RefCount: Integer;
```

Remarks

Returns the count of reference to a TSharedObject object.

Request Support   DAC Forum   Provide Feedback

5.12.1.7.3 Methods

Methods of the **TSharedObject** class.

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

Public

Name	Description

	Increments the reference count for the number of references dependent on the TSharedObject object.
[AddRef](#)	
[Release](#)	Decrements the reference count.

See Also

- [TSharedObject Class](#)

- [TSharedObject Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.7.3.1 AddRef Method

Increments the reference count for the number of references dependent on the TSharedObject object.

Class

[TSharedObject](#)

Syntax

```
procedure AddRef;
```

Remarks

Increments the reference count for the number of references dependent on the TSharedObject object.

See Also

- [Release](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.12.1.7.3.2 Release Method

Decrements the reference count.

Class

[TSharedObject](#)

Syntax

```
procedure Release;
```

Remarks

Call the Release method to decrement the reference count. When RefCount is 1, TSharedObject is deleted from memory.

See Also

- [AddRef](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.12.2 Types

Types in the **MemData** unit.

Types

Name	Description
[TLocateExOptions](#)	Represents the set of [TLocateExOption](#).
[TUpdateRecKinds](#)	Represents the set of TUpdateRecKind.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.12.2.1 TLocateExOptions Set

Represents the set of [TLocateExOption](#).

Unit

MemData

Syntax

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

Request Support   DAC Forum   Provide Feedback

5.12.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

MemData

Syntax

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

Request Support   DAC Forum   Provide Feedback

5.12.3 Enumerations

Enumerations in the **MemData** unit.

Enumerations

Name	Description
TCompressBlobMode	Specifies when the values should be compressed and the way they should be stored.
TConnLostCause	Specifies the cause of the connection loss.
TDANumericType	Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.
TLocateExOption	Allows to set additional search parameters which will be used by the LocateEx

	method.
TSortType	Specifies a sort type for string fields.
TUpdateRecKind	Indicates records for which the ApplyUpdates method will be performed.

Request Support   DAC Forum   Provide Feedback

5.12.3.1 TCompressBlobMode Enumeration

Specifies when the values should be compressed and the way they should be stored.

Unit

MemData

Syntax

```
TCompressBlobMode = (cbNone, cbClient, cbServer, cbClientServer);
```

Values

Value	Meaning
cbClient	Values are compressed and stored as compressed data at the client side. Before posting data to the server decompression is performed and data at the server side stored in the original form. Allows to reduce used client memory due to increase access time to field values. The time spent on the opening DataSet and executing Post increases.
cbClientServer	Values are compressed and stored in compressed form. Allows to decrease the volume of used memory at client and server sides. Access time to the field values increases as for cbClient. The time spent on opening DataSet and executing Post decreases.**Note:** On using cbServer or cbClientServer data on the server is stored as compressed. Other applications can add records in uncompressed format but can't read and write already compressed data. If compressed BLOB is partially changed by another application (if signature was not changed), DAC will consider its value as NULL.Blob compression is not applied to Memo fields because of possible cutting.
cbNone	Values not compressed. The default value.
cbServer	Values are compressed before passing to the server and store at

	the server in compressed form. Allows to decrease database size on the server. Access time to the field values does not change. The time spent on opening DataSet and executing Post usually decreases.

Request Support   DAC Forum   Provide Feedback

5.12.3.2 TConnLostCause Enumeration

Specifies the cause of the connection loss.

Unit

MemData

Syntax

```
TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply,
clServiceQuery, clTransStart, clConnectionApply, clConnect);
```

Values

Value	Meaning
clApply	Connection loss detected during DataSet.ApplyUpdates (Reconnect/Reexecute possible).
clConnect	Connection loss detected during connection establishing (Reconnect possible).
clConnectionApply	Connection loss detected during Connection.ApplyUpdates (Reconnect/Reexecute possible).
clExecute	Connection loss detected during SQL execution (Reconnect with exception is possible).
clOpen	Connection loss detected during execution of a SELECT statement (Reconnect with exception possible).
clRefresh	Connection loss detected during query opening (Reconnect/ Reexecute possible).
clServiceQuery	Connection loss detected during service information request (Reconnect/Reexecute possible).
clTransStart	Connection loss detected during transaction start (Reconnect/ Reexecute possible). clTransStart has less priority then clConnectionApply.
clUnknown	The connection loss reason is unknown.

Request Support   DAC Forum   Provide Feedback

5.12.3.3 TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

Unit

MemData

Syntax

```
TDANumericType = (ntFloat, ntBCD, ntFmtBCD);
```

Values

Value	Meaning
ntBCD	Data is stored on the client side as currency and represented as TBCDField. This format allows storing data with precision up to 0,0001.
ntFloat	Data stored on the client side is in double format and represented as TFloatField. The default value.
ntFmtBCD	Data on client is in TBCD format and is represented as TFMTBCDField.Allows to reflect the whole range of possible values for the MySQL NUMERIC type without accuracy losses. Fields of this type are processed quite slowly.

Request Support   DAC Forum   Provide Feedback

5.12.3.4 TLocateExOption Enumeration

Allows to set additional search parameters which will be used by the LocateEx method.

Unit

MemData

Syntax

```
TLocateExOption = (lxCaseInsensitive, lxPartialKey, lxNearest,
lxNext, lxUp, lxPartialCompare);
```

Values

Value	Meaning
lxCaseInsensitive	Similar to loCaseInsensitive. Key fields and key values are matched without regard to the case.
lxNearest	LocateEx moves the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. For this option to work correctly dataset should be sorted by the fields the search is performed in. If dataset is not sorted, the function may return a line that is not connected with the search condition.
lxNext	LocateEx searches from the current record.
lxPartialCompare	Similar to lxPartialKey, but the difference is that it can process value entries in any position. For example, 'HAM' would match both 'HAMM', 'HAMMER.', and also 'MR HAMMER'.
lxPartialKey	Similar to loPartialKey. Key values can include only a part of the matching key field value. For example, 'HAM' would match both 'HAMM' and 'HAMMER.', but not 'MR HAMMER'.
lxUp	LocateEx searches from the current record to the first record.

Request Support   DAC Forum   Provide Feedback

5.12.3.5 TSortType Enumeration

Specifies a sort type for string fields.

Unit

MemData

Syntax

```
TSortType = (stCaseSensitive, stCaseInsensitive, stBinary);
```

Values

Value	Meaning
stBinary	Sorting by character ordinal values (this comparison is also case sensitive).
stCaseInsensitive	Sorting without case sensitivity.
stCaseSensitive	Sorting with case sensitivity.

5.12.3.6 TUpdateRecKind Enumeration

Indicates records for which the ApplyUpdates method will be performed.

Unit

MemData

Syntax

```
TUpdateRecKind = (ukUpdate, ukInsert, ukDelete);
```

Values

Value	Meaning
ukDelete	ApplyUpdates will be performed for deleted records.
ukInsert	ApplyUpdates will be performed for inserted records.
ukUpdate	ApplyUpdates will be performed for updated records.

5.13   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.

Types

Name	Description
TJsonOptions	Additional options used when saving the dataset in the JSON format.

Enumerations

Name	Description
[TJsonOption](#)	Additional options used when saving the dataset to the JSON format.
[TXmlIndentFormat](#)	Specifies the indentation format in XML format files or streams when saving the dataset using the [TMemDataSet.SaveToXML](#) methods.

Variables

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

© 1997-2025
Devart. All Rights
Reserved.
[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.13.1 Classes

Classes in the **MemDS** unit.

Classes

Name	Description
[TMemDataSet](#)	A base class for working with data and manipulating data in memory.

© 1997-2025
Devart. All Rights
[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Reserved.

5.13.1.1 TMemDataSet Class

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

For a list of all members of this type, see TMemDataSet members.

Unit

MemDS

Syntax

```
TMemDataSet = class(TDataSet);
```

Remarks

TMemDataSet derives from the TDataSet database-engine independent set of properties, events, and methods for working with data and introduces additional techniques to store and manipulate data in memory.

Request Support   DAC Forum   Provide Feedback

5.13.1.1.1 Members

TMemDataSet class overview.

Properties

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive	Specifies the upper and lower boundaries for a range.
LocalConstraints	Used to avoid setting the Required property of a TField component for NOT

	NULL fields at the time of opening TMemDataSet.
[LocalUpdate](#)	Used to prevent implicit update of rows on database server.
[Prepared](#)	Determines whether a query is prepared for execution or not.
[Ranged](#)	Indicates whether a range is applied to a dataset.
[UpdateRecordTypes](#)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#)	Used to check the status of the cached updates buffer.

Methods

Name	Description
[ApplyRange](#)	Applies a range to the dataset.
[ApplyUpdates](#)	Overloaded. Writes dataset's pending cached updates to a database.
[CancelRange](#)	Removes any ranges currently in effect for a dataset.
[CancelUpdates](#)	Clears all pending cached updates from cache and restores dataset in its prior state.
[CommitUpdates](#)	Clears the cached updates buffer.
[DeferredPost](#)	Makes permanent changes to the database server.
[EditRangeEnd](#)	Enables changing the ending value for an existing range.
[EditRangeStart](#)	Enables changing the starting value for an existing range.
[GetBlob](#)	Overloaded. Retrieves TBlob object for a field or

	current record when only its name or the field itself is known.
[Locate](#)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
[LocateEx](#)	Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.
[Prepare](#)	Allocates resources and creates field components for a dataset.
[RestoreUpdates](#)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#)	Cancels changes made to the current record when cached updates are enabled.
[SaveToJSON](#)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
[SaveToXML](#)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetRange](#)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord	Occurs when a single update component can not handle the updates.

Request Support   **DAC Forum**   **Provide Feedback**

5.13.1.1.2 Properties

Properties of the **TMemDataSet** class.

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

Public

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset

	is sorted.
[KeyExclusive](#)	Specifies the upper and lower boundaries for a range.
[LocalConstraints](#)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
[LocalUpdate](#)	Used to prevent implicit update of rows on database server.
[Prepared](#)	Determines whether a query is prepared for execution or not.
[Ranged](#)	Indicates whether a range is applied to a dataset.
[UpdateRecordTypes](#)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#)	Used to check the status of the cached updates buffer.

See Also

- [TMemDataSet Class](#)

- [TMemDataSet Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.13.1.1.2.1 CachedUpdates Property

Used to enable or disable the use of cached updates for a dataset.

Class

[TMemDataSet](#)

Syntax

```
property CachedUpdates: boolean default False;
```

Remarks

Use the CachedUpdates property to enable or disable the use of cached updates for a dataset. Setting CachedUpdates to True enables updates to a dataset (such as posting changes, inserting new records, or deleting records) to be stored in an internal cache on the client side instead of being written directly to the dataset's underlying database tables. When changes are completed, an application writes all cached changes to the database in the context of a single transaction.

Cached updates are especially useful for client applications working with remote database servers. Enabling cached updates brings up the following benefits:

• Fewer transactions and shorter transaction times.

• Minimized network traffic.

The potential drawbacks of enabling cached updates are:

• Other applications can access and change the actual data on the server while users are editing local copies of data, resulting in an update conflict when cached updates are applied to the database.

• Other applications cannot access data changes made by an application until its cached updates are applied to the database.

The default value is False.

Note: When establishing master/detail relationship the CachedUpdates property of detail dataset works properly only when TDADataSetOptions.LocalMasterDetail is set to True.

See Also
• UpdatesPending

• TMemDataSet.ApplyUpdates

• RestoreUpdates

• CommitUpdates

• CancelUpdates

• UpdateStatus

• TCustomDADataSet.Options

Request Support   DAC Forum   Provide Feedback

5.13.1.1.2.2 IndexFieldNames Property

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

Class

`TMemDataSet`

Syntax

```
property IndexFieldNames: string;
```

Remarks

Use the IndexFieldNames property to get or set the list of fields on which the recordset is sorted. Specify the name of each column in IndexFieldNames to use as an index for a table. Column names order is significant. Separate names with semicolons. The specified columns don't need to be indexed. Set IndexFieldNames to an empty string to reset the recordset to the sort order originally used when the recordset's data was first retrieved.

Each field may optionally be followed by the keyword ASC / DESC or CIS / CS / BIN.

Use ASC, DESC keywords to specify a sort order for the field. If one of these keywords is not used, the default sort order for the field is ascending.

Use CIS, CS or BIN keywords to specify the sort type for string fields:

CIS - compare without case sensitivity;

CS - compare with case sensitivity;

BIN - compare by character ordinal values (this comparison is also case sensitive).

If a dataset uses a TCustomDAConnection component, the default value of the sort type depends on the TCustomDAConnection.Options option of the connection. If a dataset does not use a connection (T:Devart.VirtualDac.TVirtualTable dataset), the default is CS.

Read IndexFieldNames to determine the field or fields on which the recordset is sorted.

Sorting is performed locally.

Note:

You cannot sort by BLOB fields.

IndexFieldNames cannot be set to True when TCustomDADataSet.UniDirectional=True.

Example

The following procedure illustrates how to set IndexFieldNames in response to a button click:

```
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

Request Support   DAC Forum   Provide Feedback

5.13.1.1.2.3 KeyExclusive Property

Specifies the upper and lower boundaries for a range.

Class

TMemDataSet

Syntax

```
property KeyExclusive: Boolean;
```

Remarks

Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values.

By default, KeyExclusive is False, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to True.

See Also

* SetRange

* SetRangeEnd

* SetRangeStart

Request Support   DAC Forum   Provide Feedback

Reserved.

5.13.1.1.2.4 LocalConstraints Property

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

Class

TMemDataSet

Syntax

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

Remarks

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

LocalConstraints is obsolete, and is only included for backward compatibility.

The default value is True.

Request Support   DAC Forum   Provide Feedback

5.13.1.1.2.5 LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

TMemDataSet

Syntax

```
property LocalUpdate: boolean default False;
```

Remarks

Set the LocalUpdate property to True to prevent implicit update of rows on database server.

Data changes are cached locally in client memory.

Request Support   DAC Forum   Provide Feedback

5.13.1.1.2.6 Prepared Property

Determines whether a query is prepared for execution or not.

Class

`TMemDataSet`

Syntax

```
property Prepared: boolean;
```

Remarks

Determines whether a query is prepared for execution or not. The Prepared property currently is not supported by MySQL and is always False.

See Also

- Prepare

Request Support   DAC Forum   Provide Feedback

5.13.1.1.2.7 Ranged Property

Indicates whether a range is applied to a dataset.

Class

`TMemDataSet`

Syntax

```
property Ranged: Boolean;
```

Remarks

Use the Ranged property to detect whether a range is applied to a dataset.

See Also
- SetRange

- SetRangeEnd

- SetRangeStart

Request Support   DAC Forum   Provide Feedback

5.13.1.1.2.8 UpdateRecordTypes Property

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

Class

TMemDataSet

Syntax

```
property UpdateRecordTypes: TUpdateRecordTypes default
[rtModified, rtInserted, rtUnmodified];
```

Remarks

Use the UpdateRecordTypes property to determine the update status for the current record when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateRecordTypes offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of records.

See Also
- CachedUpdates

Request Support   DAC Forum   Provide Feedback

5.13.1.1.2.9 UpdatesPending Property

Used to check the status of the cached updates buffer.

Class

[TMemDataSet](#)

Syntax

```
property UpdatesPending: boolean;
```

Remarks

Use the UpdatesPending property to check the status of the cached updates buffer. If UpdatesPending is True, then there are edited, deleted, or inserted records remaining in local cache and not yet applied to the database. If UpdatesPending is False, there are no such records in the cache.

See Also

- [CachedUpdates](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.13.1.1.3 Methods

Methods of the **TMemDataSet** class.

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

Public

Name	Description
[ApplyRange](#)	Applies a range to the dataset.
[ApplyUpdates](#)	Overloaded. Writes dataset's pending cached updates to a database.
[CancelRange](#)	Removes any ranges currently in effect for a dataset.
[CancelUpdates](#)	Clears all pending cached updates from cache and restores dataset in its prior state.

[CommitUpdates](#)	Clears the cached updates buffer.
[DeferredPost](#)	Makes permanent changes to the database server.
[EditRangeEnd](#)	Enables changing the ending value for an existing range.
[EditRangeStart](#)	Enables changing the starting value for an existing range.
[GetBlob](#)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
[Locate](#)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
[LocateEx](#)	Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.
[Prepare](#)	Allocates resources and creates field components for a dataset.
[RestoreUpdates](#)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#)	Cancels changes made to the current record when cached updates are enabled.
[SaveToJSON](#)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
[SaveToXML](#)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetRange](#)	Sets the starting and ending

	values of a range, and applies it.
[SetRangeEnd](#)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
[UnPrepare](#)	Frees the resources allocated for a previously prepared query on the server and client sides.
[UpdateResult](#)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
[UpdateStatus](#)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TMemDataSet Class](#)

- [TMemDataSet Class Members](#)

© 1997-2025
Devart. All Rights
Reserved.
    Request Support   **DAC Forum**   **Provide Feedback**

5.13.1.1.3.1 ApplyRange Method

Applies a range to the dataset.

Class

[TMemDataSet](#)

Syntax

```
procedure ApplyRange;
```

Remarks

Call ApplyRange to cause a range established with [SetRangeStart](#) and [SetRangeEnd](#), or [EditRangeStart](#) and [EditRangeEnd](#), to take effect.

When a range is in effect, only those records that fall within the range are available to the application for viewing and editing.

After a call to ApplyRange, the cursor is left on the first record in the range.

See Also

- [CancelRange](#)

- [EditRangeEnd](#)

- [EditRangeStart](#)

- [IndexFieldNames](#)

- [SetRange](#)

- [SetRangeEnd](#)

- [SetRangeStart](#)

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.13.1.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

Class

`TMemDataSet`

Overload List

Name	Description
[ApplyUpdates](#)	Writes dataset's pending cached updates to a database.
[ApplyUpdates(**const** UpdateRecKinds: TUpdateRecKinds)](#)	Writes dataset's pending cached updates of specified records to a database.

Request Support   DAC Forum   Provide Feedback

Writes dataset's pending cached updates to a database.

Class

TMemDataSet

Syntax

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

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

Note: The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

Example

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

```
procedure ApplyButtonClick(Sender: TObject);
begin
  with Query1 do begin
    Connection1.StartTransaction;
    try
      // ... modify data
      ApplyUpdates;          // try to write the updates to the database
      Connection1.Commit;    // on success, commit the changes
```

```
    except
      RestoreUpdates;        // restore update result for applied records
      Connection1.Rollback;  // on failure, undo the changes
      raise;                 // raise the exception to prevent a call to Comm
    end;
    CommitUpdates;           //on success, clear the cache
  end;
end;
```

See Also

- TMemDataSet.CachedUpdates

- TMemDataSet.CancelUpdates

- TMemDataSet.CommitUpdates

- TMemDataSet.UpdateStatus

Request Support   DAC Forum   Provide Feedback

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.

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.3 CancelRange Method

Removes any ranges currently in effect for a dataset.

Class

TMemDataSet

Syntax

```
procedure CancelRange;
```

Remarks

Call CancelRange to remove a range currently applied to a dataset. Canceling a range reenables access to all records in the dataset.

See Also

- ApplyRange

- EditRangeEnd

- EditRangeStart

- IndexFieldNames

- SetRange

- SetRangeEnd

- SetRangeStart

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.4 CancelUpdates Method

Clears all pending cached updates from cache and restores dataset in its prior state.

Class

[TMemDataSet](#)

Syntax

```
procedure CancelUpdates;
```

Remarks

Call the CancelUpdates method to clear all pending cached updates from cache and restore dataset in its prior state.

It restores the dataset to the state it was in when the table was opened, cached updates were last enabled, or updates were last successfully applied to the database.

When a dataset is closed, or the CachedUpdates property is set to False, CancelUpdates is called automatically.

See Also
- [CachedUpdates](#)

- [TMemDataSet.ApplyUpdates](#)

- [UpdateStatus](#)

5.13.1.1.3.5 CommitUpdates Method

Clears the cached updates buffer.

Class

[TMemDataSet](#)

Syntax

```
procedure CommitUpdates;
```

Remarks

Call the CommitUpdates method to clear the cached updates buffer after both a successful call to ApplyUpdates and a database component's Commit method. Clearing the cache after applying updates ensures that the cache is empty except for records that could not be processed and were skipped by the OnUpdateRecord or OnUpdateError event handlers. An application can attempt to modify the records still in cache.

CommitUpdates also checks wether there are pending updates in dataset. And if there are, it calls ApplyUpdates.

Record modifications made after a call to CommitUpdates repopulate the cached update buffer and require a subsequent call to ApplyUpdates to move them to the database.

See Also
- CachedUpdates

- TMemDataSet.ApplyUpdates

- UpdateStatus

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.6 DeferredPost Method

Makes permanent changes to the database server.

Class

TMemDataSet

Syntax

```
procedure DeferredPost;
```

Remarks

Call DeferredPost to make permanent changes to the database server while retaining dataset in its state whether it is dsEdit or dsInsert.

Explicit call to the Cancel method after DeferredPost has been applied does not abandon modifications to a dataset already fixed in database.

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.7 EditRangeEnd Method

Enables changing the ending value for an existing range.

Class

TMemDataSet

Syntax

```
procedure EditRangeEnd;
```

Remarks

Call EditRangeEnd to change the ending value for an existing range.

To specify an end range value, call FieldByName after calling EditRangeEnd.

After assigning a new ending value, call ApplyRange to activate the modified range.

See Also
- ApplyRange

- CancelRange

- EditRangeStart

- IndexFieldNames

- SetRange

- SetRangeEnd

- SetRangeStart

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.8 EditRangeStart Method

Enables changing the starting value for an existing range.

Class

`TMemDataSet`

Syntax

```
procedure EditRangeStart;
```

Remarks

Call EditRangeStart to change the starting value for an existing range.

To specify a start range value, call FieldByName after calling EditRangeStart.

After assigning a new ending value, call ApplyRange to activate the modified range.

See Also

- ApplyRange

- CancelRange

- EditRangeEnd

- IndexFieldNames

- SetRange

- SetRangeEnd

- SetRangeStart

© 1997-2025
Devart. All Rights   Request Support   DAC Forum   Provide Feedback
Reserved.

5.13.1.1.3.9 GetBlob Method

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

Class

`TMemDataSet`

Overload List

Name	Description
GetBlob(Field: TField)	Retrieves TBlob object for a field or current record when the field itself is known.
GetBlob(const FieldName: string)	Retrieves TBlob object for a field or current record when its name is known.

Request Support   **DAC Forum**   **Provide Feedback**

Retrieves TBlob object for a field or current record when the field itself is known.

Class

TMemDataSet

Syntax

```
function GetBlob(Field: TField): TBlob; overload;
```

Parameters

Field
  Holds an existing TField object.

Return Value
  TBlob object that was retrieved.

Remarks

Call the GetBlob method to retrieve TBlob object for a field or current record when only its name or the field itself is known. FieldName is the name of an existing field. The field should have MEMO or BLOB type.

Request Support   **DAC Forum**   **Provide Feedback**

Retrieves TBlob object for a field or current record when its name is known.

Class

TMemDataSet

Syntax

```
function GetBlob(const FieldName: string): TBlob; overload;
```

Parameters

FieldName
  Holds the name of an existing field.

Return Value

  TBlob object that was retrieved.

Example

```
MyQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

See Also

- TBlob

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.10 Locate Method

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

Class

TMemDataSet

Overload List

Name	Description
Locate(**const** KeyFields: **array of** TField; **const** KeyValues: variant; Options: TLocateOptions)	Searches a dataset by the specified fields for a specific record and positions cursor on it.
Locate(**const** KeyFields: **string**; **const** KeyValues: variant; Options: TLocateOptions)	Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Request Support   DAC Forum   Provide Feedback

Searches a dataset by the specified fields for a specific record and positions cursor on it.

Class

[TMemDataSet](#)

Syntax

```
function Locate(const KeyFields: array of TField; const
KeyValues: variant; Options: TLocateOptions): boolean;
reintroduce; overload;
```

Parameters

KeyFields

Holds TField objects in which to search.

KeyValues

Holds the variant that specifies the values to match in the key fields.

Options

Holds additional search latitude when searching in string fields.

Return Value

True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Class

[TMemDataSet](#)

Syntax

```
function Locate(const KeyFields: string; const KeyValues:
variant; Options: TLocateOptions): boolean; overload; override;
```

Parameters

KeyFields

Holds a semicolon-delimited list of field names in which to search.

KeyValues

Holds the variant that specifies the values to match in the key fields.

Options

Holds additional search latitude when searching in string fields.

Return Value

True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

Remarks

Call the Locate method to search a dataset for a specific record and position cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. An example is provided below.

Options is a set that optionally specifies additional search latitude when searching in string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

The Locate function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the TMemDataSet.IndexFieldNames property.

Example

An example of specifying multiple search values:

```
with CustTable do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P',
    '408-431-1000']), [loPartialKey]);
```

See Also

- TMemDataSet.IndexFieldNames

- TMemDataSet.LocateEx

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.11 LocateEx Method

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

Class

TMemDataSet

Overload List

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

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

Class

TMemDataSet

Syntax

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

Parameters

KeyFields
  Holds TField objects to search in.

KeyValues
  Holds the values of the fields to search for.

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

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

Request Support   DAC Forum   Provide Feedback

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

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.12 Prepare Method

Allocates resources and creates field components for a dataset.

Class

`TMemDataSet`

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate resources and create field components for a dataset. To learn whether dataset is prepared or not use the Prepared property.

The MySQL prepared protocol has certain server restrictions, and its work is not always stable. That is why it is advisable to perform test before using preparation in production versions of applications.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- Prepared

- UnPrepare

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.13 RestoreUpdates Method

Marks all records in the cache of updates as unapplied.

Class

TMemDataSet

Syntax

```
procedure RestoreUpdates;
```

Remarks

Call the RestoreUpdates method to return the cache of updates to its state before calling ApplyUpdates. RestoreUpdates marks all records in the cache of updates as unapplied. It is useful when ApplyUpdates fails.

See Also

• CachedUpdates

• TMemDataSet.ApplyUpdates

• CancelUpdates

• UpdateStatus

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.14 RevertRecord Method

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

Class

TMemDataSet

Syntax

```
procedure RevertRecord;
```

Remarks

Call the RevertRecord method to undo changes made to the current record when cached

updates are enabled.

See Also

- CachedUpdates

- CancelUpdates

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.15 SaveToJSON Method

Saves the current dataset data to a file or a stream in the JSON format.

Class

TMemDataSet

Overload List

Name	Description
SaveToJSON(Destination: TStream; Options: TJsonOptions)	Saves the current dataset data to a stream in the JSON format.
SaveToJSON(const FileName: string; Options: TJsonOptions)	Saves the current dataset data to a file in the JSON format.

Request Support   DAC Forum   Provide Feedback

Saves the current dataset data to a stream in the JSON format.

Class

TMemDataSet

Syntax

```
procedure SaveToJSON(Destination: TStream; Options: TJsonOptions =
[]); overload;
```

Parameters

Destination
  Holds a TStream object.

Options

Holds additional options used when saving the dataset. See TJsonOptions for details. By default, no additional options are set.

Remarks

Call the SaveToJSON method to save the current dataset data to a stream in the JSON format. A TStream object must be created and its position must be set in a preferable value.

See Also

- TJsonOptions

- M:Devart.VirtualDac.TVirtualTable.LoadFromFile(System.String,System.Boolean,System.Boolean)

- M:Devart.VirtualDac.TVirtualTable.LoadFromStream(Borland.Vcl.TStream,System.Boolean,System.Boolean)

Request Support   DAC Forum   Provide Feedback

Saves the current dataset data to a file in the JSON format.

Class

TMemDataSet

Syntax

```
procedure SaveToJSON(const FileName: string; Options:
TJsonOptions = []); overload;
```

Parameters

FileName

Holds the name of a destination file.

Options

Holds additional options used when saving the dataset. See TJsonOptions for details. By default, no additional options are set.

Remarks

Call the SaveToJSON method to save the current dataset data to a file in the JSON format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToJSON until the dataset is closed. Other applications can read this file while it is open, but they cannot write to it.

See Also

- TJsonOptions

- M:Devart.VirtualDac.TVirtualTable.LoadFromFile(System.String,System.Boolean,System.Boolean)

- M:Devart.VirtualDac.TVirtualTable.LoadFromStream(Borland.Vcl.TStream,System.Boolean,System.Boolean)

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.16 SaveToXML Method

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

Class

TMemDataSet

Overload List

Name	Description
SaveToXML(Destination: TStream; IndentFormat: TXmlIndentFormat)	Saves the current dataset data to a stream in the XML format compatible with ADO format.
SaveToXML(const FileName: string; IndentFormat: TXmlIndentFormat)	Saves the current dataset data to a file in the XML format compatible with ADO format.

Request Support   DAC Forum   Provide Feedback

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

Class

Reference 693

[TMemDataSet](#)

Syntax

```
procedure SaveToXML(Destination: TStream; IndentFormat:
TXmlIndentFormat = ifSpace); overload;
```

Parameters

Destination

Holds a TStream object.

IndentFormat

Specifies how to format indentation in XML format streams when saving the dataset. The default value is ifSpace. For other values, see the [TXmlIndentFormat](#) class.

Remarks

Call the SaveToXML method to save the current dataset data to a file or stream in XML format, compatible with ADO.

- If the destination file already exists, it will be overwritten.

- The destination stream remains open from the first call to SaveToXML until the dataset is closed. The file can be read by other applications while open, but it cannot be written to during that time.

When saving to a stream, ensure that a TStream object is created, and its position is set to the desired value.

See Also

- M:Devart.VirtualDac.TVirtualTable.LoadFromFile(System.String,System.Boolean,System.Boolean)

- M:Devart.VirtualDac.TVirtualTable.LoadFromStream(Borland.Vcl.TStream,System.Boolean,System.Boolean)

© 1997-2025 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

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

Class

TMemDataSet

Syntax

```
procedure SaveToXML(const FileName: string; IndentFormat:
TXmlIndentFormat = ifSpace); overload;
```

Parameters

FileName

  Holds the name of a destination file.

IndentFormat

  Specifies how to format indentation in XML format files when saving the dataset. The default value is ifSpace. For other values, see the TXmlIndentFormat class.

See Also

- M:Devart.VirtualDac.TVirtualTable.LoadFromFile(System.String,System.Boolean,System.Boolean)

- M:Devart.VirtualDac.TVirtualTable.LoadFromStream(Borland.Vcl.TStream,System.Boolean,System.Boolean)

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.17 SetRange Method

Sets the starting and ending values of a range, and applies it.

Class

TMemDataSet

Syntax

```
procedure SetRange(const StartValues: array of System.TVarRec;
const EndValues: array of System.TVarRec; StartExlusive: Boolean
= False; EndExclusive: Boolean = False);
```

Parameters

StartValues

  Indicates the field values that designate the first record in the range. In C++, StartValues_Size is the index of the last value in the StartValues array.

EndValues

Indicates the field values that designate the last record in the range. In C++, EndValues_Size is the index of the last value in the EndValues array.

StartExlusive

lindicates the upper and lower boundaries of the start range.

EndExclusive

Indicates the upper and lower boundaries of the end range.

Remarks

Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently specified range, if any.

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a single procedure call. SetRange performs the following functions:

- 1. Puts the dataset into dsSetKey state.

- 2. Erases any previously specified starting range values and ending range values.

- 3. Sets the start and end range values.

- 4. Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are ignored when performing a search.

See Also
- ApplyRange

- CancelRange

- EditRangeEnd

- EditRangeStart

- IndexFieldNames

- KeyExclusive

- SetRangeEnd

- SetRangeStart

Request Support   DAC Forum   Provide Feedback

Reserved.

5.13.1.1.3.18 SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Class

[TMemDataSet](#)

Syntax

```
procedure SetRangeEnd;
```

Remarks

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL.

Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call [ApplyRange](#) to activate the modified range.

See Also

- [ApplyRange](#)

- [CancelRange](#)

- [EditRangeStart](#)

- [IndexFieldNames](#)

- [SetRange](#)

- [SetRangeStart](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.13.1.1.3.19 SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

```
procedure SetRangeStart;
```

Remarks

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL.

Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call ApplyRange to activate the modified range.

See Also
- ApplyRange
- CancelRange
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.20 UnPrepare Method

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

Class

TMemDataSet

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free the resources allocated for a previously prepared query on the server and client sides.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also
- Prepare

Request Support   DAC Forum   Provide Feedback

5.13.1.1.3.21 UpdateResult Method

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

Class

TMemDataSet

Syntax

```
function UpdateResult: TUpdateAction;
```

Return Value

a value of the TUpdateAction enumeration.

Remarks

Call the UpdateResult method to read the status of the latest call to the ApplyUpdates method while cached updates are enabled. UpdateResult reflects updates made on the records that have been edited, inserted, or deleted.

UpdateResult works on the record by record basis and is applicable to the current record only.

See Also
- CachedUpdates

Request Support　　　**DAC Forum**　　　**Provide Feedback**

5.13.1.1.3.22 UpdateStatus Method

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

Class

TMemDataSet

Syntax

```
function UpdateStatus: TUpdateStatus; override;
```

Return Value

a value of the TUpdateStatus enumeration.

Remarks

Call the UpdateStatus method to determine the current update status for the dataset when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of the dataset.

See Also

- CachedUpdates

Request Support　　　**DAC Forum**　　　**Provide Feedback**

5.13.1.1.4 Events

Events of the **TMemDataSet** class.

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

Public

Name	Description

OnUpdateError	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord	Occurs when a single update component can not handle the updates.

See Also

- TMemDataSet Class

- TMemDataSet Class Members

Request Support   DAC Forum   Provide Feedback

5.13.1.1.4.1 OnUpdateError Event

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

Class

TMemDataSet

Syntax

```
property OnUpdateError: TUpdateErrorEvent;
```

Remarks

Write the OnUpdateError event handler to respond to exceptions generated when cached updates are applied to a database.

E is a pointer to an EDatabaseError object from which application can extract an error message and the actual cause of the error condition. The OnUpdateError handler can use this information to determine how to respond to the error condition.

UpdateKind describes the type of update that generated the error.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler.

The error handler can use the TField.OldValue and TField.NewValue properties to evaluate error conditions and set TField.NewValue to a new value to reapply. In this case, set UpdateAction to uaRetry before exiting.

Note: If a call to ApplyUpdates raises an exception and ApplyUpdates is not called within the context of a try...except block, an error message is displayed. If the OnUpdateError handler cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is displayed twice. To prevent redisplay, set UpdateAction to uaAbort in the error handler.

See Also

- CachedUpdates

Request Support   DAC Forum   Provide Feedback

5.13.1.1.4.2 OnUpdateRecord Event

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

Class

TMemDataSet

Syntax

```
property OnUpdateRecord: TUpdateRecordEvent;
```

Remarks

Write the OnUpdateRecord event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over parameter substitution in update components.

UpdateKind describes the type of update to perform.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

See Also

- CachedUpdates

Request Support   DAC Forum   Provide Feedback

5.13.2 Types

Types in the **MemDS** unit.

Types

Name	Description
TJsonOptions	Additional options used when saving the dataset in the JSON format.

Request Support   DAC Forum   Provide Feedback

5.13.2.1 TJsonOptions Set

Additional options used when saving the dataset in the JSON format.

Unit

MemDS

Syntax

```
TJsonOptions = set of TJsonOption;
```

See Also

- TMemDataSet.SaveToJSON

- TMemDataSet.SaveToJSON

Request Support   DAC Forum   Provide Feedback

5.13.3   Enumerations

Enumerations in the **MemDS** unit.

Enumerations

Name	Description
[TJsonOption](#)	Additional options used when saving the dataset to the JSON format.
[TXmlIndentFormat](#)	Specifies the indentation format in XML format files or streams when saving the dataset using the [TMemDataSet.SaveToXML](#) methods.

© 1997-2025
Devart. All Rights
Reserved.

Request Support   **DAC Forum**   **Provide Feedback**

5.13.3.1 TJsonOption Enumeration

Additional options used when saving the dataset to the JSON format.

Unit

[MemDS](#)

Syntax

```
TJsonOption = (joDataOnly, joDataAsArray, joIncludeNulls);
```

Values

Value	Meaning
joDataAsArray	When the option is set, each row of the dataset is saved as a JSON array. Otherwise (by default), each row of the dataset is saved as a JSON object.
joDataOnly	When the option is set, only the dataset rows are saved to JSON. By default, information about the dataset structure is also saved to JSON.
joIncludeNulls	When the option is set, fields with NULL values are saved to JSON. Otherwise (by default), NULL fields are excluded from JSON.

See Also

- TJsonOptions

- TMemDataSet.SaveToJSON

- TMemDataSet.SaveToJSON

Request Support   DAC Forum   Provide Feedback

5.13.3.2 TXmlIndentFormat Enumeration

Specifies the indentation format in XML format files or streams when saving the dataset using the TMemDataSet.SaveToXML methods.

Unit

MemDS

Syntax

```
TXmlIndentFormat = (ifSpace, ifTabulation, ifNone);
```

Values

Value	Meaning
ifNone	No indentation is used.
ifSpace	Space characters are used for indentation.
ifTabulation	Tabs are used for indentation.

Request Support   DAC Forum   Provide Feedback

5.13.4 Variables

Variables in the **MemDS** unit.

Variables

Name	Description
DoNotRaiseExcetionOnUaFail	An exception will be raised if the value of the

	UpdateAction parameter is uaFail.
[SendDataSetChangeEventAfterOpen](#)	The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.13.4.1 DoNotRaiseExcetionOnUaFail Variable

An exception will be raised if the value of the UpdateAction parameter is uaFail.

Unit

[MemDS](#)

Syntax

```
DoNotRaiseExcetionOnUaFail: boolean = False;
```

Remarks

Starting with MyDAC 5.20.0.12, if the [OnUpdateRecord](#) event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.13.4.2 SendDataSetChangeEventAfterOpen Variable

The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

Unit

[MemDS](#)

Syntax

```
SendDataSetChangeEventAfterOpen: boolean = True;
```

Remarks

Starting with MyDAC 5.20.0.11, the DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

Request Support   DAC Forum   Provide Feedback

5.14 MyAccess

This unit contains implementation of most public classes of MyDAC.

Classes

Name	Description
TCustomMyConnection	A base class for connecting to MySQL server.
TCustomMyConnectionOptions	This class allows setting up the behaviour of the TCustomMyConnection class.
TCustomMyDataSet	A base class defining functionality for the classes derived from it.
TCustomMyStoredProc	A class implementing functionality to access stored procedures on a database server.
TCustomMyTable	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
TMyCommand	A component for execution of SQL statements and stored procedures which do

	not return rowsets.
TMyConnection	A component for setting up and controlling connections to MySQL database server.
TMyConnectionOptions	This class allows setting up the behaviour of the TMyConnection class.
TMyConnectionSSLOptions	This class allows setting up the behaviour of the TMyConnection class.
TMyDataSetOptions	This class allows setting up the behaviour of the TMyDataSet class.
TMyDataSource	TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a form.
TMyEncryptor	The class that performs encrypting and decrypting of data.
TMyMetaData	A component for obtaining metainformation about database objects from the server.
TMyQuery	A component for executing queries and operating record sets. It also provides flexible way to update data.
TMyStoredProc	A component for accessing and executing stored procedures and functions.
TMyTable	A component for retrieving and updating data in a single table without writing SQL statements.
TMyTableOptions	This class allows setting up the behaviour of the TMyTable class.
TMyTransaction	A component for managing transactions.
TMyUpdateSQL	A component for tuning update operations for the DataSet component.

Types

Name	Description
[TMyUpdateExecuteEvent](#)	This type is used for the [TCustomMyDataSet.AfterUpdateExecute](#) and [TCustomMyDataSet.BeforeUpdateExecute](#) events.

Enumerations

Name	Description
[TLockRecordType](#)	Specifies the type of the record locking.
[TLockType](#)	Specifies the type of the table locking.
[TMyIsolationLevel](#)	Specifies the xtent to which all outside transactions interfere with subsequent transactions of current connection.

Routines

Name	Description
[GetServerList](#)	Returns the list of the MySQL servers in LAN. MySQL server does not provide usual ways of such list getting, so it can be incomplete.

Constants

Name	Description
[MyDACVersion](#)	Read this constant to get current version number for MyDAC.

Request Support   DAC Forum   Provide Feedback

5.14.1 Classes

Classes in the **MyAccess** unit.

Classes

Name	Description
[TCustomMyConnection](#)	A base class for connecting to MySQL server.
[TCustomMyConnectionOptions](#)	This class allows setting up the behaviour of the TCustomMyConnection class.
[TCustomMyDataSet](#)	A base class defining functionality for the classes derived from it.
[TCustomMyStoredProc](#)	A class implementing functionality to access stored procedures on a database server.
[TCustomMyTable](#)	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
[TMyCommand](#)	A component for execution of SQL statements and stored procedures which do not return rowsets.
[TMyConnection](#)	A component for setting up and controlling connections to MySQL database server.
[TMyConnectionOptions](#)	This class allows setting up the behaviour of the TMyConnection class.
[TMyConnectionSSLOptions](#)	This class allows setting up the behaviour of the TMyConnection class.
[TMyDataSetOptions](#)	This class allows setting up the behaviour of the TMyDataSet class.
[TMyDataSource](#)	TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a

	form.
[TMyEncryptor](#)	The class that performs encrypting and decrypting of data.
[TMyMetaData](#)	A component for obtaining metainformation about database objects from the server.
[TMyQuery](#)	A component for executing queries and operating record sets. It also provides flexible way to update data.
[TMyStoredProc](#)	A component for accessing and executing stored procedures and functions.
[TMyTable](#)	A component for retrieving and updating data in a single table without writing SQL statements.
[TMyTableOptions](#)	This class allows setting up the behaviour of the TMyTable class.
[TMyTransaction](#)	A component for managing transactions.
[TMyUpdateSQL](#)	A component for tuning update operations for the DataSet component.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.1 TCustomMyConnection Class

A base class for connecting to MySQL server.

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

Unit

[MyAccess](#)

Syntax

```
TCustomMyConnection = class(TCustomDAConnection);
```

Remarks

The TCustomMyConnection component is used to establish connection to database server, provide customized login support, and perform transaction control. TCustomMyConnection is the base component for connecting to MySQL server.

Inheritance Hierarchy

TCustomDAConnection

  TCustomMyConnection

See Also

- TMyConnection

- TMyEmbConnection

    Request Support   DAC Forum   Provide Feedback

5.14.1.1.1 Members

TCustomMyConnection class overview.

Properties

Name	Description
ClientVersion	Contains the version of the MySQL Client library.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectionTimeout	Used to specify the amount of time to attempt to establish a connection.
ConnectString (inherited from **TCustomDAConnection**)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Database	Used to specify a 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 extent to which all outside transactions interfere with subsequent transactions of current connection.
LastWarningCount	Used to get the number of warnings resulting from the execution of the most recent statement.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the behaviour of the TMyConnectionOptions 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	Holds the version of MySQL server.
ThreadId	Used to return the thread ID of the current connection.
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 TCustomMyConnection components.
[Commit](#) (inherited from **[TCustomDAConnection](#)**)	Commits current transaction.
[Connect](#) (inherited from **[TCustomDAConnection](#)**)	Establishes a connection to the server.
[CreateDataSet](#)	Returns a new instance of TCustomMyDataSet class and associates it with this connection object.
[CreateSQL](#) (inherited from **[TCustomDAConnection](#)**)	Creates a component for queries execution.
[Disconnect](#) (inherited from **[TCustomDAConnection](#)**)	Performs disconnect.
[ExecProc](#) (inherited from **[TCustomDAConnection](#)**)	Allows to execute stored procedure or function providing its name and parameters.
[ExecProcEx](#) (inherited from **[TCustomDAConnection](#)**)	Allows to execute a stored procedure or function.
[ExecSQL](#)	Executes any SQL statement outside [TMyQuery](#) or [TMyCommand](#) components.
[ExecSQLEx](#) (inherited from **[TCustomDAConnection](#)**)	Executes any SQL statement outside the TQuery or TSQL components.
[GetCharsetNames](#)	Populates a string list with the names of available charsets.
[GetDatabaseNames](#) (inherited from **[TCustomDAConnection](#)**)	Returns a database list from the server.
[GetExecuteInfo](#)	Returns the result of the last query execution.
[GetKeyFieldNames](#) (inherited from **[TCustomDAConnection](#)**)	Provides a list of available key field names.
[GetStoredProcNames](#) (inherited from **[TCustomDAConnection](#)**)	Returns a list of stored procedures from the server.

GetTableNames (inherited from **TCustomDAConnection**)	Provides a list of available tables names.
GetTriggerNames	Returns a list of triggers from the server.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
PerformConnect (inherited from **TCustomDAConnection**)	Establishes a connection to the server regardless of the LoginPrompt property.
Ping (inherited from **TCustomDAConnection**)	Used to check state of connection to the server.
ReleaseSavepoint	Releases the specified savepoint without affecting any work that has been performed after its creation.
RemoveFromPool (inherited from **TCustomDAConnection**)	Marks the connection that should not be returned to the pool after disconnect.
Rollback (inherited from **TCustomDAConnection**)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can roll back later.
StartTransaction (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.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.1.2 Properties

Properties of the **TCustomMyConnection** class.

For a complete list of the **TCustomMyConnection** class members, see the

TCustomMyConnection Members topic.

Public

Name	Description
ClientVersion	Contains the version of the MySQL Client library.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectionTimeout	Used to specify the amount of time to attempt to establish a connection.
ConnectString (inherited from **TCustomDAConnection**)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Database	Used to specify a 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 extent to which all outside transactions interfere with subsequent transactions of current connection.
LastWarningCount	Used to get the number of warnings resulting from the execution of the most recent statement.

LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the behaviour of the TMyConnectionOptions 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	Holds the version of MySQL server.
ThreadId	Used to return the thread ID of the current connection.
Username (inherited from **TCustomDAConnection**)	Used to supply a user name for login.

See Also

- TCustomMyConnection Class

- TCustomMyConnection Class Members

    Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.1.2.1 ClientVersion Property

Contains the version of the MySQL Client library.

Class

`TCustomMyConnection`

Syntax

```
property ClientVersion: string;
```

Remarks

Contains the version of the MySQL Client library (libmysql.dll or libmysqld.dll).

See Also

- TCustomDAConnection.Connect

Request Support   DAC Forum   Provide Feedback

5.14.1.1.2.2 ConnectionTimeout Property

Used to specify the amount of time to attempt to establish a connection.

Class

TCustomMyConnection

Syntax

```
property ConnectionTimeout: integer default
DefValConnectionTimeout;
```

Remarks

Use the ConnectionTimeout property to specify the amount of time to attempt to establish a connection.

Use ConnectionTimeout property to specify the amount of time, in seconds, that can be expired before an attempt to make a connection is considered unsuccessful.

The default value is 15 seconds.

See Also

- TCustomDAConnection.Connect

- TCustomMyDataSet.CommandTimeout

- TMyCommand.CommandTimeout

Request Support   DAC Forum   Provide Feedback

5.14.1.1.2.3 Database Property

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

Class

TCustomMyConnection

Syntax

```
property Database: string;
```

Remarks

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

Altering Database property makes new database name take effect immediately.

Setting Database='mysql' allows you to omit database specifier in SELECT statements. That is, instead of

```
SELECT * FROM mysql.user;
```
you may just write

```
SELECT * FROM user
```

See Also

- TCustomDAConnection.Server

- TMyConnection.Port

- TCustomDAConnection.Username

- TCustomDAConnection.Password

- TCustomDAConnection.GetDatabaseNames

Request Support   DAC Forum   Provide Feedback

5.14.1.1.2.4 IsolationLevel Property

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

Class

[TCustomMyConnection](#)

Syntax

```
property IsolationLevel: TMyIsolationLevel default
ilReadCommitted;
```

Remarks

Use the IsolationLevel property to specify the extent to which all outside transactions interfere with subsequent transactions of current connection.

Changes to IsolationLevel take effect at a time of starting new transaction or opening new connection.

See Also

- [TCustomDAConnection.StartTransaction](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.1.2.5 LastWarningCount Property

Used to get the number of warnings resulting from the execution of the most recent statement.

Class

[TCustomMyConnection](#)

Syntax

```
property LastWarningCount: cardinal;
```

Remarks

Use the LastWarningCount property to get the number of warnings resulting from the execution of the most recent statement.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Reserved.

5.14.1.1.2.6 Options Property

Specifies the behaviour of the TMyConnectionOptions object.

Class

`TCustomMyConnection`

Syntax

```
property Options: TCustomMyConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMyConnectionOptions object.

Descriptions of all options are in the table below.

Option Name	Description
Charset	Used to set a character set used by the client.
NullForZeroDelphiDate	Used to hide the '30-12-1899' dates.
NumericType	Used to specify the format of storing and representation of the NUMERIC (DECIMAL) fields for all TCustomMyDataSets, associated with the given connection.
OptimizedBigInt	Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.
UseUnicode	Used to inform server that all data between client and server sides will be passed in Utf8 coding.

See Also

- TCustomDAConnection.Server
- Database
- National Characters

Request Support   DAC Forum   Provide Feedback

Reserved.

5.14.1.1.2.7 ServerVersion Property

Holds the version of MySQL server.

Class

TCustomMyConnection

Syntax

```
property ServerVersion: string;
```

Remarks

The version of MySQL server.

See Also

* TCustomDAConnection.Connect

Request Support   DAC Forum   Provide Feedback

5.14.1.1.2.8 ThreadId Property

Used to return the thread ID of the current connection.

Class

TCustomMyConnection

Syntax

```
property ThreadId: cardinal;
```

Remarks

Use the ThreadId property to return the thread ID of the current connection. This value can be used as an argument to TMyServerControl.KillProcess to kill the thread.

See Also

* TMyServerControl.KillProcess

5.14.1.1.3 Methods

Methods of the **TCustomMyConnection** class.

For a complete list of the **TCustomMyConnection** class members, see the

TCustomMyConnection Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect	Shares database connection between the TCustomMyConnection components.
Commit (inherited from **TCustomDAConnection**)	Commits current transaction.
Connect (inherited from **TCustomDAConnection**)	Establishes a connection to the server.
CreateDataSet	Returns a new instance of TCustomMyDataSet class and associates it with this connection object.
CreateSQL (inherited from **TCustomDAConnection**)	Creates a component for queries execution.
Disconnect (inherited from **TCustomDAConnection**)	Performs disconnect.
ExecProc (inherited from **TCustomDAConnection**)	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx (inherited from **TCustomDAConnection**)	Allows to execute a stored procedure or function.
ExecSQL	Executes any SQL statement outside TMyQuery or TMyCommand components.
ExecSQLEx (inherited from **TCustomDAConnection**)	Executes any SQL statement outside the TQuery or TSQL

	components.
GetCharsetNames	Populates a string list with the names of available charsets.
GetDatabaseNames (inherited from **TCustomDAConnection**)	Returns a database list from the server.
GetExecuteInfo	Returns the result of the last query execution.
GetKeyFieldNames (inherited from **TCustomDAConnection**)	Provides a list of available key field names.
GetStoredProcNames (inherited from **TCustomDAConnection**)	Returns a list of stored procedures from the server.
GetTableNames (inherited from **TCustomDAConnection**)	Provides a list of available tables names.
GetTriggerNames	Returns a list of triggers from the server.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
PerformConnect (inherited from **TCustomDAConnection**)	Establishes a connection to the server regardless of the LoginPrompt property.
Ping (inherited from **TCustomDAConnection**)	Used to check state of connection to the server.
ReleaseSavepoint	Releases the specified savepoint without affecting any work that has been performed after its creation.
RemoveFromPool (inherited from **TCustomDAConnection**)	Marks the connection that should not be returned to the pool after disconnect.
Rollback (inherited from **TCustomDAConnection**)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can

	roll back later.
StartTransaction (inherited from **TCustomDAConnection**)	Begins a new user transaction.

See Also

- TCustomMyConnection Class

- TCustomMyConnection Class Members

Request Support   DAC Forum   Provide Feedback

5.14.1.1.3.1 AssignConnect Method

Shares database connection between the TCustomMyConnection components.

Class

TCustomMyConnection

Syntax

```
procedure AssignConnect(Source: TCustomMyConnection); virtual;
```

Parameters

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

Remarks

Use the AssignConnect method to share database connection between the TCustomMyConnection components.

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

Request Support   DAC Forum   Provide Feedback

5.14.1.1.3.2 CreateDataSet Method

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

Class

TCustomMyConnection

Syntax

```
function CreateDataSet(AOwner: TComponent = nil):
TCustomDADataSet; override;
```

Return Value
a new instance of TCustomMyDataSet class.

Remarks

The CreateDataSet method returns a new instance of TCustomMyDataSet class and associates it with this connection object.

See Also

- M:Devart.MyDac.TCustomMyConnection.CreateCommand()

Request Support   DAC Forum   Provide Feedback

5.14.1.1.3.3 ExecSQL Method

Executes any SQL statement outside TMyQuery or TMyCommand components.

Class

TCustomMyConnection

Syntax

```
function ExecSQL(const Text: string; const Params: array of
variant): variant; override;
```

Parameters

Text
  Holds the SQL statement.

Params
  Holds the array of the parameters values arranged in the same order as they appear in the SQL statement.

Return Value
  Null.

Remarks

Call the ExecSQL method to execute any SQL statement outside TMyQuery or TMyCommand components. Supply the Params array with values of parameters arranged in the same order as they appear in SQL statement that is passed in Text string parameter.

Note: If a query doesn't have parameters (Params.Count = 0), this query will be executed faster.

Request Support   DAC Forum   Provide Feedback

5.14.1.1.3.4 GetCharsetNames Method

Populates a string list with the names of available charsets.

Class

TCustomMyConnection

Syntax

```
procedure GetCharsetNames(List: TStrings);
```

Parameters

List
  Holds the string list to populate.

Remarks

Call the GetCharsetName method to populate a string list with the names of available charsets.

Note: Any contents already in the target string list object are eliminated and overwritten by the

data produces by GetCharsetNames.

See Also

- Options

Request Support   DAC Forum   Provide Feedback

5.14.1.1.3.5 GetExecuteInfo Method

Returns the result of the last query execution.

Class

`TCustomMyConnection`

Syntax

```
function GetExecuteInfo: string;
```

Return Value

the result of the last query execution.

Remarks

Call the GetExecuteInfo method to returns the result of the last query execution.

The description of the result format you can see at MySQL Reference Manual mysql_info

Note: If you execute a query at TCustomDADataSet.Execute with

TCustomMyDataSet.FetchAll set to False, a result cannot be retrieved.

Example

The method makes sense for the following SQL statements:

```
INSERT INTO ... SELECT ...
INSERT INTO ... VALUES (...),(...),(...)...
LOAD DATA INFILE ...
ALTER TABLE
UPDATE
```

See Also

- TCustomDADataSet.Execute

- TCustomDASQL.Execute

Request Support **DAC Forum** **Provide Feedback**

5.14.1.1.3.6 GetTriggerNames Method

Returns a list of triggers from the server.

Class

TCustomMyConnection

Syntax

```
procedure GetTriggerNames(List: TStrings; AllTriggers: boolean = False);
```

Parameters

List
   A TStrings descendant that will be filled with the names of triggers in the database.
AllTriggers
   True, if triggers from all databases are returned. False is only for the current database.

Remarks

Call the GetTriggerNames method to get the names of triggers. GetTriggerNames populates a string list with the names of triggers in the database. If AllProcs = True, the procedure returns to the List parameter the names of the triggers that belong to all databases; otherwise, List will contain the names of triggers that belong to the current database.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetTriggerNames.

See Also

- TCustomDAConnection.GetDatabaseNames

- TCustomDAConnection.GetTableNames

- TCustomDAConnection.GetStoredProcNames

Request Support **DAC Forum** **Provide Feedback**

5.14.1.1.3.7 ReleaseSavepoint Method

Releases the specified savepoint without affecting any work that has been performed after its creation.

Class

TCustomMyConnection

Syntax

```
procedure ReleaseSavepoint(const Name: string);
```

Parameters

Name
  Holds the savepoint name.

Remarks

Call the ReleaseSavepoint method to release the specified savepoint without affecting any work that has been performed after its creation.

See Also

- RollbackToSavepoint

- Savepoint

Request Support   DAC Forum   Provide Feedback

5.14.1.1.3.8 RollbackToSavepoint Method

Cancels all updates for the current transaction.

Class

TCustomMyConnection

Syntax

```
procedure RollbackToSavepoint(const Name: string);
```

Parameters

Name

Holds the name identifying the last defined savepoint.

Remarks

Call the RollbackToSavepoint method to cancel all updates for the current transaction and restore its state up to the moment of the last defined savepoint.

See Also

- Savepoint

- TCustomDAConnection.Rollback

- ReleaseSavepoint

Request Support   DAC Forum   Provide Feedback

5.14.1.1.3.9 Savepoint Method

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

Class

TCustomMyConnection

Syntax

```
procedure Savepoint(const Name: string);
```

Parameters

Name
  Holds the name of the savepoint.

Remarks

Call the Savepoint method to define a point in the transaction to which you can roll back later. As the parameter, you can pass any valid name to identify the savepoint.

To roll back to the last savepoint call RollbackToSavepoint.

See Also

- RollbackToSavepoint

- ReleaseSavepoint

Request Support	DAC Forum	Provide Feedback

5.14.1.2 TCustomMyConnectionOptions Class

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

For a list of all members of this type, see TCustomMyConnectionOptions members.

Unit

MyAccess

Syntax

```
TCustomMyConnectionOptions = class(TDAConnectionOptions);
```

Inheritance Hierarchy

TDAConnectionOptions

   TCustomMyConnectionOptions

Request Support	DAC Forum	Provide Feedback

5.14.1.2.1 Members

TCustomMyConnectionOptions class overview.

Properties

Name	Description
AllowImplicitConnect (inherited from **TDAConnectionOptions**)	Specifies whether to allow or not implicit connection opening.
Charset	Used to set a character set used by the client.
DefaultSortType (inherited from **TDAConnectionOptions**)	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

DisconnectedMode (inherited from **TDAConnectionOptions**)	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Prevents an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection.On ConnectionLost event occurs, and a failover operation can be performed after the connection is lost.
NullForZeroDelphiDate	Used to hide the '30-12-1899' dates.
NumericType	Used to specify the format of storing and representation of the NUMERIC (DECIMAL) fields for all TCustomMyDataSets, associated with the given connection.
OptimizedBigInt	Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.
UseUnicode	Used to inform server that all data between client and server sides will be passed in Utf8 coding.

Request Support   DAC Forum   Provide Feedback

5.14.1.2.2 Properties

Properties of the **TCustomMyConnectionOptions** class.

For a complete list of the **TCustomMyConnectionOptions** class members, see the

TCustomMyConnectionOptions Members topic.

Public

Name	Description
[Charset](#)	Used to set a character set used by the client.
[DefaultSortType](#) (inherited from [**TDAConnectionOptions**](#))	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the [TMemDataSet.IndexFieldNames](#) property of a dataset.
[DisconnectedMode](#) (inherited from [**TDAConnectionOptions**](#))	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.
[KeepDesignConnected](#) (inherited from [**TDAConnectionOptions**](#))	Prevents 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 the connection is lost.
[NullForZeroDelphiDate](#)	Used to hide the '30-12-1899' dates.
[NumericType](#)	Used to specify the format of storing and representation of the NUMERIC (DECIMAL) fields for all [TCustomMyDataSet](#)s, associated with the given connection.
[OptimizedBigInt](#)	Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.
[UseUnicode](#)	Used to inform server that all data between client and server sides will be passed in Utf8 coding.

Published

Name	Description
AllowImplicitConnect (inherited from **TDAConnectionOptions**)	Specifies whether to allow or not implicit connection opening.

See Also

- TCustomMyConnectionOptions Class

- TCustomMyConnectionOptions Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.2.2.1 Charset Property

Used to set a character set used by the client.

Class

`TCustomMyConnectionOptions`

Syntax

```
property Charset: string;
```

Remarks

Use the Charset property to set a character set used by the client. Actually, if this property is enabled, then the "SET NAMES <Charset>" query is executed on establishing a connection. If the UseUnicode property is set, the Charset property will be ignored. The list of available character sets you can see by executing the TCustomMyConnection.GetCharsetNames method.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.2.2.2 NullForZeroDelphiDate Property

Used to hide the '30-12-1899' dates.

Class

TCustomMyConnectionOptions

Syntax

```
property NullForZeroDelphiDate: boolean default False;
```

Remarks

Use the NullForZeroDelphiDate property to hide the '30-12-1899' dates.

If NullForZeroDelphiDate is set to True, the values of all datetime fields will be changed to Null. If the property is set to False, the '30-12-1899' value will be used as an ordinary date. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.2.2.3 NumericType Property

Used to specify the format of storing and representation of the NUMERIC (DECIMAL) fields for all TCustomMyDataSets, associated with the given connection.

Class

TCustomMyConnectionOptions

Syntax

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

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.14.1.2.2.4 OptimizedBigInt Property

Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.

Class

TCustomMyConnectionOptions

Syntax

```
property OptimizedBigInt: boolean default False;
```

Remarks

Setting this option converts all fields with field length less than 11 of TLargeIntField type into TIntegerField. This allows to process fields that are results of numeric function or cast values as usual Integer fields. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.2.2.5 UseUnicode Property

Used to inform server that all data between client and server sides will be passed in Utf8 coding.

Class

TCustomMyConnectionOptions

Syntax

```
property UseUnicode: boolean default DefValUseUnicode;
```

Remarks

Informs server that all data between client and server sides will be passed in Utf8 coding. Setting this option converts all fields of the TStringField type into TWideStringField that allows to work correctly with symbols of almost all languages simultaneously. On the other hand, it causes a delay in working. If the UseUnicode property is enabled, the Charset property will be ignored. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.3 TCustomMyDataSet Class

A base class defining functionality for the classes derived from it.

For a list of all members of this type, see TCustomMyDataSet members.

Unit

`MyAccess`

Syntax

```
TCustomMyDataSet = class(TCustomDADataSet);
```

Remarks

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

Inheritance Hierarchy

TMemDataSet

  TCustomDADataSet

    TCustomMyDataSet

See Also

- TMyQuery

- TCustomMyTable

- Master/Detail Relationships

Request Support   DAC Forum   Provide Feedback

5.14.1.3.1 Members

TCustomMyDataSet 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.
CommandTimeout	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption	Used to specify encryption options in a dataset.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a

	query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
InsertId	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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	Specifies when to perform locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship

	between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
Options	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used

	when applying a deletion to a record.
[SQLInsert](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[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 specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
[AddWhere](#) (inherited from **[TCustomDADataSet](#)**)	Adds condition to the WHERE clause of SELECT

	statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.

Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.

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 record for the current connection.
LockTable	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshQuick	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when

	cached updates are enabled.
[SaveSQL](#) (inherited from **[TCustomDADataSet](#)**)	Saves the SQL property value to BaseSQL.
[SaveToJSON](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
[SaveToXML](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetOrderBy](#) (inherited from **[TCustomDADataSet](#)**)	Builds an ORDER BY clause of a SELECT statement.
[SetRange](#) (inherited from **[TMemDataSet](#)**)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
[SQLSaved](#) (inherited from **[TCustomDADataSet](#)**)	Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.
[UnLock](#) (inherited from **[TCustomDADataSet](#)**)	Releases a record lock.
[UnLockTable](#)	Releases a table locked by the [TCustomMyDataSet.LockTable](#) method.
[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.

Request Support   DAC Forum   Provide Feedback

5.14.1.3.2 Properties

Properties of the **TCustomMyDataSet** class.

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

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
CommandTimeout	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption	Used to specify encryption options in a dataset.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a

	query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
InsertId	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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	Specifies when to perform locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship

	between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
Options	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used

	when applying a deletion to a record.
[SQLInsert](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[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 specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

See Also

- [TCustomMyDataSet Class](#)

- [TCustomMyDataSet Class Members](#)

5.14.1.3.2.1 CommandTimeout Property

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

Class

TCustomMyDataSet

Syntax

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

Remarks

Use the CommandTimeoutr property to specify the amount of time to attempt execution of a command.

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.

In the case of exceeding waiting time error CR_SERVER_LOST 'Lost connection to MySQL server during query' raises.

The default value is 0 (infinite).

See Also

- TCustomMyConnection.ConnectionTimeout

- TMyCommand.CommandTimeout

Request Support   DAC Forum   Provide Feedback

5.14.1.3.2.2 Connection Property

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

Class

[TCustomMyDataSet](#)

Syntax

```
property Connection: TCustomMyConnection;
```

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 TCustomMyConnection objects.

At runtime, set the Connection property to reference an existing TCustomMyConnection object.

See Also

- [TCustomMyConnection](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.3.2.3 Encryption Property

Used to specify encryption options in a dataset.

Class

[TCustomMyDataSet](#)

Syntax

```
property Encryption: TMyEncryption;
```

Remarks

Set the Encryption options for using encryption in a dataset.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.3.2.4 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TCustomMyDataSet

Syntax

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

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.

The default value is True.

Note: When setting TCustomMyDataSet.FetchAll = False you should keep in mind that execution of such queries blocks current session. By default MyDAC will create additional session if it is necessary. But this can cause the following problems:

- Each additional session runs outside the transaction context thus TMyConnection.TCustomDAConnection.Commit and TMyConnection.TCustomDAConnection.Rollback operations in main session won't apply changes made in additional sessions.

- Temporary tables created in one session are not accessible from other sessions, therefore simultaneous using of FetchAll = False and temporary tables is impossible.

- TCustomMyDataSet.Lock cannot be used

- LockTable cannot be used

In order to avoid creating additional connection you can set Options to False.

See Also
- TCustomDADataSet.FetchRows

Request Support   DAC Forum   Provide Feedback

5.14.1.3.2.5 InsertId Property

Returns the ID generated for an AUTO_INCREMENT column by the previous query.

Class

`TCustomMyDataSet`

Syntax

```
property InsertId: int64;
```

Remarks

Use the InsertId property to return the ID generated for an AUTO_INCREMENT column by the previous query. Use this function after you have performed an INSERT query into a table that contains an AUTO_INCREMENT field.

If the query does not perform an insertion into a table that contains an AUTO_INCREMENT field the value of InsertId won't be defined.

InsertId property has sense only if SQL includes INSERT statement within itself. In case of SELECT statements, the value of auto-increment field can be obtained from corresponding table fields.

See Also
- TCustomDADataSet.Execute

Request Support   DAC Forum   Provide Feedback

5.14.1.3.2.6 LockMode Property

Specifies when to perform locking of an editing record.

Class

`TCustomMyDataSet`

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.

   Request Support   DAC Forum   Provide Feedback

5.14.1.3.2.7 Options Property

Specifies the behaviour of TCustomMyDataSet object.

Class

`TCustomMyDataSet`

Syntax

```
property Options: TMyDataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TCustomMyDataSet object.

Descriptions of all options are in the table below.

Option Name	Description
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on a query execution.
AutoRefresh	Used to automatically refresh dataset every AutoRefreshInterval seconds.

AutoRefreshInterval	Used to define in what time interval in seconds the Refresh or RefreshQuick method of a DataSet is called.
BinaryAsString	Used to specify a method of representation of the BINARY and VARBINARY fields.
CheckRowVersion	Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.
CreateConnection	Used to specify if an additional connection to a server should be established to execute an additional query in the FetchAll=False mode.
DefaultValues	Used to fill the DefaultExpression property of TField objects with appropriate value.
EnableBoolean	Used to specify the method of representation of the TINYINT(1) fields.
FieldsAsString	Used to store all non-BLOB fields as string (native MySQL format).
FullRefresh	Used to specify the fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. Default value is false.
NullForZeroDate	Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).
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 record count for this SELECT so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomMyDataSet to quote all field names in autogenerated SQL statements.
RemoveOnRefresh	Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.

[ReturnParams](#)	Used to return the new value of the fields to dataset after insert or update.
[SaveBlobsAsHex](#)	Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.
[SetEmptyStrToNull](#)	Force replace of empty strings with NULL values in data. The default value is False.
[SetFieldsReadOnly](#)	Used to specify whether fields not belonging to the current updating table get read-only attribute.
[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records does not equal 1.
[TrimFixedChar](#)	Used to specify whether to discard all trailing spaces in string fields of the dataset.
[TrimVarChar](#)	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.3.2.8 SmartFetch Property

The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.

Class

`TCustomMyDataSet`

Syntax

```
property SmartFetch: TSmartFetchOptions;
```

See Also

- TSmartFetchOptions

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.3.2.9 UpdateObject Property

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

Class

[TCustomMyDataSet](#)

Syntax

```
property UpdateObject: TMyUpdateSQL;
```

Remarks

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

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.3.3 Methods

Methods of the **TCustomMyDataSet** class.

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

Public

Name	Description
[AddWhere](#) (inherited from **[TCustomDADataSet](#)**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
[ApplyRange](#) (inherited from **[TMemDataSet](#)**)	Applies a range to the dataset.
[ApplyUpdates](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Writes dataset's pending cached updates to a database.
[BreakExec](#) (inherited from **[TCustomDADataSet](#)**)	Breaks execution of the SQL statement on the server.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.

FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 record for the current connection.
LockTable	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshQuick	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.

SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable	Releases a table locked by the TCustomMyDataSet.LockTable method.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TCustomMyDataSet Class](#)

- [TCustomMyDataSet Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.3.3.1 GetFieldEnum Method

Retrieve the list of acceptable values for a specified field given by the FieldName parameter.

Class

`TCustomMyDataSet`

Syntax

```
procedure GetFieldEnum(List: TStrings; FieldName: string;
TableName: string = '');
```

Parameters

List
  holds the list of acceptable values for a specified field.

FieldName
  Holds the field name.

TableName
  Holds the table name.

Remarks

Call the GetFieldEnum method to retrieve the list of acceptable values for a specified field given by the FieldName parameter. Field should be of the ENUM or the SET type. If GetFieldEnum is called with the TableName parameter empty, TCustomMyDataSet tries to determine table name searching corresponding field name in the fields requested from server.

Example

The code presented in Example 1) demonstrates the usage of the GetFieldEnum method. This code can be tested on the sample table presented in Example 2). The result output in memo is shown in Example 3).

```
Example 1)
```

```
MyQuery.SQL.Text := 'SELECT `id`, `SET_column` FROM tb_with_set_column';
MyQuery.Open;
MyQuery.GetFieldEnum(Memo.Lines, 'SET_column');
Example 2)
DROP TABLE if EXISTS tb_enum;
CREATE TABLE `tb_enum` (
`uid` INT(11) not NULL PRIMARY KEY AUTO_INCREMENT,
`c_enum` ENUM('value1','value2','value3') DEFAULT NULL
);
Example 3)
value1
value2
value3
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.3.3.2 Lock Method

Locks the current record for the current connection.

Class

[TCustomMyDataSet](#)

Overload List

Name	Description
[Lock](#)	Locks the current record for the current connection.
[Lock(LockType: TLockRecordType)](#)	Locks the current record for the current connection.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Locks the current record for the current connection.

Class

[TCustomMyDataSet](#)

Syntax

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

Remarks

Locks the current record for the current connection. Serves to prevent simultaneous editing of the same record by several users. Makes sense only for InnoDB tables and can be called only inside transaction context.

If any other connection tries to modify the locked record, it will wait for the time specified by innodb_lock_wait_timeout server variable, by default 50 seconds.If during this time the record will not be unlocked, an exception raises.

The record is unlocked on finishing transaction (Commit or Rollback).

Note: This method is uncompatible with the TCustomMyDataSet.FetchAll property set to False.

Lock can be used only with queries returning resultset.

See Also
- TCustomDAConnection.StartTransaction

- TCustomDAConnection.Commit

- TCustomDAConnection.Rollback

- TCustomMyDataSet.LockTable

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

Locks the current record for the current connection.

Class

TCustomMyDataSet

Syntax

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

Parameters

LockType
  Holds the type of the record locking.

© 1997-2025
Devart. All Rights

Request Support   DAC Forum   Provide Feedback

Reserved.

5.14.1.3.3.3 LockTable Method

Locks table for the current connection.

Class

[TCustomMyDataSet](#)

Syntax

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

Parameters

LockType
  Specifies the type of the table locking.

Remarks

Call the LockTable method to lock table for the current connection. The main purpose of this method is to speed up working with the table.

Table can be released on:

• calling UnlockTable;

• closing connection;

• calling LockTable once more for the same connection.

If a query has several tables then a table specified in [TMyQuery.UpdatingTable](#) is used.

Note: This method is uncompatible with the [FetchAll](#) property set to False.

See Also

• [UnLockTable](#)

• [TCustomMyDataSet.Lock](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.3.3.4 OpenNext Method

Opens next rowset in the statement.

Class

TCustomMyDataSet

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

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

See Also

- TCustomDADataSet.Execute

Request Support   DAC Forum   Provide Feedback

5.14.1.3.3.5 RefreshQuick Method

Retrieves changes posted to the server by another clients on the client side quickly.

Class

TCustomMyDataSet

Syntax

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

Parameters

CheckDeleted
   True, if records deleted by another clients will be checked additionally.

Remarks

Call the RefreshQuick method to retrieve changes posted to the server by another clients on the client side quickly. The main difference from the Refresh method is that not all data corresponding to the query is retrieved on the client, but only the rows which were added or modified at the moment of the last update. A necessity of data inquiry for each row is defined by TIMESTAMP field.

If CheckDeleted parameter set to True records deleted by another clients will be checked additionally.

For RefreshQuick to work it is necessary that a query includes unique key fields and TIMESTAMP field.

This method is especially effective for queries with huge data level in the single row.

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.

Request Support   DAC Forum   Provide Feedback

5.14.1.3.3.6 UnLockTable Method

Releases a table locked by the LockTable method.

Class

`TCustomMyDataSet`

Syntax

```
procedure UnLockTable;
```

Remarks

Call the UnLockTable method to release a table locked by LockTable.

See Also

* LockTable

* TCustomMyDataSet.Lock

Request Support   DAC Forum   Provide Feedback

5.14.1.3.4 Events

Events of the **TCustomMyDataSet** class.

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

Public

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

See Also

- [TCustomMyDataSet Class](#)

- [TCustomMyDataSet Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.3.4.1 AfterUpdateExecute Event

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

Class

`TCustomMyDataSet`

Syntax

```
property AfterUpdateExecute: TMyUpdateExecuteEvent;
```

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](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.3.4.2 BeforeUpdateExecute Event

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

Class

TCustomMyDataSet

Syntax

```
property BeforeUpdateExecute: TMyUpdateExecuteEvent;
```

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

Request Support   DAC Forum   Provide Feedback

5.14.1.4 TCustomMyStoredProc Class

A class implementing functionality to access stored procedures on a database server.

For a list of all members of this type, see TCustomMyStoredProc members.

Unit

MyAccess

Syntax

```
TCustomMyStoredProc = class(TCustomMyDataSet);
```

Remarks

TCustomMyStoredProc implements functionality to access stored procedures on a database server.

You need only to define the StoredProcName property, while not bothering about writing a

SQL statement manually.

Use the Execute method at runtime to generate a request that instructs server to execute procedure and return parameters in the Params property.

Stored procedures are supported only for MySQL 5.0.

Inheritance Hierarchy

TMemDataSet

   TCustomDADataSet

      TCustomMyDataSet

         TCustomMyStoredProc

See Also

- TCustomMyDataSet

- TMyStoredProc

Request Support   DAC Forum   Provide Feedback

5.14.1.4.1 Members

TCustomMyStoredProc 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.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query

Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.

IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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 (inherited from **TCustomMyDataSet**)	Specifies when to perform locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
Options (inherited from **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.

SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
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 **TCustomMyDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ExecProc	Executes a SQL statement on the server.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is

	fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Builds a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assigns it to the SQL property.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are

	enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the TCustomMyDataSet.LockTable method.
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 **TCustomMyDataSet**)	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 **TCustomMyDataSet**)	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.

© 1997-2025 Devart. All Rights Reserved.   **Request Support**   **DAC Forum**   **Provide Feedback**

5.14.1.4.2 Properties

Properties of the **TCustomMyStoredProc** class.

For a complete list of the **TCustomMyStoredProc** class members, see the

TCustomMyStoredProc Members topic.

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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 (inherited from **TCustomMyDataSet**)	Specifies when to perform locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship

	between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
Options (inherited from **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used

	when applying a deletion to a record.
[SQLInsert](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[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 **[TCustomMyDataSet](#)**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

See Also

- [TCustomMyStoredProc Class](#)

- TCustomMyStoredProc Class Members

Request Support   DAC Forum   Provide Feedback

5.14.1.4.2.1 StoredProcName Property

Used to specify the name of the stored procedure to call on the server.

Class

TCustomMyStoredProc

Syntax

```
property StoredProcName: string;
```

Remarks

Use the StoredProcName property to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, then when the application attempts to prepare the procedure prior to execution, an exception is raised.

Request Support   DAC Forum   Provide Feedback

5.14.1.4.3 Methods

Methods of the **TCustomMyStoredProc** class.

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

Public

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.

ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ExecProc	Executes a SQL statement on the server.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.

Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Builds a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assigns it to the SQL property.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as

	unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the TCustomMyDataSet.LockTable method.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.

[UpdateResult](#) (inherited from **[TMemDataSet](#)**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
[UpdateStatus](#) (inherited from **[TMemDataSet](#)**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TCustomMyStoredProc Class](#)

- [TCustomMyStoredProc Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.4.3.1 ExecProc Method

Executes a SQL statement on the server.

Class

```
TCustomMyStoredProc
```

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](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Reserved.

5.14.1.4.3.2 PrepareSQL Method

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

Class

`TCustomMyStoredProc`

Syntax

```
procedure PrepareSQL;
```

Remarks

Call the PrepareSQL method to build a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property. Then the generated query is verified to be valid and, if necessary, the list of parameters is modified.

PrepareSQL is called implicitly when TCustomMyStoredProc is executed.

See Also
- TCustomDADataSet.Params
- StoredProcName
- ExecProc

Request Support   DAC Forum   Provide Feedback

5.14.1.5 TCustomMyTable 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 TCustomMyTable members.

Unit

`MyAccess`

Syntax

```
TCustomMyTable = class(TCustomMyDataSet);
```

Remarks

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

Inheritance Hierarchy

TMemDataSet
   TCustomDADataSet
      TCustomMyDataSet
        TCustomMyTable

See Also

- TMyTable

- TCustomMyDataSet

- TMyQuery

- Master/Detail Relationships

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.14.1.5.1 Members

TCustomMyTable 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.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
IndexDefs	Contains information about

	the indexes for a table.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TCustomDADataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
Limit	Used to set the number of rows retrieved from the query.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode (inherited from **TCustomMyDataSet**)	Specifies when to perform locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
Offset	Used to allow retrieving data from the server starting from the specified row.
Options	Specifies the behaviour of the TMyTable 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when

	its Open method is called.
[SQLDelete](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
[SQLInsert](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[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 **[TCustomMyDataSet](#)**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
EmptyTable	Deletes all records from the database table specified by

	the TableName property.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Determines KeyFields and build query of TCustomMyTable.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.

RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the

	TCustomMyDataSet.LockTable method.
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 **TCustomMyDataSet**)	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 **TCustomMyDataSet**)	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.

Request Support   DAC Forum   Provide Feedback

5.14.1.5.2 Properties

Properties of the **TCustomMyTable** class.

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

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
IndexDefs	Contains information about the indexes for a table.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TCustomDADataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
Limit	Used to set the number of rows retrieved from the query.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit

	update of rows on database server.
[LockMode](inherited) (inherited from **TCustomMyDataSet**)	Specifies when to perform locking of an editing record.
[MacroCount](inherited) (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
[Macros](inherited) (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
[MasterFields](inherited) (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) (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
[Offset](inherited)	Used to allow retrieving data from the server starting from the specified row.
[Options](inherited)	Specifies the behaviour of the TMyTable object.
[OrderFields](inherited)	Used to build ORDER BY clause of SQL statements.
[ParamCheck](inherited) (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
[ParamCount](inherited) (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
[Params](inherited) (inherited from **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
[Prepared](inherited) (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
[Ranged](inherited) (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.

ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
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 **[TCustomMyDataSet](#)**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

See Also

- [TCustomMyTable Class](#)

- [TCustomMyTable Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.5.2.1 IndexDefs Property

Contains information about the indexes for a table.

Class

`TCustomMyTable`

Syntax

```
property IndexDefs: TIndexDefs;
```

Remarks

The IndexDefs property is used to contain information about the indexes for a table.

IndexDefs is a collection of index definitions, each of which describes an available index for the table.

In contrast to BDE, can be used only for viewing the list of indexes already created for the table.

As an additional request for the server is required to fill it, IndexDefs is filled on the first call.

Request Support   DAC Forum   Provide Feedback

5.14.1.5.2.2 Limit Property

Used to set the number of rows retrieved from the query.

Class

TCustomMyTable

Syntax

```
property Limit: integer default - 1;
```

Remarks

Use the Limit property to set the number of rows retrieved from the query. If Limit is -1, all records will be obtained.

See Also

- Offset

Request Support   DAC Forum   Provide Feedback

5.14.1.5.2.3 Offset Property

Used to allow retrieving data from the server starting from the specified row.

Class

TCustomMyTable

Syntax

```
property Offset: integer default 0;
```

Remarks

Use the Offset property to allow retrieving data from the server starting from the specified row. The default value is 0.

See Also

- Limit

Request Support   DAC Forum   Provide Feedback

5.14.1.5.2.4 Options Property

Specifies the behaviour of the TMyTable object.

Class

`TCustomMyTable`

Syntax

```
property Options: TMyTableOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMyTable object.

Descriptions of all options are in the table below.

Option Name	Description
HandlerIndex	Used to assign an index and a value that this index should satisfy.
UseHandler	Used for the HANDLER statement to be used instead of the SELECT statement.

Request Support   DAC Forum   Provide Feedback

5.14.1.5.2.5 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

[TCustomMyTable](#)

Syntax

```
property OrderFields: string;
```

Remarks

TCustomMyTable uses the OrderFields property to build ORDER BY clause of SQL statements.

Place commas to separate fields in a single string.

TCustomMyTable is reopened when the OrderFields property is being changed.

See Also

- [TCustomMyTable](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.5.2.6 TableName Property

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

Class

[TCustomMyTable](#)

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.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.5.3 Methods

Methods of the **TCustomMyTable** class.

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

Public

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the

	ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
EmptyTable	Deletes all records from the database table specified by the TableName property.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.

Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Determines KeyFields and build query of TCustomMyTable.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range

	of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the TCustomMyDataSet.LockTable method.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TCustomMyTable Class

- TCustomMyTable Class Members

Request Support   DAC Forum   Provide Feedback

5.14.1.5.3.1 EmptyTable Method

Deletes all records from the database table specified by the TableName property.

Class

TCustomMyTable

Syntax

```
procedure EmptyTable;
```

Remarks

Call the EmptyTable method to delete all records from the database table specified by the TableName property.

See Also

- TableName

Request Support   DAC Forum   Provide Feedback

5.14.1.5.3.2 PrepareSQL Method

Determines KeyFields and build query of TCustomMyTable.

Class

TCustomMyTable

Syntax

```
procedure PrepareSQL;
```

Remarks

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

See Also

- OrderFields

- TableName

- TCustomDADataSet.FilterSQL

Request Support   DAC Forum   Provide Feedback

5.14.1.6 TMyCommand Class

A component for execution of SQL statements and stored procedures which do not return rowsets.

For a list of all members of this type, see TMyCommand members.

Unit

MyAccess

Syntax

```
TMyCommand = class(TCustomDASQL);
```

Remarks

Use TMyCommand to access database server using SQL statements.

TMyCommand object in a client application is used mainly to execute SQL statements on database server. SQL statement should not retrieve records from a database since TMyCommand does not provide storage for returned data.

Inheritance Hierarchy

TCustomDASQL

  TMyCommand

See Also

- TMyQuery

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.14.1.6.1 Members

TMyCommand 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 to attempt to execute

	a command.
[Connection](#)	Used to specify a connection object that will be used to connect to a data store.
[Debug](#) (inherited from **TCustomDASQL**)	Used to display the statement that is being executed and the values and types of its parameters.
[FinalSQL](#) (inherited from **TCustomDASQL**)	Used to return a SQL statement with expanded macros.
[InsertId](#)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
[MacroCount](#) (inherited from **TCustomDASQL**)	Used to get the number of macros associated with the Macros property.
[Macros](#) (inherited from **TCustomDASQL**)	Makes it possible to change SQL queries easily.
[ParamCheck](#) (inherited from **TCustomDASQL**)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
[ParamCount](#) (inherited from **TCustomDASQL**)	Indicates the number of parameters in the Params property.
[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.
[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](#) (inherited from **TCustomDASQL**)	Breaks execution of an SQL satatement on the server.
[Execute](#) (inherited from **TCustomDASQL**)	Overloaded. Executes a SQL statement on the server.
[Executing](#) (inherited from **TCustomDASQL**)	Checks whether TCustomDASQL still executes a SQL statement.
[FindMacro](#) (inherited from **TCustomDASQL**)	Finds a macro with the specified name.
[FindParam](#) (inherited from **TCustomDASQL**)	Finds a parameter with the specified name.
[MacroByName](#) (inherited from **TCustomDASQL**)	Finds a macro with the specified name.
[ParamByName](#) (inherited from **TCustomDASQL**)	Finds a parameter with the specified name.
[Prepare](#) (inherited from **TCustomDASQL**)	Allocates, opens, and parses cursor for a query.
[UnPrepare](#) (inherited from **TCustomDASQL**)	Frees the resources allocated for a previously prepared query on the server and client sides.
[WaitExecuting](#) (inherited from **TCustomDASQL**)	Waits until TCustomDASQL executes a SQL statement.

Events

Name	Description
[AfterExecute](#) (inherited from **TCustomDASQL**)	Occurs after a SQL statement has been executed.

Request Support   DAC Forum   Provide Feedback

5.14.1.6.2 Properties

Properties of the **TMyCommand** class.

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

Public

Name	Description
ChangeCursor (inherited from **TCustomDASQL**)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Debug (inherited from **TCustomDASQL**)	Used to display the statement that is being executed and the values and types of its parameters.
FinalSQL (inherited from **TCustomDASQL**)	Used to return a SQL statement with expanded macros.
InsertId	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
MacroCount (inherited from **TCustomDASQL**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDASQL**)	Makes it possible to change SQL queries easily.
ParamCheck (inherited from **TCustomDASQL**)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount (inherited from **TCustomDASQL**)	Indicates the number of parameters in the Params property.
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.

[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.

Published

Name	Description
[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.

See Also

- [TMyCommand Class](#)

- [TMyCommand Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.6.2.1 CommandTimeout Property

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

Class

[TMyCommand](#)

Syntax

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

Remarks

Use the CommandTimeout property to specify the amount of time to attempt to execute a command.

Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Measured in seconds.

If a command executed successfully prior to the expiration of the seconds specified, CommandTimeout has no effect.

In the case of exceeding waiting time error CR_SERVER_LOST 'Lost connection to MySQL server during query' raises.

The default value is 0 (infinite).

See Also
- TCustomMyConnection.ConnectionTimeout

- TCustomMyDataSet.CommandTimeout

Request Support   DAC Forum   Provide Feedback

5.14.1.6.2.2 Connection Property

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

Class

TMyCommand

Syntax

```
property Connection: TCustomMyConnection;
```

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 TMyConnection objects.

At run-time, set the Connection property to reference an existing TMyConnection object.

See Also
* TCustomMyConnection

Request Support   DAC Forum   Provide Feedback

5.14.1.6.2.3 InsertId Property

Returns the ID generated for an AUTO_INCREMENT column by the previous query.

Class
`TMyCommand`

Syntax
```
property InsertId: int64;
```

Remarks

Use the InsertId to return the ID generated for an AUTO_INCREMENT column by the previous query. Use this function after you have performed an INSERT query into a table that contains an AUTO_INCREMENT field.

If the query does not perform an insertion into a table that contains an AUTO_INCREMENT field the value of InsertId won't be defined.

See Also
* TCustomDASQL.Execute

Request Support   DAC Forum   Provide Feedback

5.14.1.7 TMyConnection Class

A component for setting up and controlling connections to MySQL database server.

For a list of all members of this type, see TMyConnection members.

Unit

[MyAccess](#)

Syntax

```
TMyConnection = class(TCustomMyConnection);
```

Remarks

TMyConnection component is used to establish connection to database server, provide customized login support, and perform transaction control. TMyConnection publishes connection-related properties derived from its ancestor class TCustomMyConnection and introduces specific properties.

Inheritance Hierarchy

[TCustomDAConnection](#)

  [TCustomMyConnection](#)

    TMyConnection

See Also

- [TCustomMyConnection](#)

- [TMyEmbConnection](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.7.1 Members

[TMyConnection](#) class overview.

Properties

Name	Description
[ClientVersion](#) (inherited from **[TCustomMyConnection](#)**)	Contains the version of the MySQL Client library.
[ConnectDialog](#) (inherited from **[TCustomDAConnection](#)**)	Allows to link a [TCustomConnectDialog](#) component.
[Connected](#)	Used to establish a database connection.

ConnectionTimeout (inherited from **TCustomMyConnection**)	Used to specify the amount of time to attempt to establish a connection.
ConnectString (inherited from **TCustomDAConnection**)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Database	Used to specify the database name.
HttpOptions	Holds a THttpOptions object that contains settings for HTTP connection.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
IOHandler	Used to assign an external component for communication between MyDAC and MySQL server.
IsolationLevel (inherited from **TCustomMyConnection**)	Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.
LastWarningCount (inherited from **TCustomMyConnection**)	Used to get the number of warnings resulting from the execution of the most recent statement.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the behaviour of TMyConnection object.
Password	Used to specify the password.
Pooling (inherited from **TCustomDAConnection**)	Enables or disables using connection pool.
PoolingOptions (inherited from **TCustomDAConnection**)	Specifies the behaviour of connection pool.

Port	Used to specify the port number for a TCP/IP connection.
Server	Used to specify the server to connect to.
ServerVersion (inherited from **TCustomMyConnection**)	Holds the version of MySQL server.
SSLOptions	Used to set the properties required for protected SSL connection with the server.
ThreadId (inherited from **TCustomMyConnection**)	Used to return the thread ID of the current connection.
Username	Used to specify the username.

Methods

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect (inherited from **TCustomMyConnection**)	Shares database connection between the TCustomMyConnection components.
Commit (inherited from **TCustomDAConnection**)	Commits current transaction.
Connect (inherited from **TCustomDAConnection**)	Establishes a connection to the server.
CreateDataSet (inherited from **TCustomMyConnection**)	Returns a new instance of TCustomMyDataSet class and associates it with this connection object.
CreateSQL (inherited from **TCustomDAConnection**)	Creates a component for queries execution.
Disconnect (inherited from **TCustomDAConnection**)	Performs disconnect.
ExecProc (inherited from **TCustomDAConnection**)	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx (inherited from **TCustomDAConnection**)	Allows to execute a stored procedure or function.
ExecSQL (inherited from **TCustomMyConnection**)	Executes any SQL statement outside TMyQuery

	or [TMyCommand](#) components.
[ExecSQLEx](#) (inherited from **[TCustomDAConnection](#)**)	Executes any SQL statement outside the TQuery or TSQL components.
[GetCharsetNames](#) (inherited from **[TCustomMyConnection](#)**)	Populates a string list with the names of available charsets.
[GetDatabaseNames](#) (inherited from **[TCustomDAConnection](#)**)	Returns a database list from the server.
[GetExecuteInfo](#) (inherited from **[TCustomMyConnection](#)**)	Returns the result of the last query execution.
[GetKeyFieldNames](#) (inherited from **[TCustomDAConnection](#)**)	Provides a list of available key field names.
[GetStoredProcNames](#) (inherited from **[TCustomDAConnection](#)**)	Returns a list of stored procedures from the server.
[GetTableNames](#) (inherited from **[TCustomDAConnection](#)**)	Provides a list of available tables names.
[GetTriggerNames](#) (inherited from **[TCustomMyConnection](#)**)	Returns a list of triggers from the server.
[MonitorMessage](#) (inherited from **[TCustomDAConnection](#)**)	Sends a specified message through the [TCustomDASQLMonitor](#) component.
[PerformConnect](#) (inherited from **[TCustomDAConnection](#)**)	Establishes a connection to the server regardless of the LoginPrompt property.
[Ping](#) (inherited from **[TCustomDAConnection](#)**)	Used to check state of connection to the server.
[ReleaseSavepoint](#) (inherited from **[TCustomMyConnection](#)**)	Releases the specified savepoint without affecting any work that has been performed after its creation.
[RemoveFromPool](#) (inherited from **[TCustomDAConnection](#)**)	Marks the connection that should not be returned to the pool after disconnect.

Rollback (inherited from **TCustomDAConnection**)	Discards all current data changes and ends transaction.
RollbackToSavepoint (inherited from **TCustomMyConnection**)	Cancels all updates for the current transaction.
Savepoint (inherited from **TCustomMyConnection**)	Defines a point in the transaction to which you can roll back later.
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.
OnWarning	Occurs when LastWarningCount is greater than 0.

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2 Properties

Properties of the **TMyConnection** class.

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

Public

Name	Description
ClientVersion (inherited from **TCustomMyConnection**)	Contains the version of the MySQL Client library.

ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectionTimeout (inherited from **TCustomMyConnection**)	Used to specify the amount of time to attempt to establish a connection.
ConnectString (inherited from **TCustomDAConnection**)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
IsolationLevel (inherited from **TCustomMyConnection**)	Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.
LastWarningCount (inherited from **TCustomMyConnection**)	Used to get the number of warnings resulting from the execution of the most recent statement.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Pooling (inherited from **TCustomDAConnection**)	Enables or disables using connection pool.
PoolingOptions (inherited from **TCustomDAConnection**)	Specifies the behaviour of connection pool.
ServerVersion (inherited from **TCustomMyConnection**)	Holds the version of MySQL server.
ThreadId (inherited from **TCustomMyConnection**)	Used to return the thread ID of the current connection.

Published

Name	Description
Connected	Used to establish a database connection.

[Database](#)	Used to specify the database name.
[HttpOptions](#)	Holds a THttpOptions object that contains settings for HTTP connection.
[IOHandler](#)	Used to assign an external component for communication between MyDAC and MySQL server.
[Options](#)	Specifies the behaviour of TMyConnection object.
[Password](#)	Used to specify the password.
[Port](#)	Used to specify the port number for a TCP/IP connection.
[Server](#)	Used to specify the server to connect to.
[SSLOptions](#)	Used to set the properties required for protected SSL connection with the server.
[Username](#)	Used to specify the username.

See Also

- [TMyConnection Class](#)

- [TMyConnection Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.7.2.1 Connected Property

Used to establish a database connection.

Class

[TMyConnection](#)

Syntax

```
property Connected stored IsConnectedStored;
```

Remarks

Indicates whether the database connection is active. Set this property to True to establish a database connection. Setting this property to False will close a connection.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.7.2.2 Database Property

Used to specify the database name.

Class

[TMyConnection](#)

Syntax

```
property Database: string;
```

Remarks

Use the Database property to specify the name of a database to be used as the default source of data for SQL queries after establishing a connection.

Changing the value of this property takes effect immediately.

You can use the Database property to omit the database name in SELECT statments. For example, you can write

```
SELECT * FROM user;
```
instead of

```
SELECT * FROM mysql.user;
```

See Also

- [Server](#)

- [Port](#)

- [Username](#)

- [Password](#)

- [TCustomDAConnection.GetDatabaseNames](#)

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2.3 HttpOptions Property

Holds a THttpOptions object that contains settings for HTTP connection.

Class

TMyConnection

Syntax

```
property HttpOptions: THttpOptions;
```

Remarks

The HttpOptions property holds a THttpOptions object that contains settings for HTTP connection.

For more information on HTTP tunneling refer to the Network Tunneling article.

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2.4 IOHandler Property

Used to assign an external component for communication between MyDAC and MySQL server.

Class

TMyConnection

Syntax

```
property IOHandler: TCRIOHandler;
```

Remarks

Use the IOHandler property to assign an external component for communication between MyDAC and MySQL server. The component must be a descendant of the TMyIOHandler abstract class.

There is an example of implementation and usage of such component in the SecureBridge demo. This component provides integration with the SecureBridge library, so SecureBridge should be also installed to build and install it. You can read more about this demo in the Demo Projects topic, and in the *Readme.html* file located in the the SecureBridge demo directory.

SecureBridge is a library that can be used for ensuring protection of important data transferred between MySQL server and MyDAC based applications through public networks.

See Also

- TMyIOHandler

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2.5 Options Property

Specifies the behaviour of TMyConnection object.

Class

`TMyConnection`

Syntax

```
property Options: TMyConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMyConnection object.

Descriptions of all options are in the table below.

Option Name	Description
Charset	Used to set the character set that MyDAC uses to read and write character data.
CheckBackslashes	Used to check the value of the NO_BACKSLASH_ESCAPES server variable.
Compress	Used to apply compression on transferring data.
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.
[Direct](#)	Used to work without using MySQL client library (libmysql.dll).
[DisconnectedMode](#)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
[Embedded](#)	Used to specify the server that will be used to connect.
[Interactive](#)	Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.
[IPVersion](#)	Used to specify Internet Protocol Version.
[KeepDesignConnected](#)	Used to prevent an application from establishing a connection at startup.
[LocalFailover](#)	If True, the [TCustomDAConnection.OnConnectionLost](#) event occurs and a failover operation can be performed after connection breaks.
[NullForZeroDelphiDate](#)	Used to hide the '30-12-1899' dates.
[NumericType](#)	Specifies the format of storing and representing NUMERIC (DECIMAL) fields.
[OptimizedBigInt](#)	Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.
[Protocol](#)	Used to specify the protocol to use when connecting to server.
[UseUnicode](#)	Used to specify whether the UTF8 charset will be used.

See Also

- [TCustomDAConnection.Server](#)

- [TCustomMyConnection.Database](#)

- [TCustomMyConnection.Options](#)

- [Embedded Server](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Reserved.

5.14.1.7.2.6 Password Property

Used to specify the password.

Class

`TMyConnection`

Syntax

```
property Password: string;
```

Remarks

Use the Password property to specify the password of your MySQL account.

When the value of the property is changed, TMyConnection calls the Disconnect method.

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2.7 Port Property

Used to specify the port number for a TCP/IP connection.

Class

`TMyConnection`

Syntax

```
property Port: integer default MYSQL_PORT;
```

Remarks

Use the Port property to specify the port number for a TCP/IP connection. Note that TCustomDAConnection.Server property determines the type of the connection.

The default value is 3306.

The Port property can be used only if TCustomMyConnection.Options is set to False.

See Also

- TCustomDAConnection.Server

- TCustomMyConnection.Database

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2.8 Server Property

Used to specify the server to connect to.

Class

`TMyConnection`

Syntax

```
property Server: string;
```

Remarks

Use the Server property to specify the server name.

When the value of the property is changed, TMyConnection calls the Disconnect method.

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2.9 SSLOptions Property

Used to set the properties required for protected SSL connection with the server.

Class

`TMyConnection`

Syntax

```
property SSLOptions: TMyConnectionSSLOptions;
```

Remarks

Use the SSLOptions property to set the properties required for protected SSL connection with the server. These properties can be used only for Options.Protocol = mpSSL.

For using ssleay32.dll and libeay32.dll files are needed.

The detailed description of these properties you can find in MySQL Reference Manual

See Also

- Options

Request Support   DAC Forum   Provide Feedback

5.14.1.7.2.10 Username Property

Used to specify the username.

Class

TMyConnection

Syntax

```
property Username: string;
```

Remarks

Use the Password property to specify the username of your MySQL account.

When the value of the property is changed, TMyConnection calls the Disconnect method.

Request Support   DAC Forum   Provide Feedback

5.14.1.7.3 Events

Events of the **TMyConnection** class.

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

Public

Name	Description
OnConnectionLost (inherited from	This event occurs when connection was lost.

TCustomDAConnection)	
OnError (inherited from **TCustomDAConnection**)	This event occurs when an error has arisen in the connection.

Published

Name	**Description**
OnWarning	Occurs when LastWarningCount is greater than 0.

See Also

- TMyConnection Class

- TMyConnection Class Members

Request Support   DAC Forum   Provide Feedback

5.14.1.7.3.1 OnWarning Event

Occurs when LastWarningCount is greater than 0.

Class

TMyConnection

Syntax

```
property OnWarning: TMyWarningEvent;
```

Remarks

The OnWarning event is triggered when the value of

TCustomMyConnection.LastWarningCount is greater than 0.

Request Support   DAC Forum   Provide Feedback

5.14.1.8 TMyConnectionOptions Class

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

For a list of all members of this type, see TMyConnectionOptions members.

Unit

MyAccess

Syntax

```
TMyConnectionOptions = class(TCustomMyConnectionOptions);
```

Inheritance Hierarchy

TDAConnectionOptions

   TCustomMyConnectionOptions

     TMyConnectionOptions

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.1 Members

TMyConnectionOptions class overview.

Properties

Name	Description
AllowImplicitConnect (inherited from **TDAConnectionOptions**)	Specifies whether to allow or not implicit connection opening.
Charset	Used to set the character set that MyDAC uses to read and write character data.
CheckBackslashes	Used to check the value of the NO_BACKSLASH_ESCAPES server variable.
Compress	Used to apply compression on transferring data.
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.
[Direct](#)	Used to work without using MySQL client library (libmysql.dll).
[DisconnectedMode](#)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
[Embedded](#)	Used to specify the server that will be used to connect.
[Interactive](#)	Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.
[IPVersion](#)	Used to specify Internet Protocol Version.
[KeepDesignConnected](#)	Used to prevent an application from establishing a connection at startup.
[LocalFailover](#)	If True, the [TCustomDAConnection.OnConnectionLost](#) event occurs and a failover operation can be performed after connection breaks.
[NullForZeroDelphiDate](#)	Used to hide the '30-12-1899' dates.
[NumericType](#)	Specifies the format of storing and representing NUMERIC (DECIMAL) fields.
[OptimizedBigInt](#)	Used to convert all fields with field length less than 11 of TLargeIntField type into

	TIntegerField.
[Protocol](#)	Used to specify the protocol to use when connecting to server.
[UseUnicode](#)	Used to specify whether the UTF8 charset will be used.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.8.2 Properties

Properties of the **TMyConnectionOptions** class.

For a complete list of the **TMyConnectionOptions** class members, see the

[TMyConnectionOptions Members](#) topic.

Published

Name	Description
[AllowImplicitConnect](#) (inherited from **[TDAConnectionOptions](#)**)	Specifies whether to allow or not implicit connection opening.
[Charset](#)	Used to set the character set that MyDAC uses to read and write character data.
[CheckBackslashes](#)	Used to check the value of the NO_BACKSLASH_ESCAP ES server variable.
[Compress](#)	Used to apply compression on transferring data.
[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.IndexFieldNa mes](#) property of a dataset.
[Direct](#)	Used to work without using MySQL client library (libmysql.dll).

[DisconnectedMode](#)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
[Embedded](#)	Used to specify the server that will be used to connect.
[Interactive](#)	Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.
[IPVersion](#)	Used to specify Internet Protocol Version.
[KeepDesignConnected](#)	Used to prevent an application from establishing a connection at startup.
[LocalFailover](#)	If True, the [TCustomDAConnection.OnConnectionLost](#) event occurs and a failover operation can be performed after connection breaks.
[NullForZeroDelphiDate](#)	Used to hide the '30-12-1899' dates.
[NumericType](#)	Specifies the format of storing and representing NUMERIC (DECIMAL) fields.
[OptimizedBigInt](#)	Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.
[Protocol](#)	Used to specify the protocol to use when connecting to server.
[UseUnicode](#)	Used to specify whether the UTF8 charset will be used.

See Also

- [TMyConnectionOptions Class](#)

- [TMyConnectionOptions Class Members](#)

   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.8.2.1 Charset Property

Used to set the character set that MyDAC uses to read and write character data.

Class

[TMyConnectionOptions](#)

Syntax

```
property Charset: string;
```

Remarks

Use the Charset property to set the character set that MyDAC uses to read and write character data.

   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.8.2.2 CheckBackslashes Property

Used to check the value of the NO_BACKSLASH_ESCAPES server variable.

Class

[TMyConnectionOptions](#)

Syntax

```
property CheckBackslashes: boolean default False;
```

Remarks

Use the CheckBackslashes property to check the value of the NO_BACKSLASH_ESCAPES server variable. It enables or disables the usage of the backslash character ('\') as an escape character within strings. The backslash character is used when transferring parameters in a

query and when dumping a database to a SQL script with the TMyDump component.

If the CheckBackslashes property is set to True, the value of the NO_BACKSLASH_ESCAPES variable is read from the server when establishing a connection. The received value will determine if the backslash character is used as an escape character within strings.

If the CheckBackslashes property is set to False, the backslash character will be used in any case. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.3 Compress Property

Used to apply compression on transferring data.

Class

TMyConnectionOptions

Syntax

```
property Compress: boolean default DefValCompress;
```

Remarks

Use the Compress property to use compression on transferring data. Setting this property to True is quite effective on transferring big volumes of data through slow connection. Pay attention that each row is compressed separately. Be careful when setting this option as in some cases it may decrease fetch speed instead of increasing. This property is ignored under CLR. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.4 DefaultSortType Property

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

Class

TMyConnectionOptions

Syntax

```
property DefaultSortType: TSortType;
```

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

Request Support | DAC Forum | Provide Feedback

5.14.1.8.2.5 Direct Property

Used to work without using MySQL client library (libmysql.dll).

Class

TMyConnectionOptions

Syntax

```
property Direct: boolean default DefValDirect;
```

Remarks

Use the Direct property to work without using MySQL client library (libmysql.dll). Used only if

Embedded is disabled.

Request Support | DAC Forum | Provide Feedback

5.14.1.8.2.6 DisconnectedMode Property

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

performing the operation.

Class

TMyConnectionOptions

Syntax

```
property DisconnectedMode: boolean;
```

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.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.7 Embedded Property

Used to specify the server that will be used to connect.

Class

TMyConnectionOptions

Syntax

```
property Embedded: boolean default DefValEmbedded;
```

Remarks

Use the Embedded property to specify what server will be used to connect - MySQL server or Embedded MySQL server. You can read about the features and using Embedded server at the Embedded Server. In most cases, you should use TMyEmbConnection instead of this option.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.8 Interactive Property

Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.

Class

TMyConnectionOptions

Syntax

```
property Interactive: boolean default DefValInteractive;
```

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.9 IPVersion Property

Used to specify Internet Protocol Version.

Class

TMyConnectionOptions

Syntax

```
property IPVersion: TIPVersion default DefValIPVersion;
```

Remarks

Use the IPVersion property to specify Internet Protocol Version.

Supported values:

- ivIPBoth - specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used;

- ivIPv4 (default) - specifies that Internet Protocol Version 4 (IPv4) will be used;

- ivIPv6 - specifies that Internet Protocol Version 6 (IPv6) will be used.

Note : When the TIPVersion property is set to ivIPBoth, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.

See Also

- TIPVersion

Request Support **DAC Forum** **Provide Feedback**

5.14.1.8.2.10 KeepDesignConnected Property

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

Class

TMyConnectionOptions

Syntax

```
property KeepDesignConnected: boolean;
```

Remarks

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

Request Support **DAC Forum** **Provide Feedback**

5.14.1.8.2.11 LocalFailover Property

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

Class

TMyConnectionOptions

Syntax

```
property LocalFailover: boolean;
```

Remarks

If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks. Read the Working in an Unstable Network topic for more information about using failover.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.12 NullForZeroDelphiDate Property

Used to hide the '30-12-1899' dates.

Class

TMyConnectionOptions

Syntax

```
property NullForZeroDelphiDate: boolean default False;
```

Remarks

Use the NullForZeroDelphiDate property to hide the '30-12-1899' dates. If NullForZeroDelphiDate is set to True, the values of all datetime fields will be changed to Null. If the property is set to False, the '30-12-1899' value will be used as an ordinary date. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.13 NumericType Property

Specifies the format of storing and representing NUMERIC (DECIMAL) fields.

Class

TMyConnectionOptions

Syntax

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

Remarks

Use the NumericType property to specify the format of storing and representing NUMERIC (DECIMAL) fields.

- 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 is slower.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.14 OptimizedBigInt Property

Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.

Class

TMyConnectionOptions

Syntax

```
property OptimizedBigInt: boolean default False;
```

Remarks

Setting this option converts all fields with field length less than 11 of TLargeIntField type into TIntegerField. This allows to process fields that are results of numeric function or cast values as usual Integer fields. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.8.2.15 Protocol Property

Used to specify the protocol to use when connecting to server.

Class

[TMyConnectionOptions](#)

Syntax

```
property Protocol: TMyProtocol default DefValProtocol;
```

Remarks

Use the Protocol property to specify which protocol to use when connecting to server. To use these constants you should add MyClasses unit to uses clause.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.8.2.16 UseUnicode Property

Used to specify whether the UTF8 charset will be used.

Class

[TMyConnectionOptions](#)

Syntax

```
property UseUnicode: boolean default DefValUseUnicode;
```

Remarks

Use the UseUnicode property to set MySQL client charset to UTF8 and convert client data according to this charset. When a value is assigned to this property, TMyConnection is closed.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.9 TMyConnectionSSLOptions Class

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

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

Unit

[MyAccess](#)

Syntax

```
TMyConnectionSSLOptions = class(TDAConnectionSSLOptions);
```

Inheritance Hierarchy

TDAConnectionSSLOptions

   TMyConnectionSSLOptions

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.9.1 Members

TMyConnectionSSLOptions class overview.

Properties

Name	Description
CACert (inherited from **TDAConnectionSSLOptions**)	Holds the path to the certificate authority file.
Cert (inherited from **TDAConnectionSSLOptions**)	Holds the path to the client certificate.
CipherList (inherited from **TDAConnectionSSLOptions**)	Holds the list of allowed SSL ciphers.
Key (inherited from **TDAConnectionSSLOptions**)	Holds the path to the private client key.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.10 TMyDataSetOptions Class

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

For a list of all members of this type, see TMyDataSetOptions members.

Unit

MyAccess

Syntax

```
TMyDataSetOptions = class(TDADataSetOptions);
```

Inheritance Hierarchy

[TDADataSetOptions](#)

  TMyDataSetOptions

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.10.1 Members

[TMyDataSetOptions](#) class overview.

Properties

Name	Description
[AutoPrepare](#)	Used to execute automatic [TCustomDADataSet.Prepare](#) on a query execution.
[AutoRefresh](#)	Used to automatically refresh dataset every AutoRefreshInterval seconds.
[AutoRefreshInterval](#)	Used to define in what time interval in seconds the Refresh or [TCustomMyDataSet.RefreshQuick](#) method of a DataSet is called.
[BinaryAsString](#)	Used to specify a method of representation of the BINARY and VARBINARY fields.
[CacheCalcFields](#) (inherited from **[TDADataSetOptions](#)**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
[CheckRowVersion](#)	Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.

CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
CreateConnection	Used to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.
DefaultValues	Used to fill the DefaultExpression property of TField objects with appropriate value.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
EnableBoolean	Used to specify the method of representation of the TINYINT(1) fields.
FieldsAsString	Used to store all non-BLOB fields as string (native MySQL format).
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 automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. Default value is false.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not

	refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NullForZeroDate	Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).
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 record count for this SELECT so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomMyDataSet to quote all field names in autogenerated SQL statements.
RemoveOnRefresh	Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.

[ReturnParams](#)	Used to return the new value of the fields to dataset after insert or update.
[SaveBlobsAsHex](#)	Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.
[SetEmptyStrToNull](#)	Force replace of empty strings with NULL values in data. The default value is False.
[SetFieldsReadOnly](#)	Used to specify whether fields not belonging to the current updating table get read-only attribute.
[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records does not equal 1.
[TrimFixedChar](#)	Used to specify whether to discard all trailing spaces in string fields of the dataset.
[TrimVarChar](#)	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
[UpdateAllFields](#) (inherited from **[TDADataSetOptions](#)**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
[UpdateBatchSize](#) (inherited from **[TDADataSetOptions](#)**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.10.2 Properties

Properties of the **TMyDataSetOptions** class.

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

Public

Name	Description
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.

MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
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 automatically refresh dataset every AutoRefreshInterval seconds.
AutoRefreshInterval	Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.
BinaryAsString	Used to specify a method of representation of the BINARY and VARBINARY fields.
CheckRowVersion	Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or

	delete data.
[CreateConnection](#)	Used to specify if an additional connection to a server should be established to execute an additional query in the [TCustomMyDataSet.FetchAll](#)=False mode.
[DefaultValues](#)	Used to fill the DefaultExpression property of TField objects with appropriate value.
[EnableBoolean](#)	Used to specify the method of representation of the TINYINT(1) fields.
[FieldsAsString](#)	Used to store all non-BLOB fields as string (native MySQL format).
[FullRefresh](#)	Used to specify the fields to include in automatically generated SQL statement when calling the [TCustomDADataSet.RefreshRecord](#) method. Default value is false.
[NullForZeroDate](#)	Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).
[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 record count for this SELECT so the RecordCount property reflects the actual number of records.
[QuoteNames](#)	Used for TCustomMyDataSet to

	quote all field names in autogenerated SQL statements.
[RemoveOnRefresh](#)	Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.
[RequiredFields](#)	Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.
[ReturnParams](#)	Used to return the new value of the fields to dataset after insert or update.
[SaveBlobsAsHex](#)	Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.
[SetEmptyStrToNull](#)	Force replace of empty strings with NULL values in data. The default value is False.
[SetFieldsReadOnly](#)	Used to specify whether fields not belonging to the current updating table get read-only attribute.
[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records does not equal 1.
[TrimFixedChar](#)	Used to specify whether to discard all trailing spaces in string fields of the dataset.
[TrimVarChar](#)	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

See Also

- TMyDataSetOptions Class

- TMyDataSetOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.1 AutoPrepare Property

Used to execute automatic TCustomDADataSet.Prepare on a query execution.

Class

TMyDataSetOptions

Syntax

```
property AutoPrepare: boolean;
```

Remarks

Set the AutoPrepare property to execute automatic TCustomDADataSet.Prepare on a query execution. Makes sense for cases when a query will be executed several times, for example, in Master/Detail relationships.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.2 AutoRefresh Property

Used to automatically refresh dataset every AutoRefreshInterval seconds.

Class

TMyDataSetOptions

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 TCustomMyDataSet.RefreshQuick

method will be executed, otherwise the Refresh method will be executed. This option is only available for Windows.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.3 AutoRefreshInterval Property

Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.

Class

TMyDataSetOptions

Syntax

```
property AutoRefreshInterval: integer default 60;
```

Remarks

Use the AutoRefreshInterval property to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called. This option is only available for Windows.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.4 BinaryAsString Property

Used to specify a method of representation of the BINARY and VARBINARY fields.

Class

TMyDataSetOptions

Syntax

```
property BinaryAsString: boolean default True;
```

Remarks

Use the BinaryAsString property to specify a method of representation of the BINARY and

VARBINARY fields. If set to True, these fields will be represented as string fields; otherwise, as TBytesField and TVarBytesField correspondingly. If the binary fields should not be processed as strings then set this property to False. The default value is True.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.5 CheckRow Version Property

Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.

Class

TMyDataSetOptions

Syntax

```
property CheckRowVersion: boolean default False;
```

Remarks

Use the CheckRowVersion property to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data. If CheckRowVersion is True and DataSet has timestamp field when only this field is added into WHERE clause of generated SQL statement. If CheckRowVersion is True, but there is no TIMESTAMP field, then all nonblob fields will be added to WHERE clause. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.6 CreateConnection Property

Used to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.

Class

TMyDataSetOptions

Syntax

```
property CreateConnection: boolean default True;
```

Remarks

Use the CreateConnection property to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode. If a DataSet is opened in TCustomMyDataSet.FetchAll=False, the current connection is blocked until all records have been fetched. If this option is set to True, an additional connection is created to fetch data to avoid blocking of the current connection.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.7 DefaultValues Property

Used to fill the DefaultExpression property of TField objects with appropriate value.

Class

TMyDataSetOptions

Syntax

```
property DefaultValues: boolean;
```

Remarks

If True, TCustomMyDataSet fills the DefaultExpression property of TField objects with appropriate value.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.8 EnableBoolean Property

Used to specify the method of representation of the TINYINT(1) fields.

Class

TMyDataSetOptions

Syntax

```
property EnableBoolean: boolean default True;
```

Remarks

Use the EnableBoolean property to specify the method of representation of the TINYINT(1) fields. If set to True, these fields will be represented as TBooleanField; otherwise, as TSmallintField. The default value is True.

Request Support DAC Forum Provide Feedback

5.14.1.10.2.9 FieldsAsString Property

Used to store all non-BLOB fields as string (native MySQL format).

Class

TMyDataSetOptions

Syntax

```
property FieldsAsString: boolean default False;
```

Remarks

All non-BLOB fields are stored as string (native MySQL format). The default value is False.

Request Support DAC Forum Provide Feedback

5.14.1.10.2.10 FullRefresh Property

Used to specify the fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. Default value is false.

Class

TMyDataSetOptions

Syntax

```
property FullRefresh: boolean;
```

Remarks

Use the FullRefresh property to specify what fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. If the FullRefresh property is True, all the fields from query are included into a SQL statement to refresh single record. If FullRefresh is False, only fields from TMyQuery.UpdatingTable are included. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.11 NullForZeroDate Property

Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).

Class

TMyDataSetOptions

Syntax

```
property NullForZeroDate: boolean default True;
```

Remarks

For datetime fields with invalid values, for example '2002-12-32', MySQL returns on fetch the '0000-00-00' value. According to the NullForZeroDate option this value will be represented as Null or '0001-01-01' ('0100-01-01' for CLR). The default value is True.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

TMyDataSetOptions

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.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.13 QueryRecCount Property

Used for TCustomDADataSet to perform additional query to get record count for this SELECT so the RecordCount property reflects the actual number of records.

Class

TMyDataSetOptions

Syntax

```
property QueryRecCount: boolean;
```

Remarks

If True, and the TCustomMyDataSet.FetchAll property is False, 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 TCustomMyDataSet.FetchAll property is True.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.14 QuoteNames Property

Used for TCustomMyDataSet to quote all field names in autogenerated SQL statements.

Class

TMyDataSetOptions

Syntax

```
property QuoteNames: boolean;
```

Remarks

If True, TCustomMyDataSet quotes all field names in autogenerated SQL statements such as update SQL.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.10.2.15 RemoveOnRefresh Property

Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.

Class

TMyDataSetOptions

Syntax

```
property RemoveOnRefresh: boolean;
```

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 its key value.

This option makes sense only if the StrictUpdate option is set to False. If the StrictUpdate option is True, error will be generated regardless of the RemoveOnRefresh option value.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.10.2.16 RequiredFields Property

Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.

Class

[TMyDataSetOptions](#)

Syntax

```
property RequiredFields: boolean default False;
```

Remarks

If True, TCustomDADataSet sets the Required property of TField objects for NOT NULL fields. It is useful when table has a trigger that updates NOT NULL fields. The default value is False.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.10.2.17 ReturnParams Property

Used to return the new value of the fields to dataset after insert or update.

Class

[TMyDataSetOptions](#)

Syntax

```
property ReturnParams: boolean;
```

Remarks

Use the ReturnParams property to return the new value of the fields to dataset after insert or update. Actual value of field after insert or update may be different from the value stored in local memory if the table has a trigger. When ReturnParams is True, OUT parameters of SQLInsert and SQLUpdate statements is assigned to corresponding fields. The default value is False.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.10.2.18 SaveBlobsAsHex Property

Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.

Class

TMyDataSetOptions

Syntax

```
property SaveBlobsAsHex: boolean default True;
```

Remarks

Use the SaveBlobsAsHex property to save BLOB data in hexadecimal format when calling the TMyQuery.SaveToXML method. If the SaveBlobsAsHex property is set to False, the value of the BLOB field is saved as a string. The default value is True.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.19 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. The default value is False.

Class

TMyDataSetOptions

Syntax

```
property SetEmptyStrToNull: boolean;
```

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.20 SetFieldsReadOnly Property

Used to specify whether fields not belonging to the current updating table get read-only attribute.

Class

TMyDataSetOptions

Syntax

```
property SetFieldsReadOnly: boolean default False;
```

Remarks

Use the SetFieldsReadOnly property to specify whether fields not belonging to the current updating table get read-only attribute. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.21 StrictUpdate Property

Used for TCustomDADataSet to raise an exception when the number of updated or deleted records does not equal 1.

Class

TMyDataSetOptions

Syntax

```
property StrictUpdate: boolean;
```

Remarks

TCustomDADataSet raises an exception when the number of updated or deleted records does not equal 1. Setting this option also causes an exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you use a non-SQL block. The default value is True.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.22 TrimFixedChar Property

Used to specify whether to discard all trailing spaces in string fields of the dataset.

Class

TMyDataSetOptions

Syntax

```
property TrimFixedChar: boolean;
```

Remarks

Use the TrimFixedChar property to specify whether to discard all trailing spaces in string fields of the dataset. The default value is True.

Request Support   DAC Forum   Provide Feedback

5.14.1.10.2.23 TrimVarChar Property

Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

Class

TMyDataSetOptions

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.

Request Support   DAC Forum   Provide Feedback

5.14.1.11 TMyDataSource Class

TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TMyDataSource members.

Unit

MyAccess

Syntax

```
TMyDataSource = class(TCRDataSource);
```

Remarks

TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a form.

TMyDataSource inherits its functionality directly from the TDataSource component.

At design-time assign individual data-aware components' DataSource properties from their drop-down listboxes.

Inheritance Hierarchy

[TCRDataSource](#)

   TMyDataSource

   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.11.1 Members

[**TMyDataSource**](#) class overview.

   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.12 TMyEncryptor Class

The class that performs encrypting and decrypting of data.

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

Unit

[MyAccess](#)

Syntax

```
TMyEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

[TCREncryptor](#)

   TMyEncryptor

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.12.1 Members

TMyEncryptor class overview.

Properties

Name	Description
DataHeader (inherited from **TCREncryptor**)	Specifies whether the additional information is stored with the encrypted data.
EncryptionAlgorithm (inherited from **TCREncryptor**)	Specifies the algorithm of data encryption.
HashAlgorithm (inherited from **TCREncryptor**)	Specifies the algorithm of generating hash data.
InvalidHashAction (inherited from **TCREncryptor**)	Specifies the action to perform on data fetching when hash data is invalid.
Password (inherited from **TCREncryptor**)	Used to set a password that is used to generate a key for encryption.

Methods

Name	Description
SetKey (inherited from **TCREncryptor**)	Sets a key, using which data is encrypted.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.13 TMyMetaData Class

A component for obtaining metainformation about database objects from the server.

For a list of all members of this type, see TMyMetaData members.

Unit

MyAccess

Syntax

```
TMyMetaData = class(TDAMetaData);
```

Remarks

The TMyMetaData component is used to obtain metainformation from the server about objects in the database, such as tables, table columns, stored procedures, etc. TMyMetaData publishes properties of TDAMetaData.

Inheritance Hierarchy

TMemDataSet

   TDAMetaData

      TMyMetaData

See Also

- TCustomDADataSet.Debug

- TCustomDASQL.Debug

- DBMonitor

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.14.1.13.1 Members

TMyMetaData class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Connection (inherited from **TDAMetaData**)	Used to specify a connection object to use to connect to a data store.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.

KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MetaDataKind (inherited from **TDAMetaData**)	Used to specify which kind of metainformation to show.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Restrictions (inherited from **TDAMetaData**)	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.

CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds (inherited from **TDAMetaData**)	Used to get values acceptable in the MetaDataKind property.
GetRestrictions (inherited from **TDAMetaData**)	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.

SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

Reserved.

5.14.1.14 TMyQuery 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 TMyQuery members.

Unit

MyAccess

Syntax

```
TMyQuery = class(TCustomMyDataSet);
```

Remarks

TMyQuery is a direct descendant of the TCustomMyDataSet component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.

Use TMyQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TMyQuery 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, TMyQuery will retrieve primary keys for UpdatingTable from metadata. TMyQuery 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

[TMemDataSet](#)

   [TCustomDADataSet](#)

      [TCustomMyDataSet](#)

        TMyQuery

See Also

- [Updating Data with MyDAC Dataset Components](#)

- [Master/Detail Relationships](#)

- [TMyStoredProc](#)

- [TMyTable](#)

| **Request Support** | **DAC Forum** | **Provide Feedback**

5.14.1.14.1 Members

[TMyQuery](#) 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.
[CommandTimeout](#) (inherited from **[TCustomMyDataSet](#)**)	Used to specify the amount of time to attempt execution of a command.
[Conditions](#) (inherited from **[TCustomDADataSet](#)**)	Used to add WHERE conditions to a query
[Connection](#) (inherited from **[TCustomMyDataSet](#)**)	Used to specify a connection object that will be used to connect to a data

	store.
[DataTypeMap](#) (inherited from **[TCustomDADataSet](#)**)	Used to set data type mapping rules
[Debug](#) (inherited from **[TCustomDADataSet](#)**)	Used to display the statement that is being executed and the values and types of its parameters.
[DetailFields](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
[Disconnected](#) (inherited from **[TCustomDADataSet](#)**)	Used to keep dataset opened after connection is closed.
[Encryption](#) (inherited from **[TCustomMyDataSet](#)**)	Used to specify encryption options in a dataset.
[FetchAll](#)	Defines whether to request all records of the query from database server when the dataset is being opened.
[FetchRows](#) (inherited from **[TCustomDADataSet](#)**)	Used to define the number of rows to be transferred across the network at the same time.
[FilterSQL](#) (inherited from **[TCustomDADataSet](#)**)	Used to change the WHERE clause of SELECT statement and reopen a query.
[FinalSQL](#) (inherited from **[TCustomDADataSet](#)**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
[IndexFieldNames](#) (inherited from **[TMemDataSet](#)**)	Used to get or set the list of fields on which the recordset is sorted.
[InsertId](#) (inherited from **[TCustomMyDataSet](#)**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
[IsQuery](#) (inherited from **[TCustomDADataSet](#)**)	Used to check whether SQL statement returns rows.
[KeyExclusive](#) (inherited from **[TMemDataSet](#)**)	Specifies the upper and

	lower boundaries for a range.
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 **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by

	calling the TCustomDADataSet.RefreshRecord procedure.
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 **TCustomMyDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.
UpdatingTable	Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.

FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.

SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the TCustomMyDataSet.LockTable method.
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 **TCustomMyDataSet**)	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 **TCustomMyDataSet**)	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.

© 1997-2025 Devart. All Rights Reserved.   **Request Support**   **DAC Forum**   **Provide Feedback**

5.14.1.14.2 Properties

Properties of the **TMyQuery** class.

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

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.

CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous

	query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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 **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL

	statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[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 **[TCustomMyDataSet](#)**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

Published

Name	Description
[FetchAll](#)	Defines whether to request all records of the query from database server when the dataset is being opened.
[LockMode](#)	Used to specify what kind of lock will be performed when editing a record.
[UpdatingTable](#)	Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

See Also

- TMyQuery Class

- TMyQuery Class Members

   Request Support   DAC Forum   Provide Feedback

5.14.1.14.2.1   FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TMyQuery

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

When the FetchAll property is False, the first call to TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

   Request Support   DAC Forum   Provide Feedback

5.14.1.14.2.2   LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TMyQuery

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
- TMyStoredProc.LockMode

- TMyTable.LockMode

Request Support   DAC Forum   Provide Feedback

5.14.1.14.2.3 UpdatingTable Property

Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Class

TMyQuery

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 TCustomMyDataSet.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
TCustomMyDataSet.Options)

Example

The first example specifies a query in which only one value 'Dept' for the UpdatingTable
property is allowed.

The second example shows a query in which allowed values for UpdatingTable are 'Dept' and
'Emp'.

By default the updating table will be the first used table, so 'DEPT' and all fields of DEPT will
be editable. If however UpdatingTable is set to be 'EMP' all fields of EMP will be editable.

```
Example 1.
    SELECT * FROM Dept
Example 2.
    SELECT * FROM Dept, Emp
        WHERE Dept.DeptNo = Emp.DeptNo
```

Request Support   DAC Forum   Provide Feedback

5.14.1.15 TMyStoredProc Class

A component for accessing and executing stored procedures and functions.

For a list of all members of this type, see TMyStoredProc members.

Unit

MyAccess

Syntax

```
TMyStoredProc = class(TCustomMyStoredProc);
```

Remarks

Use TMyStoredProc 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

TMemDataSet

   TCustomDADataSet

      TCustomMyDataSet

         TCustomMyStoredProc

            TMyStoredProc

See Also

- TMyQuery

- TMyCommand

- Updating Data with MyDAC Dataset Components

Request Support   DAC Forum   Provide Feedback

5.14.1.15.1 Members

TMyStoredProc 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.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be

	used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.

KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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 **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used

	to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[SQLUpdate](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
[StoredProcName](#) (inherited from **[TCustomMyStoredProc](#)**)	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 **[TCustomMyDataSet](#)**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.
[UpdatingTable](#)	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.
[ApplyRange](#) (inherited from **[TMemDataSet](#)**)	Applies a range to the dataset.

ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ExecProc (inherited from **TCustomMyStoredProc**)	Executes a SQL statement on the server.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.

Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL (inherited from **TCustomMyStoredProc**)	Builds a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assigns it to the SQL property.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as

	unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the TCustomMyDataSet.LockTable method.
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 **TCustomMyDataSet**)	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 **TCustomMyDataSet**)	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.

Request Support   DAC Forum   Provide Feedback

5.14.1.15.2 Properties

Properties of the **TMyStoredProc** class.

For a complete list of the **TMyStoredProc** class members, see the TMyStoredProc

Members topic.

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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 **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.

SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StoredProcName (inherited from **TCustomMyStoredProc**)	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 **TCustomMyDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Published

Name	Description
[LockMode](#)	Used to specify what kind of lock will be performed when editing a record.
[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

- [TMyStoredProc Class](#)

- [TMyStoredProc Class Members](#)

Request Support   DAC Forum   Provide Feedback

5.14.1.15.2.1 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

`TMyStoredProc`

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

- [TMyQuery.LockMode](#)

- [TMyTable.LockMode](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.15.2.2 UpdatingTable Property

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

[TMyStoredProc](#)

Syntax

```
property UpdatingTable: string;
```

Remarks

the UpdatingTable property is used on Insert, Update, Delete or RefreshRecord (see also [TCustomMyDataSet.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 the 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 [TCustomMyDataSet.Options](#) is True).

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.16 TMyTable 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 [TMyTable](#) members.

Unit

MyAccess

Syntax

```
TMyTable = class(TCustomMyTable);
```

Remarks

The TMyTable component allows retrieving and updating data in a single table without writing SQL statements. Use TMyTable to access data in a table . Use the TableName property to specify table name. TMyTable 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

TMemDataSet

   TCustomDADataSet

      TCustomMyDataSet

         TCustomMyTable

           TMyTable

See Also

- [Updating Data with MyDAC Dataset Components](#)

- [Master/Detail Relationships](#)

- [TCustomMyDataSet](#)

- [TMyQuery](#)

- [TCustomMyTable](#)

© 1997-2025 Devart. All Rights Reserved.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.14.1.16.1 Members

TMyTable 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.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.

FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexDefs (inherited from **TCustomMyTable**)	Contains information about the indexes for a table.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TCustomDADataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
Limit (inherited from **TCustomMyTable**)	Used to set the number of rows retrieved from the query.
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.
[Offset](inherited from **TCustomMyTable**)	Used to allow retrieving data from the server starting from the specified row.
[Options](inherited from **TCustomMyTable**)	Specifies the behaviour of the TMyTable 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
[Prepared](inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
[Ranged](inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
[ReadOnly](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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
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 **TCustomMyDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.

DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
EmptyTable (inherited from **TCustomMyTable**)	Deletes all records from the database table specified by the TableName property.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.

ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL (inherited from **TCustomMyTable**)	Determines KeyFields and build query of TCustomMyTable.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the

	dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the TCustomMyDataSet.LockTable method.
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 **TCustomMyDataSet**)	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 **TCustomMyDataSet**)	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.

Request Support   DAC Forum   Provide Feedback

5.14.1.16.2 Properties

Properties of the **TMyTable** class.

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

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomMyDataSet**)	Used to specify a connection object that will be used to connect to a data store.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and

		types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)		Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)		Used to keep dataset opened after connection is closed.
Encryption (inherited from **TCustomMyDataSet**)		Used to specify encryption options in a dataset.
FetchRows (inherited from **TCustomDADataSet**)		Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)		Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)		Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexDefs (inherited from **TCustomMyTable**)		Contains information about the indexes for a table.
IndexFieldNames (inherited from **TMemDataSet**)		Used to get or set the list of fields on which the recordset is sorted.
InsertId (inherited from **TCustomMyDataSet**)		Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)		Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)		Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TCustomDADataSet**)		Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.

Limit (inherited from **TCustomMyTable**)	Used to set the number of rows retrieved from the query.
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.
Offset (inherited from **TCustomMyTable**)	Used to allow retrieving data from the server starting from the specified row.
Options (inherited from **TCustomMyTable**)	Specifies the behaviour of the TMyTable 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query

	is prepared for execution or not.
[Ranged](#) (inherited from **[TMemDataSet](#)**)	Indicates whether a range is applied to a dataset.
[ReadOnly](#) (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.
[SmartFetch](#) (inherited from **[TCustomMyDataSet](#)**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
[SQL](#) (inherited from **[TCustomDADataSet](#)**)	Used to provide a SQL statement that a query component executes when its Open method is called.
[SQLDelete](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
[SQLInsert](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[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 **TCustomMyDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Published

Name	Description
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
LockMode	Used to specify what kind of lock will be performed when editing a record.
OrderFields	Used to build ORDER BY clause of SQL statements.
TableName	Used to specify the name of the database table this component encapsulates.

See Also

- **TMyTable Class**

- **TMyTable Class Members**

Request Support   DAC Forum   Provide Feedback

5.14.1.16.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

`TMyTable`

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

When the FetchAll property is False, the first call to TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

Request Support   DAC Forum   Provide Feedback

5.14.1.16.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

`TMyTable`

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

- TMyStoredProc.LockMode

- TMyQuery.LockMode

Request Support   DAC Forum   Provide Feedback

5.14.1.16.2.3 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

TMyTable

Syntax

```
property OrderFields: string;
```

Remarks

TMyTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas.

TMyTable is reopened when OrderFields is being changed.

See Also

- TMyTable

Request Support   DAC Forum   Provide Feedback

5.14.1.16.2.4 TableName Property

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

Class

TMyTable

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.

Request Support   DAC Forum   Provide Feedback

5.14.1.17 TMyTableOptions Class

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

For a list of all members of this type, see TMyTableOptions members.

Unit

MyAccess

Syntax

```
TMyTableOptions = class(TMyDataSetOptions);
```

Inheritance Hierarchy

TDADataSetOptions

  TMyDataSetOptions

    TMyTableOptions

Request Support   DAC Forum   Provide Feedback

5.14.1.17.1   Members

TMyTableOptions class overview.

Properties

Name	Description
AutoPrepare (inherited from **TMyDataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on a query execution.
AutoRefresh (inherited from **TMyDataSetOptions**)	Used to automatically refresh dataset every AutoRefreshInterval seconds.
AutoRefreshInterval (inherited from **TMyDataSetOptions**)	Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.
BinaryAsString (inherited from **TMyDataSetOptions**)	Used to specify a method of representation of the BINARY and VARBINARY fields.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CheckRowVersion (inherited from **TMyDataSetOptions**)	Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
CreateConnection (inherited from **TMyDataSetOptions**)	Used to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.

DefaultValues (inherited from **TMyDataSetOptions**)	Used to fill the DefaultExpression property of TField objects with appropriate value.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
EnableBoolean (inherited from **TMyDataSetOptions**)	Used to specify the method of representation of the TINYINT(1) fields.
FieldsAsString (inherited from **TMyDataSetOptions**)	Used to store all non-BLOB fields as string (native MySQL format).
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
FullRefresh (inherited from **TMyDataSetOptions**)	Used to specify the fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. Default value is false.
HandlerIndex	Used to assign an index and a value that this index should satisfy.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from	Allows to use NULL values in the fields by which the

TDADataSetOptions)	relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NullForZeroDate (inherited from **TMyDataSetOptions**)	Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).
NumberRange (inherited from **TMyDataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount (inherited from **TMyDataSetOptions**)	Used for TCustomDADataSet to perform additional query to get record count for this SELECT so the RecordCount property reflects the actual number of records.
QuoteNames (inherited from **TMyDataSetOptions**)	Used for TCustomMyDataSet to quote all field names in autogenerated SQL statements.
RemoveOnRefresh (inherited from **TMyDataSetOptions**)	Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.
RequiredFields (inherited from **TMyDataSetOptions**)	Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.
ReturnParams (inherited from **TMyDataSetOptions**)	Used to return the new value of the fields to dataset after insert or update.
SaveBlobsAsHex (inherited from **TMyDataSetOptions**)	Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML

	method.
SetEmptyStrToNull (inherited from **TMyDataSetOptions**)	Force replace of empty strings with NULL values in data. The default value is False.
SetFieldsReadOnly (inherited from **TMyDataSetOptions**)	Used to specify whether fields not belonging to the current updating table get read-only attribute.
StrictUpdate (inherited from **TMyDataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records does not equal 1.
TrimFixedChar (inherited from **TMyDataSetOptions**)	Used to specify whether to discard all trailing spaces in string fields of the dataset.
TrimVarChar (inherited from **TMyDataSetOptions**)	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
UpdateAllFields (inherited from **TDADataSetOptions**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from **TDADataSetOptions**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UseHandler	Used for the HANDLER statement to be used instead of the SELECT statement.

Request Support   DAC Forum   Provide Feedback

5.14.1.17.2 Properties

Properties of the **TMyTableOptions** class.

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

Public

Name	Description
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
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 (inherited from **TMyDataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on a query execution.
AutoRefresh (inherited from **TMyDataSetOptions**)	Used to automatically refresh dataset every AutoRefreshInterval seconds.
AutoRefreshInterval (inherited from **TMyDataSetOptions**)	Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.
BinaryAsString (inherited from **TMyDataSetOptions**)	Used to specify a method of representation of the BINARY and VARBINARY fields.
CheckRowVersion (inherited from **TMyDataSetOptions**)	Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.
CreateConnection (inherited from **TMyDataSetOptions**)	Used to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.

DefaultValues (inherited from **TMyDataSetOptions**)	Used to fill the DefaultExpression property of TField objects with appropriate value.
EnableBoolean (inherited from **TMyDataSetOptions**)	Used to specify the method of representation of the TINYINT(1) fields.
FieldsAsString (inherited from **TMyDataSetOptions**)	Used to store all non-BLOB fields as string (native MySQL format).
FullRefresh (inherited from **TMyDataSetOptions**)	Used to specify the fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. Default value is false.
HandlerIndex	Used to assign an index and a value that this index should satisfy.
NullForZeroDate (inherited from **TMyDataSetOptions**)	Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).
NumberRange (inherited from **TMyDataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount (inherited from **TMyDataSetOptions**)	Used for TCustomDADataSet to perform additional query to get record count for this SELECT so the RecordCount property reflects the actual number of records.
QuoteNames (inherited from **TMyDataSetOptions**)	Used for TCustomMyDataSet to quote all field names in autogenerated SQL statements.
RemoveOnRefresh (inherited from **TMyDataSetOptions**)	Used for dataset to remove a record locally if the

	RefreshRecord procedure can't find necessary record on the server.
RequiredFields (inherited from **TMyDataSetOptions**)	Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.
ReturnParams (inherited from **TMyDataSetOptions**)	Used to return the new value of the fields to dataset after insert or update.
SaveBlobsAsHex (inherited from **TMyDataSetOptions**)	Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.
SetEmptyStrToNull (inherited from **TMyDataSetOptions**)	Force replace of empty strings with NULL values in data. The default value is False.
SetFieldsReadOnly (inherited from **TMyDataSetOptions**)	Used to specify whether fields not belonging to the current updating table get read-only attribute.
StrictUpdate (inherited from **TMyDataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records does not equal 1.
TrimFixedChar (inherited from **TMyDataSetOptions**)	Used to specify whether to discard all trailing spaces in string fields of the dataset.
TrimVarChar (inherited from **TMyDataSetOptions**)	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
UseHandler	Used for the HANDLER statement to be used instead of the SELECT statement.

See Also

- TMyTableOptions Class

- TMyTableOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.14.1.17.2.1 HandlerIndex Property

Used to assign an index and a value that this index should satisfy.

Class

`TMyTableOptions`

Syntax

```
property HandlerIndex: string;
```

Remarks

Use the HandlerIndex property to assign an index and a value that this index should satisfy.

Request Support   DAC Forum   Provide Feedback

5.14.1.17.2.2 UseHandler Property

Used for the HANDLER statement to be used instead of the SELECT statement.

Class

`TMyTableOptions`

Syntax

```
property UseHandler: boolean default False;
```

Remarks

If this option is enabled, the HANDLER statement is used instead of the SELECT statement.

From the MySQL Refernce Manual:

"There are several reasons to use the HANDLER interface instead of normal SELECT statements:

HANDLER is faster than SELECT:

A designated storage engine handler object is allocated for the HANDLER ... OPEN. The object is reused for subsequent HANDLER statements for that table; it need not be reinitialized for each one.

There is less parsing involved.

There is no optimizer or query-checking overhead.

The table does not have to be locked between two handler requests.

The handler interface does not have to provide a consistent look of the data (for example, dirty reads are allowed), so the storage engine can use optimizations that SELECT does not normally allow.

For applications that use a low-level ISAM-like interface, HANDLER makes it much easier to port them to MySQL.

HANDLER enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with SELECT. The HANDLER interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database."

The FilterSQL property is used to assign the WHERE condition. To return a specific number of rows, assign the Limit property.

If the Limit property equals -1 and the UseHandler option is set to True, requested records count equals to MaxInt.

The default value of the UseHandler option is False.

Request Support   DAC Forum   Provide Feedback

5.14.1.18 TMyTransaction Class

A component for managing transactions.

For a list of all members of this type, see TMyTransaction members.

Unit

MyAccess

Syntax

```
TMyTransaction = class(TDATransaction);
```

Remarks

The TMyTransaction component is used to manage transactions in your application.

Inheritance Hierarchy

TDATransaction

  TMyTransaction

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.14.1.18.1 Members

TMyTransaction class overview.

Properties

Name	Description
Active (inherited from **TDATransaction**)	Used to determine if the transaction is active.
DefaultCloseAction (inherited from **TDATransaction**)	Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Methods

Name	Description
Commit (inherited from **TDATransaction**)	Commits the current transaction.
Rollback (inherited from **TDATransaction**)	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction (inherited from **TDATransaction**)	Begins a new transaction.

Events

Name	Description
OnCommit (inherited from **TDATransaction**)	Occurs after the transaction has been successfully committed.
OnCommitRetaining (inherited from **TDATransaction**)	Occurs after CommitRetaining has been executed.
OnError (inherited from **TDATransaction**)	Used to process errors that occur during executing a transaction.
OnRollback (inherited from **TDATransaction**)	Occurs after the transaction has been successfully rolled back.
OnRollbackRetaining (inherited from **TDATransaction**)	Occurs after RollbackRetaining has been executed.

Request Support   DAC Forum   Provide Feedback

5.14.1.19 TMyUpdateSQL Class

A component for tuning update operations for the DataSet component.

For a list of all members of this type, see TMyUpdateSQL members.

Unit

MyAccess

Syntax

```
TMyUpdateSQL = class(TCustomDAUpdateSQL);
```

Remarks

Use the TMyUpdateSQL component to provide DML statements for the dataset components that return read-only result set. This component also allows setting objects that can be used for executing update operations. You may prefer to use directly SQLInsert, SQLUpdate, and SQLDelete properties of the TCustomDADataSet descendants.

Inheritance Hierarchy

TCustomDAUpdateSQL

   TMyUpdateSQL

Request Support   **DAC Forum**   **Provide Feedback**

5.14.1.19.1 Members

TMyUpdateSQL 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.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.2 Types

Types in the **MyAccess** unit.

Types

Name	Description
[TMyUpdateExecuteEvent](#)	This type is used for the [TCustomMyDataSet.AfterUpdateExecute](#) and [TCustomMyDataSet.BeforeUpdateExecute](#) events.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.2.1 TMyUpdateExecuteEvent Procedure Reference

This type is used for the [TCustomMyDataSet.AfterUpdateExecute](#) and [TCustomMyDataSet.BeforeUpdateExecute](#) events.

Unit

MyAccess

Syntax

```
TMyUpdateExecuteEvent = procedure (Sender: TCustomMyDataSet;
StatementTypes: TStatementTypes; Params: TDAParams) of object;
```

Parameters

Sender
   An object that raised the event.

StatementTypes
   Holds the type of the SQL statement being executed.

Params
   Holds the parameters with which the SQL statement will be executed.

Request Support   DAC Forum   Provide Feedback

5.14.3 Enumerations

Enumerations in the **MyAccess** unit.

Enumerations

Name	Description
TLockRecordType	Specifies the type of the record locking.
TLockType	Specifies the type of the table locking.
TMyIsolationLevel	Specifies the xtent to which all outside transactions interfere with subsequent transactions of current connection.

Request Support   DAC Forum   Provide Feedback

5.14.3.1 TLockRecordType Enumeration

Specifies the type of the record locking.

Unit

MyAccess

Syntax

```
TLockRecordType = (lrImmediately, lrDelayed);
```

Values

Value	Meaning
lrDelayed	Locking is performed just at the time of execution Post for this record.
lrImmediately	Checks for locking directly on calling the Lock method. In this case if set Lock(lrImmediately) call to BeforeEdit event you can unable the same row in the table to be modified by several users at the same time.

Request Support   DAC Forum   Provide Feedback

5.14.3.2 TLockType Enumeration

Specifies the type of the table locking.

Unit

MyAccess

Syntax

```
TLockType = (ltRead, ltReadLocal, ltWrite, ltWriteLowPriority);
```

Values

Value	Meaning
ltRead	All connections including the current one can read only from the specified table.
ltReadLocal	Similar to ltRead. Moreover, allows non-conflicting INSERT statements to execute while the lock is held.
ltWrite	The current connection can read and write to the table. Other connections wait for the UnlockTable call.
ltWriteLowPriority	You can use ltWriteLowPriority locks to allow other threads to obtain ltRead locks while the thread is waiting for the ltWrite lock. You should use only ltWriteLowPriority locks if you are sure that eventually there will be a time when no threads will have a ltRead

	lock.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.3.3 TMyIsolationLevel Enumeration

Specifies the xtent to which all outside transactions interfere with subsequent transactions of current connection.

Unit

MyAccess

Syntax

```
TMyIsolationLevel = (ilReadCommitted, ilReadUnCommitted,
ilRepeatableRead, ilSerializable);
```

Values

Value	Meaning
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 isolation level. The default value.
ilReadUnCommitted	The most unrestricted level of 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 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).
ilSerializable	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 concurrent accesses to the same set of data, but may lead to a greater latency at times of congested database environment.

Request Support   **DAC Forum**   **Provide Feedback**

5.14.4 Routines

Routines in the **MyAccess** unit.

Routines

Name	Description
GetServerList	Returns the list of the MySQL servers in LAN. MySQL server does not provide usual ways of such list getting, so it can be incomplete.

Request Support DAC Forum Provide Feedback

5.14.4.1 GetServerList Procedure

Returns the list of the MySQL servers in LAN. MySQL server does not provide usual ways of such list getting, so it can be incomplete.

Unit

MyAccess

Syntax

```
procedure GetServerList(List: TStrings);
```

Parameters

List
  the list of the MySQL servers in LAN.

Request Support DAC Forum Provide Feedback

5.14.5 Constants

Constants in the **MyAccess** unit.

Constants

Name	Description

| MyDACVersion | Read this constant to get current version number for MyDAC. |

Request Support DAC Forum Provide Feedback

5.14.5.1 MyDACVersion Constant

Read this constant to get current version number for MyDAC.

Unit

MyAccess

Syntax

```
MyDACVersion = '12.3.2';
```

Request Support DAC Forum Provide Feedback

5.15 MyBackup

This unit contains implementation of the TMyBackup component.

Classes

Name	Description
TMyBackup	A component that serves for backup copying specified tables on the server.

Types

Name	Description
TMyTableMsgEvent	This type is used for the TMyBackup.OnTableMsg event.

Enumerations

Name	Description
[TMyBackupMode](#)	Specifies the mode of TMyBackup work.
[TMyBackupPriority](#)	Specifies priority of the [TMyBackup.Restore](#) operation.
[TMyRestoreDuplicates](#)	Specifies the behaviour on detection records with repeated key fields on the execution of the [TMyBackup.Restore](#) method.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1 Classes

Classes in the **MyBackup** unit.

Classes

Name	Description
[TMyBackup](#)	A component that serves for backup copying specified tables on the server.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1 TMyBackup Class

A component that serves for backup copying specified tables on the server.

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

Unit

[MyBackup](#)

Syntax

```
TMyBackup = class(TComponent);
```

Remarks

Serves for backup copying specified tables on the server. Supports working in two modes defined by the TMyBackup.Mode property - bmText and bmBinary.

The list of tables is specified in TMyBackup.TableNames.

Use the TMyBackup.Path property to specify the path to the server.

TMyBackup works on the server side that greatly affects on performance. However, it has some restrictions. Firstly, backup is performed only for the tables, database structure and user rights that are not stored. Secondly, files created on the server cannot be modified and deleted by MySQL tools.

Note: When using bmBinary mode, both servers must have the same format of the tables (must be the same version). About compatibility of formats storing data please see MySQL Reference Manual.

See Also
- TMyBackup.Backup

- TMyBackup.Restore

- TMyDump

Request Support   DAC Forum   Provide Feedback

5.15.1.1.1 Members

TMyBackup class overview.

Properties

Name	Description
BackupPriority	Specifies priority of the TMyBackup.Restore operation.
Connection	Used to specify a connection object that will be used to connect to a data store.

Debug	Used to display the statement being executed.
Duplicates	Used to specify the behaviour on detection records with repeated key fields on the execution of the TMyBackup.Restore method.
EnclosedBy	Used to specify a string (a character set) with which a field can be quoted.
EscapedBy	Used to specify a symbol that will be used as a special symbol pointer.
Fields	Holds the list of the fields that will be saved.
FieldsTerminatedBy	Used to specify field delimiter.
IgnoreLines	Determines if the specified rows number at the beginning of the file will be ignored or not.
LinesTerminatedBy	Used to specify row delimiter.
Local	Used to specify that TMyBackup.Path is a local path but not a path on the server.
Mode	Used to control modes of TMyBackup work.
Path	Holds a path on the server to the folder where data files will be stored.
TableNames	Holds the list of tables which will be used in the script.

Methods

Name	Description
Backup	Copies current tables by the specified path.
Restore	Restores tables.

Events

Name	Description
[OnTableMsg](#)	Occurrs on executing of the [TMyBackup.Restore](#) operation.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1.2 Properties

Properties of the **TMyBackup** class.

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

Published

Name	Description
[BackupPriority](#)	Specifies priority of the [TMyBackup.Restore](#) operation.
[Connection](#)	Used to specify a connection object that will be used to connect to a data store.
[Debug](#)	Used to display the statement being executed.
[Duplicates](#)	Used to specify the behaviour on detection records with repeated key fields on the execution of the [TMyBackup.Restore](#) method.
[EnclosedBy](#)	Used to specify a string (a character set) with which a field can be quoted.
[EscapedBy](#)	Used to specify a symbol that will be used as a special symbol pointer.
[Fields](#)	Holds the list of the fields that will be saved.
[FieldsTerminatedBy](#)	Used to specify field

	delimiter.
[IgnoreLines](#)	Determines if the specified rows number at the beginning of the file will be ignored or not.
[LinesTerminatedBy](#)	Used to specify row delimiter.
[Local](#)	Used to specify that [TMyBackup.Path](#) is a local path but not a path on the server.
[Mode](#)	Used to control modes of TMyBackup work.
[Path](#)	Holds a path on the server to the folder where data files will be stored.
[TableNames](#)	Holds the list of tables which will be used in the script.

See Also

- [TMyBackup Class](#)

- [TMyBackup Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1.2.1 BackupPriority Property

Specifies priority of the [Restore](#) operation.

Class

`TMyBackup`

Syntax

```
property BackupPriority: TMyBackupPriority default bpDefault;
```

Remarks

Specifies a priority on [Restore](#) operation.

Used only if [Mode](#) is bmText.

See Also

- Mode

- Restore

Request Support   DAC Forum   Provide Feedback

5.15.1.1.2.2 Connection Property

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

Class

TMyBackup

Syntax

```
property Connection: TCustomMyConnection;
```

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 TCustomMyConnection objects.

At run-time, set the Connection property to reference an existing TCustomMyConnection object.

See Also

- TCustomMyConnection

Request Support   DAC Forum   Provide Feedback

5.15.1.1.2.3 Debug Property

Used to display the statement being executed.

Class

TMyBackup

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement being executed.

The default value is False.

Note: To enable debug form display you should explicitly include MyDacVcl unit in your project.

See Also

- TCustomDASQL.Debug

- TCustomDADataSet.Debug

Request Support   DAC Forum   Provide Feedback

5.15.1.1.2.4 Duplicates Property

Used to specify the behaviour on detection records with repeated key fields on the execution of the Restore method.

Class

TMyBackup

Syntax

```
property Duplicates: TMyRestoreDuplicates default bdError;
```

Remarks

Use the Duplicates property to specify the behaviour on detection records with repeated key fields on the execution of the Restore method.

Used only if Mode is bmText.

See Also

- Mode

- [Restore](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1.2.5 EnclosedBy Property

Used to specify a string (a character set) with which a field can be quoted.

Class

[TMyBackup](#)

Syntax

```
property EnclosedBy: string;
```

Remarks

Use the EnclosedBy property to specify a string (a character set) with which a field can be quoted. By default it is an empty string.

Used only if [Mode](#) is bmText.

See Also

- [Backup](#)

- [Mode](#)

- [EscapedBy](#)

- [FieldsTerminatedBy](#)

- [LinesTerminatedBy](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1.2.6 EscapedBy Property

Used to specify a symbol that will be used as a special symbol pointer.

Class

[TMyBackup](#)

Syntax

```
property EscapedBy: string;
```

Remarks

Use the EscapedBy property to specify a symbol that will be used as a special symbol pointer to define the entering of a special symbol. By default such pointer is '\\'.

Used only if [Mode](#) is bmText.

See Also

- [Backup](#)

- [Mode](#)

- [EnclosedBy](#)

- [FieldsTerminatedBy](#)

- [LinesTerminatedBy](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1.2.7 Fields Property

Holds the list of the fields that will be saved.

Class

[TMyBackup](#)

Syntax

```
property Fields: string;
```

Remarks

The Fields property is used to contain the list of the fields that will be saved. If not specified, all fields will be stored.

Use this property on execution [Backup](#) for a single table (see [TableNames](#)).

Used only if [Mode](#) is bmText.

See Also

- Mode

- TableNames

- Backup

- Restore

Request Support   DAC Forum   Provide Feedback

5.15.1.1.2.8 FieldsTerminatedBy Property

Used to specify field delimiter.

Class

TMyBackup

Syntax

```
property FieldsTerminatedBy: string;
```

Remarks

Use the FieldsTerminatedBy property to specify a string that will be used as field delimiter. By default field delimiter is '\t'.

Used only if Mode is bmText.

See Also

- Backup

- Mode

- EnclosedBy

- EscapedBy

- LinesTerminatedBy

Request Support   DAC Forum   Provide Feedback

5.15.1.1.2.9 IgnoreLines Property

Determines if the specified rows number at the beginning of the file will be ignored or not.

Class

[TMyBackup](#)

Syntax

```
property IgnoreLines: integer default 0;
```

Remarks

Use the IgnoreLines property to determine if the specified rows number at the beginning of the file will be ignored or not.

Use this property if you need:

• create non-typical formatted text file to be processed by the third-party applications

• fast load of big volume data to the database generated by the third-party application.

Used only if [Mode](#) is bmText.

See Also

• [Mode](#)

• [Restore](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1.2.10 LinesTerminatedBy Property

Used to specify row delimiter.

Class

[TMyBackup](#)

Syntax

```
property LinesTerminatedBy: string;
```

Remarks

Use the LinesTerminatedBy property to specify a string that will be used as row delimiter. By default row delimiter is '\n'.

Used only if [Mode](#) is bmText.

See Also

- [Backup](#)

- [Mode](#)

- [EnclosedBy](#)

- [EscapedBy](#)

- [FieldsTerminatedBy](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.15.1.1.2.11 Local Property

Used to specify that [Path](#) is a local path but not a path on the server.

Class

[TMyBackup](#)

Syntax

```
property Local: boolean default False;
```

Remarks

Use the Local property to specify that [Path](#) is a local path but not a path on the server on the execution of the [Restore](#) method.

Used only if [Mode](#) is bmText.

See Also

- [Mode](#)

- [Restore](#)

5.15.1.1.2.12 Mode Property

Used to control modes of TMyBackup work.

Class

[TMyBackup](#)

Syntax

```
property Mode: TMyBackupMode default bmBinary;
```

Remarks

Use the Mode property to control modes of TMyBackup work.

Note: By security reasons MySQL does not allow to overwrite files that already exist. Also MySQL requires a path to the server to be created beforehand.

When using bmBinary mode, both servers must have the same format of the tables (must be the same version). About the compatibility of formats storing data please see MySQL Reference Manual.

See Also
- [Backup](#)

- [Restore](#)

5.15.1.1.2.13 Path Property

Holds a path on the server to the folder where data files will be stored.

Class

[TMyBackup](#)

Syntax

```
property Path: string;
```

Remarks

Use the Path property to contain a path on the server to the folder where data files will be stored. Path must exist before executing the Backup and Restore operations.

See Also

- Backup

- Restore

Request Support   DAC Forum   Provide Feedback

5.15.1.1.2.14 TableNames Property

Holds the list of tables which will be used in the script.

Class

TMyBackup

Syntax

```
property TableNames: string;
```

Remarks

Use the TableNames property to hold the list of tables which will be used in the script. Table names are separated by comma or semicolon. If it has an empty value, all the tables presented in the database will be processed.

See Also

- Backup

- Restore

- TCustomDAConnection.GetTableNames

Request Support   DAC Forum   Provide Feedback

5.15.1.1.3 Methods

Methods of the **TMyBackup** class.

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

Public

Name	Description
Backup	Copies current tables by the specified path.
Restore	Restores tables.

See Also

- TMyBackup Class

- TMyBackup Class Members

Request Support   DAC Forum   Provide Feedback

5.15.1.1.3.1 Backup Method

Copies current tables by the specified path.

Class

TMyBackup

Syntax

```
procedure Backup;
```

Remarks

Call the Backup method to copy current tables by the specified path (see Path property).

A list of tables is specified in the TableNames property. If the list isn't specified, all tables will be copied.

Use the Mode property to specify a copy mode.

If Mode is bmBinary before backup copying of table files, tables are

[TCustomMyDataSet.LockTable](#) on writing and [TMyServerControl.Flush](#) is called.

See Also

- [Mode](#)

- [TableNames](#)

- [Path](#)

| [Request Support](#) | [DAC Forum](#) | [Provide Feedback](#)

5.15.1.1.3.2 Restore Method

Restores tables.

Class

[TMyBackup](#)

Syntax

```
procedure Restore;
```

Remarks

Call the Restore method to restore tables.

See Also

- [OnTableMsg](#)

| [Request Support](#) | [DAC Forum](#) | [Provide Feedback](#)

5.15.1.1.4 Events

Events of the **TMyBackup** class.

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

Published

Name	Description

OnTableMsg	Occurrs on executing of the TMyBackup.Restore operation.

See Also

- TMyBackup Class

- TMyBackup Class Members

   Request Support   **DAC Forum**   **Provide Feedback**

5.15.1.1.4.1 OnTableMsg Event

Occurrs on executing of the Restore operation.

Class

TMyBackup

Syntax

```
property OnTableMsg: TMyTableMsgEvent;
```

Remarks

The OnTableMsg event occurs on executing of the Restore operation.

See Also

- Restore

   Request Support   **DAC Forum**   **Provide Feedback**

5.15.2 Types

Types in the **MyBackup** unit.

Types

Name	Description

TMyTableMsgEvent	This type is used for the TMyBackup.OnTableMsg event.

Request Support   DAC Forum   Provide Feedback

5.15.2.1 TMyTableMsgEvent Procedure Reference

This type is used for the TMyBackup.OnTableMsg event.

Unit

MyBackup

Syntax

```
TMyTableMsgEvent = procedure (Sender: TObject; TableName: string;
MsgText: string) of object;
```

Parameters

Sender
  An object that raised the event.

TableName
  Holds the table name.

MsgText
  Holds the message text.

Request Support   DAC Forum   Provide Feedback

5.15.3 Enumerations

Enumerations in the **MyBackup** unit.

Enumerations

Name	Description
TMyBackupMode	Specifies the mode of TMyBackup work.
TMyBackupPriority	Specifies priority of the TMyBackup.Restore operation.

[TMyRestoreDuplicates](#)	Specifies the behaviour on detection records with repeated key fields on the execution of the [TMyBackup.Restore](#) method.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.15.3.1 TMyBackupMode Enumeration

Specifies the mode of TMyBackup work.

Unit

[MyBackup](#)

Syntax

```
TMyBackupMode = (bmBinary, bmText);
```

Values

Value	Meaning
bmBinary	The fastest way of saving data. Files with the tables are copied physically to the specified folder (Path). This mode is not supported for several table types, for example, InnoDB.
bmText	Data from the tables is copied as a text file. To specify its format use [TMyBackup.EnclosedBy](#), [TMyBackup.EscapedBy](#), [TMyBackup.FieldsTerminatedBy](#), [TMyBackup.LinesTerminatedBy](#) properties. This mode can be used for all table types.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.15.3.2 TMyBackupPriority Enumeration

Specifies priority of the [TMyBackup.Restore](#) operation.

Unit

[MyBackup](#)

Syntax

```
TMyBackupPriority = (bpDefault, bpLowPriority, bpConcurrent);
```

Values

Value	Meaning
bpConcurrent	Restore will be executed simultaneously with queries from other connections.
bpDefault	Other connections wait for finishing Restore.
bpLowPriority	Execution of Restore will be suspended until other connections stop reading from the table.

Request Support   DAC Forum   Provide Feedback

5.15.3.3 TMyRestoreDuplicates Enumeration

Specifies the behaviour on detection records with repeated key fields on the execution of the TMyBackup.Restore method.

Unit

MyBackup

Syntax

```
TMyRestoreDuplicates = (bdIgnore, bdReplace, bdError);
```

Values

Value	Meaning
bdError	Generate an error, ignore the rest of the file.
bdIgnore	Ignore such records.
bdReplace	Replace old record with a new one.

Request Support   DAC Forum   Provide Feedback

5.16 MyBuilderClient

This unit contains implementation of the MyBuilder class.

Classes

Name	Description
TMyBuilder	Serves to manage SQL Builder for MySQL Add-in.

5.16.1 Classes

Classes in the **MyBuilderClient** unit.

Classes

Name	Description
TMyBuilder	Serves to manage SQL Builder for MySQL Add-in.

5.16.1.1 TMyBuilder Class

Serves to manage SQL Builder for MySQL Add-in.

For a list of all members of this type, see TMyBuilder members.

Unit

MyBuilderClient

Syntax

```
TMyBuilder = class(TComponent);
```

5.16.1.1.1 Members

TMyBuilder class overview.

Request Support   DAC Forum   Provide Feedback

5.17 MyClasses

This unit contains implementation of the EMyError class.

Classes

Name	Description
EMyError	A base class that is raised when MySQl server returns error as a result.

Enumerations

Name	Description
TMyProtocol	Specifies which protocol to use when connecting to server.

Variables

Name	Description
__Strings65535ToMemo	Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields.

Request Support   DAC Forum   Provide Feedback

5.17.1 Classes

Classes in the **MyClasses** unit.

Classes

Name	Description
EMyError	A base class that is raised when MySQl server returns error as a result.

Request Support   **DAC Forum**   **Provide Feedback**

5.17.1.1 EMyError Class

A base class that is raised when MySQl server returns error as a result.

For a list of all members of this type, see EMyError members.

Unit

MyClasses

Syntax

```
EMyError = class(EDAError);
```

Remarks

EMyError is raised when MySQL Server returns error as a result, for example, of an attempt to execute invalid SQL statement. Use EMyError in an exception-handling block.

Inheritance Hierarchy

EDAError
  EMyError

See Also

- EDAError

Request Support   **DAC Forum**   **Provide Feedback**

5.17.1.1.1 Members

EMyError 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.
[LineNumber](#)	Contains the number of a query line that caused an error.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.17.1.1.2 Properties

Properties of the **EMyError** class.

For a complete list of the **EMyError** class members, see the [EMyError 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.
[LineNumber](#)	Contains the number of a query line that caused an error.

See Also

- [EMyError Class](#)

- [EMyError Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.17.1.1.2.1 LineNumber Property

Contains the number of a query line that caused an error.

Class

EMyError

Syntax

```
property LineNumber: integer;
```

Remarks

If an error, having EDAError.ErrorCode = ER_PARSE_ERROR, occured during query execution, LineNumber property contains the number of a query line that caused an error. MyDAC will retrieve this information from the error text automatically.

Request Support   DAC Forum   Provide Feedback

5.17.2 Enumerations

Enumerations in the **MyClasses** unit.

Enumerations

Name	Description
TMyProtocol	Specifies which protocol to use when connecting to server.

Request Support   DAC Forum   Provide Feedback

5.17.2.1 TMyProtocol Enumeration

Specifies which protocol to use when connecting to server.

Unit

MyClasses

Syntax

```
TMyProtocol = (mpDefault, mpTCP, mpPipe, mpMemory, mpSSL, mpHttp);
```

Values

Value	Meaning
mpDefault	Similar to mpTCP, except the cases when you connect to a local server and the OS supports sockets (Unix) or named pipes (Windows), they are used instead of TCP/IP to connect to the server.
mpHttp	Uses HTTP Network Tunneling to connect to the server.
mpMemory	Uses SharedMem to connect to the server. Can be used with Direct set to False and libmysql.dll 4.1.
mpPipe	Uses NamedPipes to connect to the server.
mpSocket	Uses sockets to connect to the server. Can be used with Direct set to False and libmysql.dll 4.1.
mpSSL	Uses protected SSL connection with the server. To use SSL you need to set TMyConnection.SSLOptions.
mpTCP	Uses TCP/IP to connect to the server.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.17.3 Variables

Variables in the **MyClasses** unit.

Variables

Name	Description
__Strings65535ToMemo	Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.17.3.1 __Strings65535ToMemo Variable

Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields.

Unit

[MyClasses](#)

Syntax

```
__Strings65535ToMemo: boolean = True;
```

Remarks

Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields. It causes wrong description of these fields by MyDAC and truncating their data. To avoid these problems, MyDAC tries to restore the correct data type. This behaviour was introduced in MyDAC 5.10.0.9. To disable this behaviour, set the __Strings65535ToMemo variable to False.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.18 MyConnectionPool

This unit contains the TMyConnectionPoolManager class for managing connection pool.

Classes

Name	Description
[TMyConnectionPoolManager](#)	A class of methods that are used for managing MyDAC connection pool.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.18.1 Classes

Classes in the **MyConnectionPool** unit.

Classes

Name	Description
[TMyConnectionPoolManager](#)	A class of methods that are used for managing MyDAC

	connection pool.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.18.1.1 TMyConnectionPoolManager Class

A class of methods that are used for managing MyDAC connection pool.

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

Unit

[MyConnectionPool](#)

Syntax

```
TMyConnectionPoolManager = class(TCRConnectionPoolManager);
```

Remarks

Use the TMyConnectionPoolManager methods to manage MyDAC connection pool.

Inheritance Hierarchy

TCRConnectionPoolManager

  TMyConnectionPoolManager

See Also

- [Connection Pooling](#)

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.18.1.1.1 Members

[**TMyConnectionPoolManager**](#) class overview.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.19 MyDacVcl

This unit contains the visual constituent of MyDAC.

Classes

Name	Description
[TMyConnectDialog](#)	A class that provides a dialog box for user to supply his login information.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.19.1 Classes

Classes in the **MyDacVcl** unit.

Classes

Name	Description
[TMyConnectDialog](#)	A class that provides a dialog box for user to supply his login information.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.19.1.1 TMyConnectDialog 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 [TMyConnectDialog](#) members.

Unit

[MyDacVcl](#)

Syntax

```
TMyConnectDialog = class(TCustomConnectDialog);
```

Remarks

The TMyConnectDialog component is a direct descendant of TCustomConnectDialog class. Use TMyConnectDialog to provide dialog box for user to supply server name, user name, and password. You may want to customize appearance of dialog box using this class's properties.

Inheritance Hierarchy

TCustomConnectDialog

   TMyConnectDialog

See Also

- TCustomDAConnection.ConnectDialog

Request Support   DAC Forum   Provide Feedback

5.19.1.1.1 Members

TMyConnectDialog 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	Holds the TMyConnection component which uses TMyConnectDialog object.
DatabaseLabel	Used to specify a prompt for database edit.
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	Used to specify a prompt for password edit.

TCustomConnectDialog)	
PortLabel	Used to specify a prompt for port 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.
ShowDatabase	Used to display a field for entering database at connect dialog.
ShowPort	Used to display a field for entering port at connect dialog.
StoreLogInfo (inherited from **TCustomConnectDialog**)	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for username edit.

Methods

Name	Description
Execute (inherited from **TCustomConnectDialog**)	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
GetServerList (inherited from **TCustomConnectDialog**)	Retrieves a list of available server names.

Request Support   DAC Forum   Provide Feedback

5.19.1.1.2 Properties

Properties of the **TMyConnectDialog** class.

For a complete list of the **TMyConnectDialog** class members, see the TMyConnectDialog
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	Holds the TMyConnection component which uses TMyConnectDialog object.
DatabaseLabel	Used to specify a prompt for database edit.
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.
PortLabel	Used to specify a prompt for port 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.
ShowDatabase	Used to display a field for entering database at connect dialog.

[ShowPort](#)	Used to display a field for entering port at connect dialog.
[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

- [TMyConnectDialog Class](#)

- [TMyConnectDialog Class Members](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.19.1.1.2.1 Connection Property

Holds the TMyConnection component which uses TMyConnectDialog object.

Class

TMyConnectDialog

Syntax

```
property Connection: TCustomMyConnection;
```

Remarks

Use the Connection property to learn which TMyConnection component uses

TMyConnectDialog object. This property is read-only.

See Also

- [TCustomDAConnection.ConnectDialog](#)

Request Support   **DAC Forum**   **Provide Feedback**

5.19.1.1.2.2 DatabaseLabel Property

Used to specify a prompt for database edit.

Class

[TMyConnectDialog](#)

Syntax

```
property DatabaseLabel: string;
```

Remarks

Use the DatabaseLabel property to specify a prompt for database edit.

See Also

- [ShowDatabase](#)

© 1997-2025
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.19.1.1.2.3 PortLabel Property

Used to specify a prompt for port edit.

Class

[TMyConnectDialog](#)

Syntax

```
property PortLabel: string;
```

Remarks

Use the PortLabel property to specify a prompt for port edit.

See Also

- [ShowPort](#)

© 1997-2025
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.19.1.1.2.4 Show Database Property

Used to display a field for entering database at connect dialog.

Class

[TMyConnectDialog](#)

Syntax

```
property ShowDatabase: boolean default True;
```

Remarks

Use the ShowDatabase property to display a field for entering database at connect dialog.

The default value is True.

See Also

- [DatabaseLabel](#)

- [ShowPort](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.19.1.1.2.5 Show Port Property

Used to display a field for entering port at connect dialog.

Class

[TMyConnectDialog](#)

Syntax

```
property ShowPort: boolean default True;
```

Remarks

Use the ShowPort property to display a field for entering port at connect dialog.

The default value is True.

See Also

- [PortLabel](#)

- [ShowDatabase](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20 MyDataTypeMap

This unit contains the implementation of mapping between MySQL and Delphi data types.

Constants

Name	Description
[myBigint](#)	Used to map **BIGINT** to Delphi data types.
[myBigintUnsigned](#)	Used to map **BIGINT UNSIGNED** to Delphi data types.
[myBinary](#)	Used to map **BINARY** to Delphi data types.
[myBit](#)	Used to map **BIT** to Delphi data types.
[myBlob](#)	Used to map **BLOB** to Delphi data types.
[myChar](#)	Used to map **CHAR** to Delphi data types.
[myDate](#)	Used to map **DATE** to Delphi data types.
[myDatetime](#)	Used to map **DATETIME** to Delphi data types.
[myDecimal](#)	Used to map **DECIMAL** to Delphi data types.
[myDouble](#)	Used to map **DOUBLE** to Delphi data types.
[myEnum](#)	Used to map **ENUM** to Delphi data types.
[myFloat](#)	Used to map **FLOAT** to Delphi data types.
[myInt](#)	Used to map **INTEGER** to Delphi data types.
[myIntUnsigned](#)	Used to map **INTEGER UNSIGNED** to Delphi data

	types.
myJSON	Used to map **JSON** to Delphi data types.
myLongBlob	Used to map **LONGBLOB** to Delphi data types.
myLongText	Used to map **LONGTEXT** to Delphi data types.
myMedium	Used to map **MEDIUMINT** to Delphi data types.
myMediumBlob	Used to map **MEDIUMBLOB** to Delphi data types.
myMediumText	Used to map **MEDIUMTEXT** to Delphi data types.
myMediumUnsigned	Used to map **MEDIUMINT UNSIGNED** to Delphi data types.
myNull	Used to map **NULL** to Delphi data types.
mySet	Used to map **SET** to Delphi data types.
mySmall	Used to map **SMALLINT** to Delphi data types.
mySmallUnsigned	Used to map **SMALLINT UNSIGNED** to Delphi data types.
myText	Used to map **TEXT** to Delphi data types.
myTime	Used to map **TIME** to Delphi data types.
myTimestamp	Used to map **TIMESTAMP** to Delphi data types.
myTiny	Used to map **TINY** to Delphi data types.
myTinyBlob	Used to map **TINYBLOB** to Delphi data types.
myTinyText	Used to map **TINYTEXT** to Delphi data types.
myTinyUnsigned	Used to map **TINYINT UNSIGNED** to Delphi data types.

myUnsignedTypes	Used to map **UNSIGNED TYPES** to Delphi data types.
myVarbinary	Used to map **VARBINARY** to Delphi data types.
myVarchar	Used to map **VARCHAR** to Delphi data types.
myYear	Used to map **YEAR** to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1 Constants

Constants in the **MyDataTypeMap** unit.

Constants

Name	Description
myBigint	Used to map **BIGINT** to Delphi data types.
myBigintUnsigned	Used to map **BIGINT UNSIGNED** to Delphi data types.
myBinary	Used to map **BINARY** to Delphi data types.
myBit	Used to map **BIT** to Delphi data types.
myBlob	Used to map **BLOB** to Delphi data types.
myChar	Used to map **CHAR** to Delphi data types.
myDate	Used to map **DATE** to Delphi data types.
myDatetime	Used to map **DATETIME** to Delphi data types.
myDecimal	Used to map **DECIMAL** to Delphi data types.
myDouble	Used to map **DOUBLE** to Delphi data types.
myEnum	Used to map **ENUM** to

	Delphi data types.
[myFloat](#)	Used to map **FLOAT** to Delphi data types.
[myInt](#)	Used to map **INTEGER** to Delphi data types.
[myIntUnsigned](#)	Used to map **INTEGER UNSIGNED** to Delphi data types.
[myJSON](#)	Used to map **JSON** to Delphi data types.
[myLongBlob](#)	Used to map **LONGBLOB** to Delphi data types.
[myLongText](#)	Used to map **LONGTEXT** to Delphi data types.
[myMedium](#)	Used to map **MEDIUMINT** to Delphi data types.
[myMediumBlob](#)	Used to map **MEDIUMBLOB** to Delphi data types.
[myMediumText](#)	Used to map **MEDIUMTEXT** to Delphi data types.
[myMediumUnsigned](#)	Used to map **MEDIUMINT UNSIGNED** to Delphi data types.
[myNull](#)	Used to map **NULL** to Delphi data types.
[mySet](#)	Used to map **SET** to Delphi data types.
[mySmall](#)	Used to map **SMALLINT** to Delphi data types.
[mySmallUnsigned](#)	Used to map **SMALLINT UNSIGNED** to Delphi data types.
[myText](#)	Used to map **TEXT** to Delphi data types.
[myTime](#)	Used to map **TIME** to Delphi data types.
[myTimestamp](#)	Used to map **TIMESTAMP** to Delphi data types.
[myTiny](#)	Used to map **TINY** to Delphi data types.

myTinyBlob	Used to map **TINYBLOB** to Delphi data types.
myTinyText	Used to map **TINYTEXT** to Delphi data types.
myTinyUnsigned	Used to map **TINYINT UNSIGNED** to Delphi data types.
myUnsignedTypes	Used to map **UNSIGNED TYPES** to Delphi data types.
myVarbinary	Used to map **VARBINARY** to Delphi data types.
myVarchar	Used to map **VARCHAR** to Delphi data types.
myYear	Used to map **YEAR** to Delphi data types.

Request Support   **DAC Forum**   **Provide Feedback**

5.20.1.1 myBigint Constant

Used to map **BIGINT** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myBigint = myBase + 10;
```

Remarks

Use the **myBigint** constant to map the MySQL **BIGINT** data type to Delphi data types.

Request Support   **DAC Forum**   **Provide Feedback**

5.20.1.2 myBigintUnsigned Constant

Used to map **BIGINT UNSIGNED** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myBigintUnsigned = myBase + 11;
```

Remarks

Use the **myBigintUnsigned** constant to map the MySQL **BIGINT UNSIGNED** data type to

Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.3 myBinary Constant

Used to map **BINARY** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myBinary = myBase + 22;
```

Remarks

Use the **myBinary** constant to map the MySQL **BINARY** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.4 myBit Constant

Used to map **BIT** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myBit = myBase + 1;
```

Remarks

Use the **myBit** constant to map the MySQL **BIT** data type to Delphi data types.

© 1997-2025
Devart. All Rights
Reserved.   Request Support   DAC Forum   Provide Feedback

5.20.1.5 myBlob Constant

Used to map **BLOB** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myBlob = myBase + 25;
```

Remarks

Use the **myBlob** constant to map the MySQL **BLOB** data type to Delphi data types.

© 1997-2025
Devart. All Rights
Reserved.   Request Support   DAC Forum   Provide Feedback

5.20.1.6 myChar Constant

Used to map **CHAR** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myChar = myBase + 20;
```

Remarks

Use the **myChar** constant to map the MySQL **CHAR** data type to Delphi data types.

    [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.7 myDate Constant

Used to map **DATE** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myDate = myBase + 15;
```

Remarks

Use the **myDate** constant to map the MySQL **DATE** data type to Delphi data types.

    [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.8 myDatetime Constant

Used to map **DATETIME** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myDatetime = myBase + 17;
```

Remarks

Use the **myDatetime** constant to map the MySQL **DATETIME** data type to Delphi data types.

    [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.9 myDecimal Constant

Used to map **DECIMAL** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myDecimal = myBase + 14;
```

Remarks

Use the **myDecimal** constant to map the MySQL **DECIMAL** data type to Delphi data types.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.10 myDouble Constant

Used to map **DOUBLE** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myDouble = myBase + 13;
```

Remarks

Use the **myDouble** constant to map the MySQL **DOUBLE** data type to Delphi data types.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.11 myEnum Constant

Used to map **ENUM** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myEnum = myBase + 28;
```

Remarks

Use the **myEnum** constant to map the MySQL **ENUM** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.12 myFloat Constant

Used to map **FLOAT** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myFloat = myBase + 12;
```

Remarks

Use the **myFloat** constant to map the MySQL **FLOAT** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.13 myInt Constant

Used to map **INTEGER** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myInt = myBase + 8;
```

Remarks

Use the **myInt** constant to map the MySQL **INTEGER** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.14 myIntUnsigned Constant

Used to map **INTEGER UNSIGNED** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myIntUnsigned = myBase + 9;
```

Remarks

Use the **myIntUnsigned** constant to map the MySQL **INTEGER UNSIGNED** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.15 myJSON Constant

Used to map **JSON** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myJSON = myBase + 34;
```

Remarks

Use the **myJSON** constant to map the MySQL **JSON** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.16 myLongBlob Constant

Used to map **LONGBLOB** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myLongBlob = myBase + 30;
```

Remarks

Use the **myLongBlob** constant to map the MySQL **LONGBLOB** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.17 myLongText Constant

Used to map **LONGTEXT** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myLongText = myBase + 32;
```

Remarks

Use the **myLongText** constant to map the MySQL **LONGTEXT** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.18 myMedium Constant

Used to map **MEDIUMINT** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myMedium = myBase + 6;
```

Remarks

Use the **myMedium** constant to map the MySQL **MEDIUMINT** data type to Delphi data types.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.20.1.19 myMediumBlob Constant

Used to map **MEDIUMBLOB** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myMediumBlob = myBase + 31;
```

Remarks

Use the **myMediumBlob** constant to map the MySQL **MEDIUMBLOB** data type to Delphi

data types.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.20.1.20 myMediumText Constant

Used to map **MEDIUMTEXT** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myMediumText = myBase + 33;
```

Remarks

Use the **myMediumText** constant to map the MySQL **MEDIUMTEXT** data type to Delphi

data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.21 myMediumUnsigned Constant

Used to map **MEDIUMINT UNSIGNED** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myMediumUnsigned = myBase + 7;
```

Remarks

Use the **myMediumUnsigned** constant to map the MySQL **MEDIUMINT UNSIGNED** data

type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.22 myNull Constant

Used to map **NULL** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myNull = myBase + 35;
```

Remarks

Use the **myNull** constant to map the MySQL **NULL** value to Delphi data types.

© 1997-2025
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.23 mySet Constant

Used to map **SET** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
mySet = myBase + 29;
```

Remarks

Use the **mySet** constant to map the MySQL **SET** data type to Delphi data types.

© 1997-2025
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.24 mySmall Constant

Used to map **SMALLINT** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
mySmall = myBase + 4;
```

Remarks

Use the **mySmall** constant to map the MySQL **SMALLINT** data type to Delphi data types.

© 1997-2025   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.25 mySmallUnsigned Constant

Used to map **SMALLINT UNSIGNED** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
mySmallUnsigned = myBase + 5;
```

Remarks

Use the **mySmallUnsigned** constant to map the MySQL **SMALLINT UNSIGNED** data type

to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.26 myText Constant

Used to map **TEXT** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myText = myBase + 27;
```

Remarks

Use the **myText** constant to map the MySQL **TEXT** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.27 myTime Constant

Used to map **TIME** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myTime = myBase + 16;
```

Remarks

Use the **myTime** constant to map the MySQL **TIME** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.28 myTimestamp Constant

Used to map **TIMESTAMP** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myTimestamp = myBase + 18;
```

Remarks

Use the **myTimestamp** constant to map the MySQL **TIMESTAMP** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.29 myTiny Constant

Used to map **TINY** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myTiny = myBase + 2;
```

Remarks

Use the **myTiny** constant to map the MySQL **TINYINT** data type to Delphi data types.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.30 myTinyBlob Constant

Used to map **TINYBLOB** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myTinyBlob = myBase + 24;
```

Remarks

Use the **myTinyBlob** constant to map the MySQL **TINYBLOB** data type to Delphi data types.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.20.1.31 myTinyText Constant

Used to map **TINYTEXT** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myTinyText = myBase + 26;
```

Remarks

Use the **myTinyText** constant to map the MySQL **TINYTEXT** data type to Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.32 myTinyUnsigned Constant

Used to map **TINYINT UNSIGNED** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myTinyUnsigned = myBase + 3;
```

Remarks

Use the **myTinyUnsigned** constant to map the MySQL **TINYINT UNSIGNED** data type to

Delphi data types.

Request Support   DAC Forum   Provide Feedback

5.20.1.33 myUnsignedTypes Constant

Used to map **UNSIGNED TYPES** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myUnsignedTypes = [myTinyUnsigned, mySmallUnsigned,
myMediumUnsigned, myIntUnsigned, myBigintUnsigned];
```

Remarks

Use the **myUnsignedTypes** constant to map the MySQL **UNSIGNED** data types to Delphi

data types.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.20.1.34 myVarbinary Constant

Used to map **VARBINARY** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myVarbinary = myBase + 23;
```

Remarks

Use the **myVarbinary** constant to map the MySQL **VARBINARY** data type to Delphi data

types.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.20.1.35 myVarchar Constant

Used to map **VARCHAR** to Delphi data types.

Unit

[MyDataTypeMap](#)

Syntax

```
myVarchar = myBase + 21;
```

Remarks

Use the **myVarchar** constant to map the MySQL **VARCHAR** data type to Delphi data types.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.20.1.36 myYear Constant

Used to map **YEAR** to Delphi data types.

Unit

MyDataTypeMap

Syntax

```
myYear = myBase + 19;
```

Remarks

Use the **myYear** constant to map the MySQL **YEAR** data type to Delphi data types.

| Request Support | DAC Forum | Provide Feedback

5.21   MyDump

This unit contains implementation of the TMyDump component.

Classes

Name	Description
TMyDump	A component for storing database or its parts as a script and also for restoring database from the received script.
TMyDumpOptions	This class allows setting up the behaviour of the TMyDump class.

Types

Name	Description
TMyDumpObjects	Represents the set of TMyDumpObject.

Enumerations

Name	Description
TMyDumpObject	Specifies the object enumeration.

Request Support   DAC Forum   Provide Feedback

5.21.1 Classes

Classes in the **MyDump** unit.

Classes

Name	Description
TMyDump	A component for storing database or its parts as a script and also for restoring database from the received script.
TMyDumpOptions	This class allows setting up the behaviour of the TMyDump class.

Request Support   DAC Forum   Provide Feedback

5.21.1.1 TMyDump Class

A component for storing database or its parts as a script and also for restoring database from the received script.

For a list of all members of this type, see TMyDump members.

Unit

MyDump

Syntax

```
TMyDump = class(TDADump);
```

Remarks

Serves to store a database or its parts as a script and also to restore database from received script.

TMyDump behaviour is similar to mysqldump program.

Use TMyDump.Objects and TDADump.TableNames properties to specify the list of objects to be stored. By default, only tables structure and data are stored.

To generate a script call TDADump.Backup or TDADump.BackupQuery method. Resulted script can be viewed in TDADump.SQL.

TMyDump works on the client side. It causes large network loading. To backup database on the server side use TMyBackup. Unlike TMyBackup, TMyDump component allows to store not only tables, but database structure including users' rights.

Inheritance Hierarchy

TDADump
  TMyDump

See Also

- TDADump.Backup
- TDADump.Restore
- TMyBackup

Request Support   DAC Forum   Provide Feedback

5.21.1.1.1 Members

TMyDump 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 the statement that is being

	executed and the values and types of its parameters.
[Objects](#)	Used to set the enumeration of object types that will be described on calling [TDADump.Backup](#).
[Options](#)	Specifies the behaviour of the TMyDump component.
[SQL](#) (inherited from **[TDADump](#)**)	Used to set or get the dump script.
[StoredProcNames](#)	Holds the list of procedures which will be used in the script.
[TableNames](#) (inherited from **[TDADump](#)**)	Used to set the names of the tables to dump.
[TriggerNames](#)	Holds the list of triggers which will be used in the script.

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.Ba ckupToFile(System.String) or M:Devart.Dac.TDADump.Ba ckupToStream(Borland.Vcl. TStream) method execution progress.
[OnError](#) (inherited from **[TDADump](#)**)	Occurs when MySQL raises some error on [TDADump.Restore](#).
[OnRestoreProgress](#) (inherited from **[TDADump](#)**)	Occurs to indicate the [TDADump.Restore](#), [TDADump.RestoreFromFile](#), or [TDADump.RestoreFromStr eam](#) method execution progress.

Request Support   **DAC Forum**   **Provide Feedback**

5.21.1.1.2 Properties

Properties of the **TMyDump** class.

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

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 the statement that is being executed and the values and types of its parameters.
[Objects](#)	Used to set the enumeration of object types that will be described on calling [TDADump.Backup](#).
[Options](#)	Specifies the behaviour of the TMyDump component.

SQL (inherited from **TDADump**)	Used to set or get the dump script.
StoredProcNames	Holds the list of procedures which will be used in the script.
TableNames (inherited from **TDADump**)	Used to set the names of the tables to dump.
TriggerNames	Holds the list of triggers which will be used in the script.

See Also

- TMyDump Class

- TMyDump Class Members

Request Support   DAC Forum   Provide Feedback

5.21.1.1.2.1 Connection Property

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

Class

TMyDump

Syntax

```
property Connection: TCustomMyConnection;
```

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 TCustomMyConnection objects.

At runtime, set Connection property to reference an existing TCustomMyConnection object.

See Also

- TCustomMyConnection

Request Support   DAC Forum   Provide Feedback

5.21.1.1.2.2 Objects Property

Used to set the enumeration of object types that will be described on calling
TDADump.Backup.

Class

TMyDump

Syntax

```
property Objects: TMyDumpObjects default [doTables, doViews,
doData];
```

Remarks

Use the Object property to set the enumeration of object types that will be described on
calling TDADump.Backup.

See Also

- TDADump.Backup

- TDADump.BackupToFile

- TDADump.BackupToStream

- TDADump.TableNames

- StoredProcNames

Request Support   DAC Forum   Provide Feedback

5.21.1.1.2.3 Options Property

Specifies the behaviour of the TMyDump component.

Class

TMyDump

Syntax

```
property Options: TMyDumpOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TMyDump component.

Descriptions of all options are in the table below.

Option Name	Description
AddLock	Used to execute LOCK TABLE before data insertion.
CommitBatchSize	Used to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data.
DisableKeys	Allows disabling keys check when inserting records.
HexBlob	Used to present BLOB values in hexdecimal notation.
InsertType	Used to specify the method of rows insertion into a table.
UseDelayedIns	Set to use INSERT DELAYED.
UseExtSyntax	Set to use extended syntax of INSERT on data insertion.

See Also

- Objects

Request Support   DAC Forum   Provide Feedback

5.21.1.1.2.4 StoredProcNames Property

Holds the list of procedures which will be used in the script.

Class

TMyDump

Syntax

```
property StoredProcNames: string;
```

Remarks

Use the StoredProcNames property to hold the list of procedures which will be used in the script. Makes sense only on execution TDADump.Backup, M:Devart.Dac.TDADump.BackupToFile(System.String), M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) and doStoredProcs set at the Objects. Names are separated by comma or semicolon. If it has an empty value, all procs presented in the database will be processed.

See Also

- TDADump.Backup

- TDADump.BackupToFile

- TDADump.BackupToStream

- Objects

- TCustomDAConnection.GetStoredProcNames

Request Support   DAC Forum   Provide Feedback

5.21.1.1.2.5 TriggerNames Property

Holds the list of triggers which will be used in the script.

Class

TMyDump

Syntax

```
property TriggerNames: string;
```

Remarks

Use the TriggerNames property to hold the list of triggers which will be used in the script. Makes sense only on execution of TMyDump.Backup, TMyDump.BackupToFile, TMyDump.BackupToStream, and doTriggers set at the Objects. Names are separated by comma or semicolon. If it has an empty value and the TMyDump.TableNames property is empty, then all triggers presented in the database will be processed. If it has an empty value

and the TMyDump.TableNames property is not empty, then all triggers for the specified tables will be processed.

Request Support   DAC Forum   Provide Feedback

5.21.1.2 TMyDumpOptions Class

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

For a list of all members of this type, see TMyDumpOptions members.

Unit

MyDump

Syntax

```
TMyDumpOptions = class(TDADumpOptions);
```

Inheritance Hierarchy

TDADumpOptions

    TMyDumpOptions

Request Support   DAC Forum   Provide Feedback

5.21.1.2.1 Members

TMyDumpOptions class overview.

Properties

Name	Description
AddDrop (inherited from **TDADumpOptions**)	Used to add DROP statements to a script before CREATE statements.
AddLock	Used to execute LOCK TABLE before data insertion.
CommitBatchSize	Used to add COMMIT statement to script after

	inserting every CommitBatchSize strings when dumping table data.
[CompleteInsert](#) (inherited from **[TDADumpOptions](#)**)	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
[DisableKeys](#)	Allows disabling keys check when inserting records.
[GenerateHeader](#) (inherited from **[TDADumpOptions](#)**)	Used to add a comment header to a script.
[HexBlob](#)	Used to present BLOB values in hexdecimal notation.
[InsertType](#)	Used to specify the method of rows insertion into a table.
[QuoteNames](#) (inherited from **[TDADumpOptions](#)**)	Used for TDADump to quote all database object names in generated SQL statements.
[UseDelayedIns](#)	Set to use INSERT DELAYED.
[UseExtSyntax](#)	Set to use extended syntax of INSERT on data insertion.

Request Support   **DAC Forum**   **Provide Feedback**

5.21.1.2.2 Properties

Properties of the **TMyDumpOptions** class.

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

Published

Name	Description
[AddDrop](#) (inherited from **[TDADumpOptions](#)**)	Used to add DROP statements to a script before CREATE statements.
[AddLock](#)	Used to execute LOCK TABLE before data

	insertion.
[CommitBatchSize](#)	Used to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data.
[CompleteInsert](#) (inherited from **[TDADumpOptions](#)**)	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
[DisableKeys](#)	Allows disabling keys check when inserting records.
[GenerateHeader](#) (inherited from **[TDADumpOptions](#)**)	Used to add a comment header to a script.
[HexBlob](#)	Used to present BLOB values in hexdecimal notation.
[InsertType](#)	Used to specify the method of rows insertion into a table.
[QuoteNames](#) (inherited from **[TDADumpOptions](#)**)	Used for TDADump to quote all database object names in generated SQL statements.
[UseDelayedIns](#)	Set to use INSERT DELAYED.
[UseExtSyntax](#)	Set to use extended syntax of INSERT on data insertion.

See Also

- [TMyDumpOptions Class](#)

- [TMyDumpOptions Class Members](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.21.1.2.2.1 AddLock Property

Used to execute LOCK TABLE before data insertion.

Class

[TMyDumpOptions](#)

Syntax

```
property AddLock: boolean default True;
```

Remarks

Use the AddLock property to execute LOCK TABLE before data insertion. Used only with doData in TMyDump.Objects.

Request Support   DAC Forum   Provide Feedback

5.21.1.2.2.2 CommitBatchSize Property

Used to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data.

Class

TMyDumpOptions

Syntax

```
property CommitBatchSize: integer default 0;
```

Remarks

Use the CommitBatchSize property to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data. Use this property to boost the productivity when recovering large amounts of data.

Request Support   DAC Forum   Provide Feedback

5.21.1.2.2.3 DisableKeys Property

Allows disabling keys check when inserting records.

Class

TMyDumpOptions

Syntax

```
property DisableKeys: boolean default False;
```

Remarks

Add /*!40000 ALTER TABLE ... DISABLE KEYS */ before inserting data. Used only with doData in TMyDump.Objects.

Request Support   DAC Forum   Provide Feedback

5.21.1.2.2.4 HexBlob Property

Used to present BLOB values in hexdecimal notation.

Class

TMyDumpOptions

Syntax

```
property HexBlob: boolean default False;
```

Remarks

If the HexBlob property is True, the BLOB values are presented in hexdecimal notation.

Request Support   DAC Forum   Provide Feedback

5.21.1.2.2.5 InsertType Property

Used to specify the method of rows insertion into a table.

Class

TMyDumpOptions

Syntax

```
property InsertType: TMyInsertType default itInsert;
```

Remarks

If the InsertType property is set to *itInsert* (by default), new rows are inserted into an existing

table, an error is raised in case if there is a duplicate entry.

If the InsertType property is set to *itInsertIgnore* , the insert operation fails silently for rows containing the unmatched value, but inserts rows that are matched without generating an error.

If the InsertType property is set to *itReplaceInto* if an old row in the table has the same value as a new row, the old row is deleted before the new row is inserted.

Request Support   DAC Forum   Provide Feedback

5.21.1.2.2.6 UseDelayedIns Property

Set to use INSERT DELAYED.

Class

`TMyDumpOptions`

Syntax

```
property UseDelayedIns: boolean default False;
```

Remarks

Set the UseDelayedIns property to use INSERT DELAYED. Used only with doData in TMyDump.Objects.

Request Support   DAC Forum   Provide Feedback

5.21.1.2.2.7 UseExtSyntax Property

Set to use extended syntax of INSERT on data insertion.

Class

`TMyDumpOptions`

Syntax

```
property UseExtSyntax: boolean default True;
```

Remarks

Set the UseExtSyntax propery to use extended syntax of INSERT on data insertion. Used only with doData in [TMyDump.Objects](#).

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.21.2 Types

Types in the **MyDump** unit.

Types

Name	Description
[TMyDumpObjects](#)	Represents the set of [TMyDumpObject](#).

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.21.2.1 TMyDumpObjects Set

Represents the set of [TMyDumpObject](#).

Unit

[MyDump](#)

Syntax

```
TMyDumpObjects = set of TMyDumpObject;
```

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.21.3 Enumerations

Enumerations in the **MyDump** unit.

Enumerations

Name	Description

TMyDumpObject	Specifies the object enumeration.

Request Support   **DAC Forum**   **Provide Feedback**

5.21.3.1 TMyDumpObject Enumeration

Specifies the object enumeration.

Unit

MyDump

Syntax

```
TMyDumpObject = (doDatabase, doUsers, doStoredProcs, doTables,
doData, doViews, doTriggers);
```

Values

Value	Meaning
doData	Stores data from the tables. The list of the tables is specified in TDADump.TableNames. If TableNames is not set, all tables are used.
doDatabase	Adds to the received script "CREATE DATABASE..." command.
doStoredProcs	Stores stored procedures structure in the script. The list of procedures is specified in TMyDump.StoredProcNames. If StoredProcNames is not set, all procedures are used.
doTables	Stores tables structure in the script. The list of tables is specified in TDADump.TableNames. If TableNames is not set, all tables are used.
doTriggers	Stores queries for creating triggers in the script. The list of triggers is specified in TMyDump.TriggerNames. If TriggerNames is not set and the TMyDump.TableNames property is empty, then all triggers of the database will be processed. If TriggerNames is not set and the TMyDump.TableNames property is not empty, then all triggers for the specified tables will be processed.
doUsers	Stores user privileges in the script.
doViews	Stores queries for creating Views in a script. The Views list is taken from TableNames.

Request Support   **DAC Forum**   **Provide Feedback**

5.22 MyEmbConnection

This unit contains implementation of the TMyEmbConnection component.

Classes

Name	Description
TMyEmbConnection	A component for establishing connection to Embedded MySQL server.

Request Support   DAC Forum   Provide Feedback

5.22.1 Classes

Classes in the **MyEmbConnection** unit.

Classes

Name	Description
TMyEmbConnection	A component for establishing connection to Embedded MySQL server.

Request Support   DAC Forum   Provide Feedback

5.22.1.1 TMyEmbConnection Class

A component for establishing connection to Embedded MySQL server.

For a list of all members of this type, see TMyEmbConnection members.

Unit

MyEmbConnection

Syntax

```
TMyEmbConnection = class(TCustomMyConnection);
```

Remarks

TMyEmbConnection component is used to establish a connection to MySQL Embedded server and provide its enhanced support.

Using TMyEmbConnection allows refusing from using configuration file (my.ini) and also lets you handle MySQL server messages written into log files.

Inheritance Hierarchy

TCustomDAConnection

   TCustomMyConnection

      TMyEmbConnection

See Also

- TCustomMyConnection

- TMyConnection

- Embedded Server

Request Support   DAC Forum   Provide Feedback

5.22.1.1.1 Members

TMyEmbConnection class overview.

Properties

Name	Description
BaseDir	Used to set the base path for MySQL Embedded server.
ClientVersion (inherited from **TCustomMyConnection**)	Contains the version of the MySQL Client library.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectionTimeout (inherited from **TCustomMyConnection**)	Used to specify the amount of time to attempt to establish a connection.

ConnectString (inherited from **TCustomDAConnection**)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Database (inherited from **TCustomMyConnection**)	Used to specify a database name that is a default source of data for SQL queries once a connection is established.
DataDir	Used to set the path to the data directory.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
IsolationLevel (inherited from **TCustomMyConnection**)	Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.
LastWarningCount (inherited from **TCustomMyConnection**)	Used to get the number of warnings resulting from the execution of the most recent statement.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options (inherited from **TCustomMyConnection**)	Specifies the behaviour of the TMyConnectionOptions object.
Params	Used to specify the list of command line parameters for Embedded server.
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 **TCustomMyConnection**)	Holds the version of MySQL server.
ThreadId (inherited from **TCustomMyConnection**)	Used to return the thread ID of the current connection.
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 **TCustomMyConnection**)	Shares database connection between the TCustomMyConnection components.
Commit (inherited from **TCustomDAConnection**)	Commits current transaction.
Connect (inherited from **TCustomDAConnection**)	Establishes a connection to the server.
CreateDataSet (inherited from **TCustomMyConnection**)	Returns a new instance of TCustomMyDataSet class and associates it with this connection object.
CreateSQL (inherited from **TCustomDAConnection**)	Creates a component for queries execution.
Disconnect (inherited from **TCustomDAConnection**)	Performs disconnect.
ExecProc (inherited from **TCustomDAConnection**)	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx (inherited from **TCustomDAConnection**)	Allows to execute a stored procedure or function.
ExecSQL (inherited from **TCustomMyConnection**)	Executes any SQL statement outside TMyQuery or TMyCommand components.
ExecSQLEx (inherited from **TCustomDAConnection**)	Executes any SQL statement outside the TQuery or TSQL components.
GetCharsetNames (inherited from	Populates a string list with the names of available

TCustomMyConnection)	charsets.
GetDatabaseNames (inherited from **TCustomDAConnection**)	Returns a database list from the server.
GetExecuteInfo (inherited from **TCustomMyConnection**)	Returns the result of the last query execution.
GetKeyFieldNames (inherited from **TCustomDAConnection**)	Provides a list of available key field names.
GetStoredProcNames (inherited from **TCustomDAConnection**)	Returns a list of stored procedures from the server.
GetTableNames (inherited from **TCustomDAConnection**)	Provides a list of available tables names.
GetTriggerNames (inherited from **TCustomMyConnection**)	Returns a list of triggers from the server.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
PerformConnect (inherited from **TCustomDAConnection**)	Establishes a connection to the server regardless of the LoginPrompt property.
Ping (inherited from **TCustomDAConnection**)	Used to check state of connection to the server.
ReleaseSavepoint (inherited from **TCustomMyConnection**)	Releases the specified savepoint without affecting any work that has been performed after its creation.
RemoveFromPool (inherited from **TCustomDAConnection**)	Marks the connection that should not be returned to the pool after disconnect.
Rollback (inherited from **TCustomDAConnection**)	Discards all current data changes and ends transaction.
RollbackToSavepoint (inherited from **TCustomMyConnection**)	Cancels all updates for the current transaction.
Savepoint (inherited from **TCustomMyConnection**)	Defines a point in the transaction to which you can

	roll back later.
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.
OnLog	Occurs when MySQL server writes down to General Query Log.
OnLogError	Occurs when MySQL server writes down to Error Log.

Request Support   **DAC Forum**   **Provide Feedback**

5.22.1.1.2 Properties

Properties of the **TMyEmbConnection** class.

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

Public

Name	Description
ClientVersion (inherited from **TCustomMyConnection**)	Contains the version of the MySQL Client library.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectionTimeout (inherited from **TCustomMyConnection**)	Used to specify the amount of time to attempt to establish a connection.

ConnectString (inherited from **TCustomDAConnection**)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Database (inherited from **TCustomMyConnection**)	Used to specify a 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 **TCustomMyConnection**)	Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.
LastWarningCount (inherited from **TCustomMyConnection**)	Used to get the number of warnings resulting from the execution of the most recent statement.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
Options (inherited from **TCustomMyConnection**)	Specifies the behaviour of the TMyConnectionOptions 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 **TCustomMyConnection**)	Holds the version of MySQL server.
ThreadId (inherited from **TCustomMyConnection**)	Used to return the thread ID of the current connection.

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

Published

Name	Description
BaseDir	Used to set the base path for MySQL Embedded server.
DataDir	Used to set the path to the data directory.
Params	Used to specify the list of command line parameters for Embedded server.

See Also

- TMyEmbConnection Class

- TMyEmbConnection Class Members

Request Support   DAC Forum   Provide Feedback

5.22.1.1.2.1 BaseDir Property

Used to set the base path for MySQL Embedded server.

Class

TMyEmbConnection

Syntax

```
property BaseDir: string stored False;
```

Remarks

Use the BaseDir property to set the base path for MySQL Embedded server. All paths are usually resolved relative to this. Corresponds to --basedir parameter. See MySQL Reference Manual for detailed description.

The default value is '.' and all auxiliary files must be located in the same folder with the

executable file of the project.

See Also

- DataDir

- Params

- Embedded Server

Request Support   DAC Forum   Provide Feedback

5.22.1.1.2.2 DataDir Property

Used to set the path to the data directory.

Class

`TMyEmbConnection`

Syntax

```
property DataDir: string stored False;
```

Remarks

Use the DataDir property to set the path to the data directory. Corresponds to --datadir parameter. See MySQL Reference Manual for a detailed description.

The default value is 'data' and all data files must be in the subfolder data by the path specified in BaseDir.

It is convenient to use this property for cases when the program is started from ReadOnly data carrier (CD-ROM, network etc). Also its usage can be suitable to separate data of different users.

See Also

- BaseDir

- Params

- Embedded Server

Request Support   DAC Forum   Provide Feedback

5.22.1.1.2.3 Params Property

Used to specify the list of command line parameters for Embedded server.

Class

[TMyEmbConnection](#)

Syntax

```
property Params: TStrings;
```

Remarks

Use the Params property to specify the list of command line parameters for Embedded server.

Pay attention that all paths must be set through "/" but not "\".

Params property also stores values for BaseDir and DataDir.

If no parameters are set in Params property its value will be obtained from the file of configuration of EmbServer (my.ini or my.cnf). Please see MySQL Reference for details.

libmysqld library is reloaded only on changing parameters.

Note, parameters names are case-sensitive.

See Also

- [BaseDir](#)
- [DataDir](#)
- [Embedded Server](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.22.1.1.3 Events

Events of the **TMyEmbConnection** class.

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

Public

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

Published

Name	Description
OnLog	Occurs when MySQL server writes down to General Query Log.
OnLogError	Occurs when MySQL server writes down to Error Log.

See Also

- TMyEmbConnection Class

- TMyEmbConnection Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.22.1.1.3.1 OnLog Event

Occurs when MySQL server writes down to General Query Log.

Class

TMyEmbConnection

Syntax

```
property OnLog: TMyLogEvent;
```

Remarks

The OnLog event occurs when MySQL server writes down to General Query Log, the same as on using --log option. See MySQL Reference Manual for detailed

description.

On assigning handler for OnLog event MySQL server does not make output to common log-file.

This event is available only for Win32.

See Also

- OnLogError

- Embedded Server

Request Support   DAC Forum   Provide Feedback

5.22.1.1.3.2 OnLogError Event

Occurs when MySQL server writes down to Error Log.

Class

`TMyEmbConnection`

Syntax

```
property OnLogError: TMyLogEvent;
```

Remarks

The OnLogError event occurs when MySQL server writes down to Error Log, the same as on using --log-error option. See MySQL Reference Manual for detailed description. It is convenient to use OnLogError event to search errors in the server configuration, as often after error message is displayed an application is terminated.

On assigning handler for OnLogError event MySQL server does not make output to common error log-file.

This event is available only for Win32.

See Also

- [OnLog](#)

- [Embedded Server](#)

5.23   MyLoader

This unit contains implementation of the TMyLoader component.

Classes

Name	Description
[TMyColumn](#)	A component representing the attributes for column loading.
[TMyLoader](#)	TMyLoader allows to load external data into the server database.

Types

Name	Description
[TMyLoaderOptions](#)	Represents the set of [TMyLoaderOption](#).

Enumerations

Name	Description
[TMyDuplicateKeys](#)	Specifies the way conflicts with duplicated key values will be resolved.
[TMyLoaderOption](#)	Specifies the behaviour of a TMyLoader object.

5.23.1 Classes

Classes in the **MyLoader** unit.

Classes

Name	Description
TMyColumn	A component representing the attributes for column loading.
TMyLoader	TMyLoader allows to load external data into the server database.

Request Support   DAC Forum   Provide Feedback

5.23.1.1 TMyColumn Class

A component representing the attributes for column loading.

For a list of all members of this type, see TMyColumn members.

Unit

`MyLoader`

Syntax

```
TMyColumn = class(TDAColumn);
```

Remarks

Each TMyLoader uses TDAColumns to maintain a collection of TMSColumn objects. TMyColumn object represents the attributes for column loading. Every TMyColumn 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 TMyLoader component.

Inheritance Hierarchy

TDAColumn

  TMyColumn

See Also

- TMyLoader

- TDAColumns

Request Support   DAC Forum   Provide Feedback

5.23.1.1.1 Members

TMyColumn 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.

Request Support   DAC Forum   Provide Feedback

5.23.1.2 TMyLoader Class

TMyLoader allows to load external data into the server database.

For a list of all members of this type, see TMyLoader members.

Unit

`MyLoader`

Syntax

```
TMyLoader = class(TDALoader);
```

Remarks

TMyLoader serves for fast loading of data to the server.

TMyLoader work is based on generation INSERT statements which insert data by several rows for one time (see TMyLoader.RowsPerQuery).

This class is used to simplify INSERT statements generation.

To affect on performance set MyLoader.Connection.Options.Compression property to True.

The quicker way is to generate text file on the client side and load it using TMyBackup.Restore method with Mode = bmText.

Inheritance Hierarchy

TDALoader

   TMyLoader

See Also

- TMyBackup

Request Support   DAC Forum   Provide Feedback

5.23.1.2.1 Members

TMyLoader class overview.

Properties

Name	Description
Columns (inherited from **TDALoader**)	Used to add a TDAColumn object for each field that will be loaded.
Connection	Used to specify a connection object that will be used to connect to a data store.
DuplicateKeys	Used to specify in what way conflicts with duplicated key values will be resolved.
Options	Specifies the behaviour of TMyLoader object.
RowsPerQuery	Used to get or set the number of rows that will be send to the server for one time.
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 (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.

Request Support   DAC Forum   Provide Feedback

5.23.1.2.2 Properties

Properties of the **TMyLoader** class.

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

Public

Name	Description
Columns (inherited from **TDALoader**)	Used to add a TDAColumn object for each field that will

	be loaded.
TableName (inherited from **TDALoader**)	Used to specify the name of the table to which data will be loaded.

Published

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
DuplicateKeys	Used to specify in what way conflicts with duplicated key values will be resolved.
Options	Specifies the behaviour of TMyLoader object.
RowsPerQuery	Used to get or set the number of rows that will be send to the server for one time.

See Also

- TMyLoader Class

- TMyLoader Class Members

Request Support   **DAC Forum**   **Provide Feedback**

5.23.1.2.2.1 Connection Property

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

Class

TMyLoader

Syntax

```
property Connection: TCustomMyConnection;
```

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 TMyConnection objects.

At runtime, set the Connection property to reference an existing TMyConnection object.

See Also

• TCustomMyConnection

Request Support   DAC Forum   Provide Feedback

5.23.1.2.2.2 DuplicateKeys Property

Used to specify in what way conflicts with duplicated key values will be resolved.

Class

TMyLoader

Syntax

```
property DuplicateKeys: TMyDuplicateKeys default dkNone;
```

Remarks

Use the DuplicateKeys property to specify in what way conflicts with duplicated key values will be resolved.

Request Support   DAC Forum   Provide Feedback

5.23.1.2.2.3 Options Property

Specifies the behaviour of TMyLoader object.

Class

TMyLoader

Syntax

```
property Options: TMyLoaderOptions default [];
```

Remarks

Set the properties of Options to specify the behaviour of a TMyLoader object.

© 1997-2025
Devart. All Rights
Reserved. **Request Support**   **DAC Forum**   **Provide Feedback**

5.23.1.2.2.4 RowsPerQuery Property

Used to get or set the number of rows that will be send to the server for one time.

Class

TMyLoader

Syntax

```
property RowsPerQuery: integer default 0;
```

Remarks

Use the RowsPerQuery property to get or set the number of rows that will be send to the server for one time. The default value is 0. In this case rows will be grouped by 16Kb (the default value of net_buffer_length).

© 1997-2025
Devart. All Rights
Reserved. **Request Support**   **DAC Forum**   **Provide Feedback**

5.23.2 Types

Types in the **MyLoader** unit.

Types

Name	Description
TMyLoaderOptions	Represents the set of **TMyLoaderOption**.

© 1997-2025
Devart. All Rights
Reserved. **Request Support**   **DAC Forum**   **Provide Feedback**

5.23.2.1 TMyLoaderOptions Set

Represents the set of TMyLoaderOption.

Unit

MyLoader

Syntax

```
TMyLoaderOptions = set of TMyLoaderOption;
```

Request Support   DAC Forum   Provide Feedback

5.23.3 Enumerations

Enumerations in the **MyLoader** unit.

Enumerations

Name	Description
TMyDuplicateKeys	Specifies the way conflicts with duplicated key values will be resolved.
TMyLoaderOption	Specifies the behaviour of a TMyLoader object.

Request Support   DAC Forum   Provide Feedback

5.23.3.1 TMyDuplicateKeys Enumeration

Specifies the way conflicts with duplicated key values will be resolved.

Unit

MyLoader

Syntax

```
TMyDuplicateKeys = (dkNone, dkIgnore, dkReplace);
```

Values

Value	Meaning
dkIgnore	The new record with duplicated key value will be ignored silently.
dkNone	An error will be raised and loading process will be stopped after an attempt to insert a record with already existing key value.
dkReplace	The old record in database with duplicated key values will be silently replaced with the new one.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.23.3.2 TMyLoaderOption Enumeration

Specifies the behaviour of a TMyLoader object.

Unit

MyLoader

Syntax

```
TMyLoaderOption = (loLock, loDelayed);
```

Values

Value	Meaning
loDelayed	Uses INSERT DELAYED syntax.
loLock	Locks tables while inserting data.

© 1997-2025 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.24 MyScript

This unit contains implementation of the TMyScript component.

Classes

Name	Description
TMyScript	A component for executing several SQL statements one by one.

© 1997-2025   Request Support   DAC Forum   Provide Feedback

5.24.1 Classes

Classes in the **MyScript** unit.

Classes

Name	Description
TMyScript	A component for executing several SQL statements one by one.

Request Support   DAC Forum   Provide Feedback

5.24.1.1 TMyScript Class

A component for executing several SQL statements one by one.

For a list of all members of this type, see TMyScript members.

Unit

MyScript

Syntax

```
TMyScript = 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 TMyCommand. Usually it is not a good solution. With only one TMyScript component you can execute several SQL statements as one. This sequence of statements is named script. To separate single statements use semicolon (;), slash (/) ,and DELIMITER . 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 TMyScript 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

TDAScript

   TMyScript

See Also

- TMyCommand

   Request Support   DAC Forum   Provide Feedback

5.24.1.1.1 Members

TMyScript 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 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 unit small queries into blocks for block executing if possible.

Methods

Name	Description
BreakExec (inherited from **TDAScript**)	Stops script execution.
ErrorOffset (inherited from **TDAScript**)	Used to get the offset of the statement if the Execute method raised an exception.
Execute (inherited from **TDAScript**)	Executes a script.
ExecuteFile (inherited from **TDAScript**)	Executes SQL statements contained in a file.
ExecuteNext (inherited from **TDAScript**)	Executes the next statement in the script and then stops.
ExecuteStream (inherited from **TDAScript**)	Executes SQL statements contained in a stream object.
FindMacro (inherited from **TDAScript**)	Finds a macro with the specified name.
MacroByName (inherited from **TDAScript**)	Finds a macro with the specified name.

Events

Name	Description
AfterExecute (inherited from **TDAScript**)	Occurs after a SQL script execution.
BeforeExecute (inherited from **TDAScript**)	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError (inherited from **TDAScript**)	Occurs when MySQL raises an error.

Request Support   **DAC Forum**   **Provide Feedback**

5.24.1.1.2 Properties

Properties of the **TMyScript** class.

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

Public

Name	Description
EndLine (inherited from **TDAScript**)	Used to get the current statement last line number in a script.
EndOffset (inherited from **TDAScript**)	Used to get the offset in the last line of the current statement.
EndPos (inherited from **TDAScript**)	Used to get the end position of the current statement.
StartLine (inherited from **TDAScript**)	Used to get the current statement start line number in a script.
StartOffset (inherited from **TDAScript**)	Used to get the offset in the first line of the current statement.
StartPos (inherited from **TDAScript**)	Used to get the start position of the current statement in a script.
Statements (inherited from **TDAScript**)	Contains a list of statements obtained from the SQL property.

Published

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 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.
SQL (inherited from **TDAScript**)	Used to get or set script text.
UseOptimization	Used to unit small queries into blocks for block executing if possible.

See Also

- TMyScript Class

- TMyScript Class Members

Request Support   DAC Forum   Provide Feedback

5.24.1.1.2.1 Connection Property

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

Class

TMyScript

Syntax

```
property Connection: TCustomMyConnection;
```

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 TMyConnection objects.

At run-time, set Connection property to reference an existing TMyConnection object.

See Also

- TCustomMyConnection

Request Support   DAC Forum   Provide Feedback

5.24.1.1.2.2 DataSet Property

Used to retrieve the results of SELECT statements execution inside a script.

Class

`TMyScript`

Syntax

```
property DataSet: TCustomMyDataSet;
```

Remarks

Use the DataSet property if you need to retrieve the results of SELECT statements execution inside a script.

See Also

- TDAScript.Execute

Request Support   DAC Forum   Provide Feedback

5.24.1.1.2.3 UseOptimization Property

Used to unit small queries into blocks for block executing if possible.

Class

[TMyScript](#)

Syntax

```
property UseOptimization: boolean;
```

Remarks

Use the UseOptimization property to unit small queries into blocks for block executing if possible. The UseOptimization option does not afect the [TDAScript.ExecuteNext](#) method performance. It works only for the [TDAScript.Execute](#) method.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.25 MyServerControl

This unit contains implementation of the TMyServerControl component.

Classes

Name	Description
[TMyServerControl](#)	A component for server control and standard service tasks execution.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.25.1 Classes

Classes in the **MyServerControl** unit.

Classes

Name	Description

TMyServerControl	A component for server control and standard service tasks execution.

Request Support   DAC Forum   Provide Feedback

5.25.1.1 TMyServerControl Class

A component for server control and standard service tasks execution.

For a list of all members of this type, see TMyServerControl members.

Unit

`MyServerControl`

Syntax

```
TMyServerControl = class(TCustomMyDataSet);
```

Remarks

TMyServerControl is a direct descendant of TCustomMyDataSet component. It serves to control the server and execute standard service tasks. List of the functions can be divided in the following groups:

- Create and delete a database (TMyServerControl.CreateDatabase, TMyServerControl.DropDatabase);

- Serves to manage MySQL services. available only for Windows (TMyServerControl.ServiceStart, TMyServerControl.ServiceStop, TMyServerControl.GetServiceNames, TMyServerControl.ServiceStatus);

- Manage MySQL server system variables (TMyServerControl.Variables, TMyServerControl.ShowVariables)

- Flush server data (TMyServerControl.Flush);

- Manage tables (TMyServerControl.AnalyzeTable, TMyServerControl.OptimizeTable, TMyServerControl.CheckTable, TMyServerControl.RepairTable);

- Manage processes (TMyServerControl.ShowProcessList, TMyServerControl.KillProcess);

- Obtain current information about the server ([TMyServerControl.ShowStatus](#)).

The last three groups show report as DataSet.

Inheritance Hierarchy

[TMemDataSet](#)

  [TCustomDADataSet](#)

    [TCustomMyDataSet](#)

      TMyServerControl

See Also

- [TMyQuery](#)

| [Request Support](#) | [DAC Forum](#) | [Provide Feedback](#) |

5.25.1.1.1 Members

[TMyServerControl](#) 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.
[CommandTimeout](#) (inherited from **[TCustomMyDataSet](#)**)	Used to specify the amount of time to attempt execution of a command.
[Conditions](#) (inherited from **[TCustomDADataSet](#)**)	Used to add WHERE conditions to a query
[Connection](#)	Used to specify a connection object to use for connecting to a data store.
[DataTypeMap](#) (inherited from **[TCustomDADataSet](#)**)	Used to set data type mapping rules
[Debug](#)	Used to display executing statement.

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 **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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 (inherited from **TCustomMyDataSet**)	Specifies when to perform locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
Options (inherited from **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.

ReadOnly (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.
SmartFetch (inherited from **TCustomMyDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
TableNames	Holds a list of tables that will be processed by the

		TMyServerControl.AnalyzeTable, TMyServerControl.OptimizeTable, TMyServerControl.CheckTable, TMyServerControl.RepairTable methods.
UniDirectional (inherited from **TCustomDADataSet**)		Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TCustomMyDataSet**)		Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)		Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)		Used to check the status of the cached updates buffer.
Variables		Used to get or set the values of MySQL Server system variables.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
AnalyzeTable	Analyzes and stores the key distribution for the table
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CheckTable	Checks MyISAM and InnoDB tables for failures.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
CreateDatabase	Creates a new database with the specified name.
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.
DropDatabase	Drops the specified database.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.

Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TCustomDADataSet**)	Determines if a parameter with the specified name exists in a dataset.
Flush	Flushes server data from the memory to disk.
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.
GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.

GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetServiceNames	Populates a list of MySQL services at the server to which Connection is connected.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
KillProcess	Breaks the execution of the specified process.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.
OptimizeTable	Packs tables deleting unused places from the file at the disk and defragmenting it.
ParamByName (inherited from **TCustomDADataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshQuick (inherited from **TCustomMyDataSet**)	Retrieves changes posted to the server by another clients on the client side quickly.

RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RepairTable	Repairs specified tables.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
ServiceStart	Starts the specified service at the server to which Connection is connected.
ServiceStatus	Returns a status of the specified service at the server to which Connection is connected.
ServiceStop	Stops the specified service at the server to which Connection is connected.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
ShowProcessList	Shows the list of the processes.
ShowStatus	Shows the current state of the server.
ShowVariables	Shows the values of some MySQL system variables that are in effect for the current connection.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnLockTable (inherited from **TCustomMyDataSet**)	Releases a table locked by the TCustomMyDataSet.LockTable method.
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 **TCustomMyDataSet**)	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 **TCustomMyDataSet**)	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.

Request Support   DAC Forum   Provide Feedback

5.25.1.1.2 Properties

Properties of the **TMyServerControl** class.

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

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
CommandTimeout (inherited from **TCustomMyDataSet**)	Used to specify the amount of time to attempt execution of a command.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query

DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
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 **TCustomMyDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TCustomMyDataSet**)	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.
InsertId (inherited from **TCustomMyDataSet**)	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
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 (inherited from **TCustomMyDataSet**)	Specifies when to perform locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
Options (inherited from **TCustomMyDataSet**)	Specifies the behaviour of TCustomMyDataSet 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 **TCustomDADataSet**)	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or

	not.
[Ranged](#) (inherited from **[TMemDataSet](#)**)	Indicates whether a range is applied to a dataset.
[ReadOnly](#) (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.
[SmartFetch](#) (inherited from **[TCustomMyDataSet](#)**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
[SQL](#) (inherited from **[TCustomDADataSet](#)**)	Used to provide a SQL statement that a query component executes when its Open method is called.
[SQLDelete](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
[SQLInsert](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
[SQLLock](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to perform a record lock.
[SQLRecCount](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
[SQLRefresh](#) (inherited from **[TCustomDADataSet](#)**)	Used to specify a SQL statement that will be used to refresh current record by calling the [TCustomDADataSet.RefreshRecord](#) procedure.
[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 **TCustomMyDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.
Variables	Used to get or set the values of MySQL Server system variables.

Published

Name	Description
Connection	Used to specify a connection object to use for connecting to a data store.
Debug	Used to display executing statement.
TableNames	Holds a list of tables that will be processed by the TMyServerControl.AnalyzeTable, TMyServerControl.OptimizeTable, TMyServerControl.CheckTable, TMyServerControl.RepairTable methods.

See Also

- **TMyServerControl Class**

- TMyServerControl Class Members

Request Support   DAC Forum   Provide Feedback

5.25.1.1.2.1 Connection Property

Used to specify a connection object to use for connecting to a data store.

Class

TMyServerControl

Syntax

```
property Connection: TCustomMyConnection;
```

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 TMyConnection objects.

At run-time, set Connection property to reference an existing TMyConnection object.

See Also
- TCustomMyConnection

Request Support   DAC Forum   Provide Feedback

5.25.1.1.2.2 Debug Property

Used to display executing statement.

Class

TMyServerControl

Syntax

```
property Debug: boolean;
```

Remarks

Set the Debug property to True to display executing statement.

The default value is False.

Note: To enable debug form display you should explicitly include MyDacVcl unit into your project.

See Also
- TCustomDADataSet.Debug

　　　Request Support　　　DAC Forum　　　Provide Feedback

5.25.1.1.2.3 TableNames Property

Holds a list of tables that will be processed by the AnalyzeTable, OptimizeTable, CheckTable, RepairTable methods.

Class
`TMyServerControl`

Syntax
```
property TableNames: string;
```

Remarks

Contains a list of tables that will be processed by the AnalyzeTable, OptimizeTable, CheckTable, RepairTable methods.

Table names are separated by comma or semicolon. If it has an empty value, all tables presented in the database will be processed.

See Also
- AnalyzeTable
- OptimizeTable
- CheckTable
- RepairTable

- TCustomDAConnection.GetTableNames

Request Support   DAC Forum   Provide Feedback

5.25.1.1.2.4 Variables Property(Indexer)

Used to get or set the values of MySQL Server system variables.

Class

TMyServerControl

Syntax

```
property Variables[const VarName: string]: string;
```

Parameters

VarName
  Holds the name of MySQL server system variables.

Remarks

Use the Variables property to get or set the values of MySQL Server system variables. String values should be assigned with quotes.

Example

For example:

```
    MyServerControl1.Variables['max_allowed_packet'] := '1234567890';
but:
    MyServerControl.Variables['time_format'] := '''%H:%i:%s''';
```

Request Support   DAC Forum   Provide Feedback

5.25.1.1.3 Methods

Methods of the **TMyServerControl** class.

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

Public

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
AnalyzeTable	Analyzes and stores the key distribution for the table
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CheckTable	Checks MyISAM and InnoDB tables for failures.
CloneCursor (inherited from **TCustomDADataSet**)	Shares data from another dataset.
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.
CreateDatabase	Creates a new database with the specified name.
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.
DropDatabase	Drops the specified database.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing

	range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TCustomDADataSet**)	Determines if a parameter with the specified name exists in a dataset.
Flush	Flushes server data from the memory to disk.
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.

GetFieldEnum (inherited from **TCustomMyDataSet**)	Retrieve the list of acceptable values for a specified field given by the FieldName parameter.
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.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetServiceNames	Populates a list of MySQL services at the server to which Connection is connected.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
KillProcess	Breaks the execution of the specified process.
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 **TCustomMyDataSet**)	Overloaded. Locks the current record for the current connection.
LockTable (inherited from **TCustomMyDataSet**)	Locks table for the current connection.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TCustomMyDataSet**)	Opens next rowset in the statement.

[OptimizeTable](#)	Packs tables deleting unused places from the file at the disk and defragmenting it.
[ParamByName](#) (inherited from **[TCustomDADataSet](#)**)	Sets or uses parameter information for a specific parameter based on its name.
[Prepare](#) (inherited from **[TCustomDADataSet](#)**)	Allocates, opens, and parses cursor for a query.
[RefreshQuick](#) (inherited from **[TCustomMyDataSet](#)**)	Retrieves changes posted to the server by another clients on the client side quickly.
[RefreshRecord](#) (inherited from **[TCustomDADataSet](#)**)	Actualizes field values for the current record.
[RepairTable](#)	Repairs specified tables.
[RestoreSQL](#) (inherited from **[TCustomDADataSet](#)**)	Restores the SQL property modified by AddWhere and SetOrderBy.
[RestoreUpdates](#) (inherited from **[TMemDataSet](#)**)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#) (inherited from **[TMemDataSet](#)**)	Cancels changes made to the current record when cached updates are enabled.
[SaveSQL](#) (inherited from **[TCustomDADataSet](#)**)	Saves the SQL property value to BaseSQL.
[SaveToJSON](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
[SaveToXML](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[ServiceStart](#)	Starts the specified service at the server to which Connection is connected.
[ServiceStatus](#)	Returns a status of the specified service at the server to which Connection is connected.

[ServiceStop](#)	Stops the specified service at the server to which Connection is connected.
[SetOrderBy](#) (inherited from **[TCustomDADataSet](#)**)	Builds an ORDER BY clause of a SELECT statement.
[SetRange](#) (inherited from **[TMemDataSet](#)**)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
[ShowProcessList](#)	Shows the list of the processes.
[ShowStatus](#)	Shows the current state of the server.
[ShowVariables](#)	Shows the values of some MySQL system variables that are in effect for the current connection.
[SQLSaved](#) (inherited from **[TCustomDADataSet](#)**)	Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.
[UnLock](#) (inherited from **[TCustomDADataSet](#)**)	Releases a record lock.
[UnLockTable](#) (inherited from **[TCustomMyDataSet](#)**)	Releases a table locked by the [TCustomMyDataSet.LockTable](#) method.
[UnPrepare](#) (inherited from **[TMemDataSet](#)**)	Frees the resources allocated for a previously prepared query on the server and client sides.
[UpdateResult](#) (inherited from **[TMemDataSet](#)**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also
- TMyServerControl Class
- TMyServerControl Class Members

| Request Support | DAC Forum | Provide Feedback

5.25.1.1.3.1 AnalyzeTable Method

Analyzes and stores the key distribution for the table

Class

`TMyServerControl`

Syntax

```
procedure AnalyzeTable;
```

Remarks

Call the AnalyzeTable method to analyze and store the key distribution for the table. During the analysis, the table is locked with a read lock. This works on MyISAM and BDB tables.

This is equivalent to running *myisamchk -a* on the table.

If the table hasn't changed since the last AnalyzeTable command, the table will not be analysed again.

The list of the tables is used from TableNames.

Refer to the ANALYZE TABLE detailed description at MySQL Reference Manual.

See Also
- TableNames
- OptimizeTable

- [CheckTable](#)

- [RepairTable](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.25.1.1.3.2 CheckTable Method

Checks MyISAM and InnoDB tables for failures.

Class

`TMyServerControl`

Syntax

```
procedure CheckTable(CheckTypes: TMyCheckTypes = [ctMedium]);
```

Parameters

CheckTypes
  Holds the type of the checking that will be performed.

Remarks

Call the CheckTable method to check MyISAM and InnoDB tables for failures.

The list of the tables is used from [TableNames](#).

Refer to the CHECK TABLE detailed description at MySQL Reference Manual.

See Also
- [TableNames](#)

- [AnalyzeTable](#)

- [OptimizeTable](#)

- [RepairTable](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.25.1.1.3.3 CreateDatabase Method

Creates a new database with the specified name.

Class

TMyServerControl

Syntax

```
procedure CreateDatabase(DatabaseName: string; IfNotExists:
boolean = True; CharsetName: string = ''; CollationName: string =
'');
```

Parameters

DatabaseName
   Holds the name of the database that will be created.

IfNotExists
   True, if a database with the specified name does not exist.

CharsetName
   Holds the charset name.

CollationName
   Holds the collation name.

Remarks

Call the CreateDatabase method to create a new database with the specified name.

If IfNotExists is True, then a database will be created only if it did not exist.

CharsetName, CollationName parameters specify the language parameters of the created database.

See Also

- DropDatabase

Request Support   DAC Forum   Provide Feedback

5.25.1.1.3.4 DropDatabase Method

Drops the specified database.

Class

TMyServerControl

Syntax

```
procedure DropDatabase(DatabaseName: string; IfExists: boolean =
True);
```

Parameters

DatabaseName
  Holds the database name.

IfExists
  If a database with the specified name does not exist, an exception will be generated.

Remarks

Call the DropDatabase method to drop the specified database.

IfExists parameter is used to raise an error message if the specified database doesn't exist.

Note: Use this function very carefully!

See Also

- DropDatabase

© 1997-2025
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.25.1.1.3.5 Flush Method

Flushes server data from the memory to disk.

Class

TMyServerControl

Syntax

```
procedure Flush(FlushTypes: TMyFlushTypes);
```

Parameters

FlushTypes
  Holds the options that define the actions to take while performing flush operation.

Remarks

Call the Flush method to flush forcedly server data from the memory to disk. Used if backup copy is needed.

Refer to FLUSH detailed description at MySQL Reference Manual.

[Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.25.1.1.3.6 GetServiceNames Method

Populates a list of MySQL services at the server to which Connection is connected.

Class

[TMyServerControl](#)

Syntax

```
procedure GetServiceNames(List: TStrings);
```

Parameters

List
  Holds a list of MySQL services.

Remarks

Call the GetServiceNames property to populate a list of MySQL services at the server to which Connection is connected.

List.Objects[] fields are filled with the status of services in TMyServiceStatus format.

This method is available only for Windows.

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetServiceNames.

See Also

- [ServiceStart](#)

- ServiceStop

- ServiceStatus

Request Support   DAC Forum   Provide Feedback

5.25.1.1.3.7 KillProcess Method

Breaks the execution of the specified process.

Class

`TMyServerControl`

Syntax

```
procedure KillProcess(ThreadId: integer);
```

Parameters

ThreadId
  Holds the thread ID of the current connection.

Remarks

Call the KillProcess method to break the execution of the specified process.

Refer to KILL detailed description at MySQL Reference Manual.

See Also
- ShowProcessList

Request Support   DAC Forum   Provide Feedback

5.25.1.1.3.8 OptimizeTable Method

Packs tables deleting unused places from the file at the disk and defragmenting it.

Class

`TMyServerControl`

Syntax

```
procedure OptimizeTable;
```

Remarks

Call the OptimizeTable method to pack tables deleting unused places from the file at the disk and defragmenting it.

It makes sense to call it from time to time when working with a table actively, especially when deleting frequently.

The list of the tables is used from TableNames.

Refer to OPTIMIZE TABLE detailed description at MySQL Reference Manual.

See Also
- TableNames
- AnalyzeTable
- CheckTable
- RepairTable

Request Support   DAC Forum   Provide Feedback

5.25.1.1.3.9 RepairTable Method

Repairs specified tables.

Class

```
TMyServerControl
```

Syntax

```
procedure RepairTable(RepairTypes: TMyRepairTypes = []);
```

Parameters

RepairTypes
   Holds the options that define the actions to take while performing repair operation on a table.

Remarks

Call the RepairTable method to repair specified tables.

Works only with MyISAM tables.

List of the tables is used from TableNames.

Refer to REPAIR TABLE detailed description at MySQL Reference Manual.

See Also

- TableNames

- AnalyzeTable

- OptimizeTable

- CheckTable

Request Support   DAC Forum   Provide Feedback

5.25.1.1.3.10 ServiceStart Method

Starts the specified service at the server to which Connection is connected.

Class

TMyServerControl

Syntax

```
procedure ServiceStart(const ServiceName: string; ParamStr:
string = '');
```

Parameters

ServiceName
  Holds the name of the service to start.

ParamStr
  Holds the list of parameters with which service will be started.

Remarks

Starts the specified service at the server to which Connection with ParamStr parameters is connected.

This method is available only for Windows.

See Also

- [ServiceStop](#)

- [GetServiceNames](#)

- [ServiceStatus](#)

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.25.1.1.3.11 ServiceStatus Method

Returns a status of the specified service at the server to which Connection is connected.

Class

`TMyServerControl`

Syntax

```
function ServiceStatus(const ServiceName: string):
TMyServiceStatus;
```

Parameters

ServiceName
  Holds the name of the service to start.

Return Value

  a status of the specified service.

Remarks

Call the ServiceStatus method to return a status of the specified service at the server to which Connection is connected.

To get a list of MySQL services you should use [GetServiceNames](#) function.

This method is available only for Windows.

See Also

- [ServiceStart](#)

- [ServiceStop](#)

- [GetServiceNames](#)

5.25.1.1.3.12 ServiceStop Method

Stops the specified service at the server to which Connection is connected.

Class

[TMyServerControl](#)

Syntax

```
procedure ServiceStop(const ServiceName: string);
```

Parameters

ServiceName
   Holds the name of the service to stop.

Remarks

Call the ServiceStop method to stop the specified service at the server to which Connection is connected.

This method is available only for Windows.

See Also

- [ServiceStart](#)

- [GetServiceNames](#)

- [ServiceStatus](#)

5.25.1.1.3.13 Show ProcessList Method

Shows the list of the processes.

Class

[TMyServerControl](#)

Syntax

```
procedure ShowProcessList(Full: boolean = False);
```

Parameters

Full
   True, if full query text will be shown.

Remarks

Call the ShowProcessList method to show the list of the processes. Full parameter specifies whether to show the full text of the query or only the first 100 symbols.

To disconnect use the KillProcess method.

Refer to SHOW [FULL] PROCESSLIST detailed description at MySQL Reference Manual.

See Also

- KillProcess

Request Support DAC Forum Provide Feedback

5.25.1.1.3.14 Show Status Method

Shows the current state of the server.

Class

`TMyServerControl`

Syntax

```
procedure ShowStatus;
```

Remarks

Call the ShowStatus method to show the current state of the server.

See Also

- ShowVariables

Request Support DAC Forum Provide Feedback

Reserved.

5.25.1.1.3.15 Show Variables Method

Shows the values of some MySQL system variables that are in effect for the current connection.

Class

TMyServerControl

Syntax

```
procedure ShowVariables;
```

Remarks

Call the ShowVariables method to show the values of some MySQL system variables that are in effect for the current connection.

See Also

- **ShowStatus**

Request Support   **DAC Forum**   **Provide Feedback**

5.26 MySqlApi

This unit contains implementation of the class.

Types

Name	Description
TMyLogEvent	This type is used for the TMyEmbConnection.OnLog and TMyEmbConnection.OnLogError events.

Variables

Name	Description

	When set, specifies path and name of MySQL client library (libmysql.dll or libmysqld.dll for embedded server).
MySQLClientLibrary	

Request Support   DAC Forum   Provide Feedback

5.26.1 Types

Types in the **MySqlApi** unit.

Types

Name	Description
TMyLogEvent	This type is used for the TMyEmbConnection.OnLog and TMyEmbConnection.OnLogError events.

Request Support   DAC Forum   Provide Feedback

5.26.1.1 TMyLogEvent Procedure Reference

This type is used for the TMyEmbConnection.OnLog and TMyEmbConnection.OnLogError events.

Unit

MySqlApi

Syntax

```
TMyLogEvent = procedure (const Text: string) of object;
```

Parameters

Text
  Holds the text of error, startup messages, entries that record client connections, and SQL statements.

Request Support   DAC Forum   Provide Feedback

5.26.2 Variables

Variables in the **MySqlApi** unit.

Variables

Name	Description
MySQLClientLibrary	When set, specifies path and name of MySQL client library (libmysql.dll or libmysqld.dll for embedded server).

Request Support   **DAC Forum**   **Provide Feedback**

5.26.2.1 MySQLClientLibrary Variable

When set, specifies path and name of MySQL client library (libmysql.dll or libmysqld.dll for embedded server).

Unit

MySqlApi

Syntax

```
MySQLClientLibrary: string;
```

Request Support   **DAC Forum**   **Provide Feedback**

5.27 MySQLMonitor

This unit contains implementation of the TMySQLMonitor component.

Classes

Name	Description
TMySQLMonitor	This component serves for

	monitoring dynamic SQL execution in MyDAC-based applications.

Request Support   DAC Forum   Provide Feedback

5.27.1 Classes

Classes in the **MySQLMonitor** unit.

Classes

Name	Description
TMySQLMonitor	This component serves for monitoring dynamic SQL execution in MyDAC-based applications.

Request Support   DAC Forum   Provide Feedback

5.27.1.1 TMySQLMonitor Class

This component serves for monitoring dynamic SQL execution in MyDAC-based applications.

For a list of all members of this type, see TMySQLMonitor members.

Unit

MySQLMonitor

Syntax

```
TMySQLMonitor = class(TCustomMySQLMonitor);
```

Remarks

Use TMySQLMonitor to monitor dynamic SQL execution in MyDAC-based applications.TMySQLMonitor 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 TMySQLMonitor object. If an

application has no TMySQLMonitor instance, the Debug window is available to display SQL statements to be sent.

Inheritance Hierarchy

[TCustomDASQLMonitor](#)

  TCustomMySQLMonitor

    TMySQLMonitor

   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.27.1.1.1 Members

[TMySQLMonitor](#) class overview.

   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.28   VirtualDataSet

5.28.1   Classes

Classes in the **VirtualDataSet** unit.

Classes

Name	Description
[TCustomVirtualDataSet](#)	A base class for representation of arbitrary data in tabular form.

   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.28.1.1   TCustomVirtualDataSet Class

A base class for representation of arbitrary data in tabular form.

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

Unit

VirtualDataSet

Syntax

```
TCustomVirtualDataSet = class(TMemDataSet);
```

Inheritance Hierarchy

TMemDataSet

  TCustomVirtualDataSet

  Request Support   DAC Forum   Provide Feedback

5.28.1.1.1 Members

TCustomVirtualDataSet class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.

UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the

	TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToJSON (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when

	cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

5.28.2 Types

Types in the **VirtualDataSet** unit.

Types

Name	Description
TOnDeleteRecordEvent	This type is used for the E:Devart.VirtualDac.TVirtualDataSet.OnDeleteRecord event.
TOnGetFieldValueEvent	This type is used for the E:Devart.VirtualDac.TVirtualDataSet.OnGetFieldValue event.
TOnGetRecordCountEvent	This type is used for the E:Devart.VirtualDac.TVirtualDataSet.OnGetRecordCount event.
TOnModifyRecordEvent	This type is used for E:Devart.VirtualDac.TVirtualDataSet.OnInsertRecord and E:Devart.VirtualDac.TVirtualDataSet.OnModifyRecord events.

5.28.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the E:Devart.VirtualDac.TVirtualDataSet.OnDeleteRecord event.

Unit

```
VirtualDataSet
```

Syntax

```
TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo:
Integer) of object;
```

Parameters

Sender
   An object that raised the event.

RecNo
   Number of the record being deleted.

5.28.2.2 TOnGetFieldValueEvent Procedure Reference

This type is used for the E:Devart.VirtualDac.TVirtualDataSet.OnGetFieldValue event.

Unit

```
VirtualDataSet
```

Syntax

```
TOnGetFieldValueEvent = procedure (Sender: TObject; Field: TField;
RecNo: Integer; out Value: Variant) of object;
```

Parameters

Sender
   An object that raised the event.

Field
   The field, which data has to be returned.

RecNo
   The number of the record, which data has to be returned.

Value
   Requested field value.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.28.2.3 TOnGetRecordCountEvent Procedure Reference

This type is used for the E:Devart.VirtualDac.TVirtualDataSet.OnGetRecordCount event.

Unit

```
VirtualDataSet
```

Syntax

```
TOnGetRecordCountEvent = procedure (Sender: TObject; out Count:
Integer) of object;
```

Parameters

Sender
   An object that raised the event.

Count
   The number of records that the virtual dataset will contain.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.28.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for E:Devart.VirtualDac.TVirtualDataSet.OnInsertRecord and
E:Devart.VirtualDac.TVirtualDataSet.OnModifyRecord events.

Unit

```
VirtualDataSet
```

Syntax

```
TOnModifyRecordEvent = procedure (Sender: TObject; var RecNo:
Integer) of object;
```

Parameters

Sender

An object that raised the event.

RecNo

Number of the record being inserted or modified.

[Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)