Table of Contents

1 What's New

New Features in LiteDAC 6.2

- Added support for RAD Studio 12 Athens Release 1

- Added support for Lazarus 3.2

- Now the Direct mode is based on version 3.45.2 of the SQLite engine

- Added support for automatic detection of transaction state when the transaction is controlled by an SQL statement

New Features in LiteDAC 6.1

- Lazarus 3.0 is supported

- The RealAsDouble option for the Connection component is added

New Features in LiteDAC 6.0

- Now the Direct mode is based on version 3.42.0 of the SQLite engine

- Added support for RAD Studio 12

- Added support for macOS Sonoma

- Added support for iOS 17

- Added support for Android 13

- Added support for nested Macros in SQL queries

- Added support Display Format for Aggregate fields

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

New Features in LiteDAC 5.3

- 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 GUID fields stored in binary format

- Added support for the YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, GETDATE, DATE,

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

- Added support for the mathematical operations in TDADataSet.Filter

- Added support for Aggregate Fields and InternalCalc Fields

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

- Improved reading fields of the BLOB family

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

New Features in LiteDAC 5.2

- Now the Direct mode is based on version 3.39.2 of the SQLite engine

- 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 mapping table columns of non-standard types to fields of the ftMemo type

- Fixed bug with mapping table columns of integer types to string fields

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

New Features in LiteDAC 5.1

- RAD Studio 11 Alexandria Release 1 is supported

- Lazarus 2.2.0 is supported

- Windows 11 is supported

- macOS Monterey is supported

- Now the Direct mode is based on version 3.37.2 of the SQLite engine

New Features in LiteDAC 5.0

- RAD Studio 11 Alexandria is supported

- macOS ARM is supported

- Added demo project for FastReport FMX

- Added the IntegerAsLargeInt option for the Connection component

New Features in LiteDAC 4.4

- RAD Studio 10.4.2 Sydney is supported

- macOS 11 Big Sur is supported

- iOS 14 is supported

- Android 11 is supported

- The JournalMode option in the Connection component is added

- The LockingMode option in the Connection component is added

- The Synchronous option in the Connection component is added

- Performance with default values of the new options is significantly improved

- Performance of batch operations is improved

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

New Features in LiteDAC 4.3

- Lazarus 2.0.10 and FPC 3.2.0 are supported

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

New Features in LiteDAC 4.2

- RAD Studio 10.4 Sydney is supported

- Lazarus 2.0.8 is supported

- macOS 64-bit in Lazarus is supported

New Features in LiteDAC 4.1

- Android 64-bit is supported

- Lazarus 2.0.6 is supported

- Now Trial edition for macOS and Linux is fully functional

New Features in LiteDAC 4.0

- 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 LiteDAC 3.5

- Lazarus 2.0.2 is supported

- Now the Direct mode is based on the version 3.27.2 of the SQLite engine

- 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 LiteDAC 3.4

- RAD Studio 10.3 Rio is supported

- Support for the BreakExec method in the Query component is added

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

New Features in LiteDAC 3.3

- Lazarus 1.8.4 is supported

- Performance of batch operations is improved

- Demo projects for IntraWeb 14 are added

- WAL in the Direct Mode for non-Windows platforms is supported

New Features in LiteDAC 3.2

- Direct Mode in Lazarus is supported

- Lazarus 1.8 and FPC 3.0.4 are supported

- BIT type is supported

- Support for custom constraints is added

- The UseBlankValues property for the Loader component is added

- The TLiteDataSetOptions.UnknownAsString property that allows mapping fields of unknown type as ftString instead of ftMemo is added

- The TLiteDataSetOptions.AdvancedTypeDetection property that allows describing columns having data of different types is added

New Features in LiteDAC 3.1

- Now the Direct mode is based on the SQLite engine version 3.20.0

- Custom SQL aggregate functions are supported

New Features in LiteDAC 3.0

- 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

- Now the Direct mode is based on the SQLite engine version 3.17.0

New Features in LiteDAC 2.7

- 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

- Now the Direct mode is based on the SQLite engine version 3.12.0

- Support for URI filenames is added

- Data Type Mapping performance is improved

- Performance of TDALoader on loading data from TDataSet is improved

New Features in LiteDAC 2.6

- RAD Studio 10 Seattle is supported

- INSERT, UPDATE and DELETE batch operations are supported

- Now Trial for Win64 is a fully functional Professional Edition

- Now at automatic refresh of Detail dataset the OnBeforeOpen event is not called

- Now the Direct mode is based on the SQLite engine version 3.8.11.1

- The EnableSharedCache option of the Connection component for non-Windows platforms is added

New Features in LiteDAC 2.5

- RAD Studio XE8 is supported

- AppMethod is supported

- Direct mode for Mac OS X, iOS and Android platforms is supported

- Database encryption for Mac OS X, iOS and Android platforms is supported

- Now the Direct mode is based on the SQLite engine version 3.8.9

- The TLiteConnection.Options.ConnectMode property is added

- The TLiteConnection.ReleaseDatabaseMemory method is added

- The TLiteConnection.IsDatabaseReadOnly method is added

- Converter from liteNull data formats to ftExtended is added

New Features in LiteDAC 2.4

- RAD Studio XE7 is supported

- Lazarus 1.2.4 is supported

- Demo projects for FastReport 5 are added

- The TCustomDADataSet.GetKeyFieldNames method is added

- The ConstraintColumns metadata kind for the TDAMetadata component is added

- Now the Direct mode is based on the SQLite engine version 3.8.6

- TLiteBackup component is added

New Features in LiteDAC 2.3

- 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 TLiteDataSetOptions.MasterFieldsNullable property is added

- Now the Direct mode is based on the SQLite engine version 3.8.4.3

- The EnableLoadExtension option is added for the Connection component

- Now update queries inside TDataSet descendants have correct owner

New Features in LiteDAC 2.2

- iOS in C++Builder XE5 is supported

- Direct mode for x64 platform is supported

- Now the Direct mode is based on the SQLite engine version 3.8.2

- 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 LiteDAC 2.1

- RAD Studio XE5 is supported

- Application development for Android is supported

- Lazarus 1.0.12 is supported

- Performance is improved

- Automatic checking for new versions is added

- Flexible management of conditions in the WHERE clause is added

- The possibility to use conditions is added

- Now the Direct mode is based on the SQLite engine version 3.8.0.2

- The TLiteUserCollation component 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 Ping method for the Connection component is added

- The AllowImplicitConnect option for the Connection component is added

- The SQLRecCount property for the Query and StoredProc components is added

- The ScanParams property for the Script component is added

- The RowsAffected property for the Script component is added

- The 'True' Boolean value is now saved in the database as 1

- Conversion from the liteText type to all the supported types is added.

New Features in LiteDAC 2.0

- 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 of TimeStamp, Single and Extended fields in VirtualTable is added

- Now the Direct mode is based on the SQLite engine version 3.7.16.2

- Now SQLite string data type without length is mapped as ftMemo instead of ftString

- Converter from Unix and Julian date formats to ftDateTime is added

New Features in LiteDAC 1.6

- Rad Studio XE3 Update 1 is now required

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

- LastInsertId property for TLiteSQL and TLiteQuery is added

New Features in LiteDAC 1.5

- Rad Studio XE3 is supported

- Windows 8 is supported

- Now the Direct mode is based on the SQLite engine version 3.7.13

- Extended error codes support is added

- Components for FastReport 4.0 are added

New Features in SQLite Data Access Components 1.0.1

- First release of LiteDAC

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

2   General Information

This section contains general information about SQLite Data Access Components

- Overview

- Features

- Requirements

- Compatibility

- Using Several DAC Products in One IDE

- Component List

- Hierarchy Chart

- Editions

- Licensing and Subscriptions

- Getting Support

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

2.1   Overview

SQLite Data Access Components (LiteDAC) is a library of components that provides direct access to SQLite databases from Delphi, C++Builder and Lazarus (FPC). LiteDAC directly uses SQLite client software to connect to database. LiteDAC is designed to help programmers develop really lightweight, faster and cleaner SQLite database applications

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

Advantages of LiteDAC Technology

LiteDAC is a direct connectivity database wrapper built specifically for SQLite. LiteDAC offers wide coverage of the SQLite feature set and emphasizes optimized data access strategies.

Wide Coverage of SQLite Features

By providing access to the most advanced database functionality, LiteDAC allows developers to harness the full capabilities of SQLite and optimize their database applications. View the full list of supported SQLite features in Features.

Native Connection Options

LiteDAC provides direct access to SQLite databases without envolving SQLite client software that heightens its performance. LiteDAC-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.

Optimized Code

The goal of LiteDAC is to enable developers to write efficient and flexible database applications. The LiteDAC library is implemented using advanced data access algorithms and optimization techniques. Classes and components undergo comprehensive performance tests and are designed to help you write high-performance, lightweight data access layers.

Compatibility with other Connectivity Methods

The LiteDAC interface retains compatibility with standard VCL data access components, like BDE.

Development and Support

LiteDAC is a SQLite connectivity solution that is actively developed and supported. LiteDAC comes with full documentation, demo projects, and fast (usually within two business days) technical support by the LiteDAC development team. Find out more about getting help or submitting feedback and suggestions to LiteDAC Development Team in Getting Support.

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

Key Features

The following list describes the main features of SQLite Data Access Components.

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

- Full support of the latest versions of SQLite

- Support for all SQLite data types

- Disconnected Model with automatic connection control for working with data offline

- All types of local sorting and filtering, including by calculated and lookup fields

- Automatic data updating with TLiteQuery and TLiteTable components.

- Unicode and national charset support.

- Supports many SQLite-specific features, such as user-defined functions.

- Advanced script execution functionality with the TLiteScript component.

- Support for using macros in SQL

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

- Includes annual LiteDAC Subscription with Priority Support.

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

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

How does LiteDAC work?

LiteDAC connects to SQLite either directly or using the SQLite client software. LiteDAC works directly through the native SQLite interface, without using BDE or ODBC.

In contrast, the Borland Database Engine (BDE) uses several layers to access SQLite, and requires additional data access software to be installed on client machines.
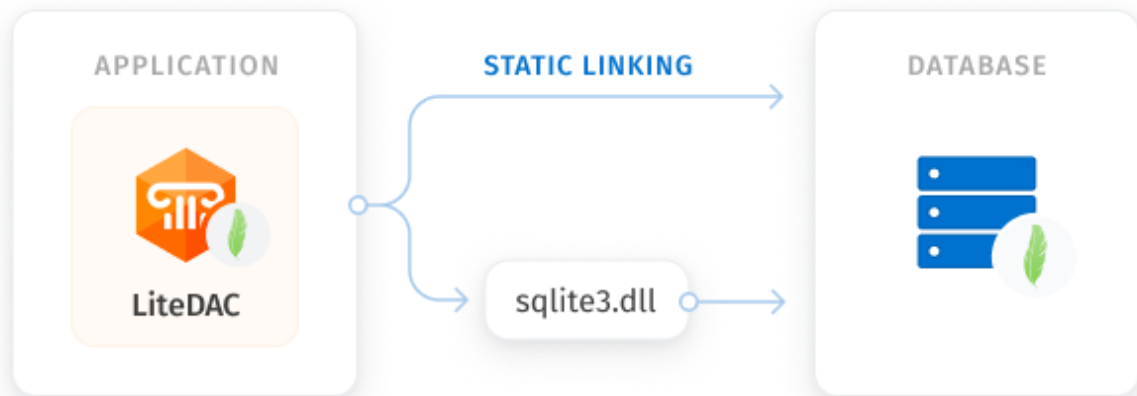
Request Support   DAC Forum   Provide Feedback

2.2   Features

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

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 database. 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, FMX, LCL development platforms are 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

- In Direct mode does not require the SQLite client software (it will be linked in an application statically) and works with a database directly

- Disconnected Model with automatic connection control for working with data offline

- Local Failover for detecting connection loss and implicitly reexecuting certain operations

Compatibility

- Full support of the latest versions of SQLite

- Direct mode support for Windows x32 and x64, MacOS, iOS and Android

- Support for all SQLite data types

- Support for Unix and Julian date and time formats

- Compatible with all IDE versions starting with Delphi 6, C++Builder 6, except Delphi 8, and with Free Pascal

- Includes provider for UniDAC Express Edition

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

- Support of all standard and third-party visual data-aware controls

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

SQLite technology support

- SQLite database encryption in Direct mode using different encryption algorithms

- Data Type Mapping

- Support for automatic database creation on connect

- Support for Shared-Cache mode

- Support for SQLite user-defined functions

- Support for SQLite user-defined collations

- Support for SQLite extensions loading

- Support for SQLite R*Tree module

- Support for SQLite FTS3 and FTS4 extensions

- Support for autoincrement fields

- Support for multi-SQL statements executing

- Fast record insertion with the TLiteLoader component

- Support for database backup using SQLite Online Backup API with the TLiteBackup component

Performance

- High overall performance

- Fast controlled fetch of large data blocks

- Optimized string data storing

- Advanced connection pooling

- High performance of applying 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

- Local master/detail relationship

- Master/detail relationship in CachedUpdates mode

Data access and data management automation

- Automatic data updating with TLiteQuery and TLiteTable components

- Automatic record refreshing

- Automatic query preparing

- Support for ftWideMemo field type in Delphi 2006 and higher

Extended data access functionality

- Data Encryption in a client application

- Separate component for executing SQL statements

- Simplified access to table data with TLiteTable component

- Ability to retrieve metadata information with TLiteMetaData component

- Support for using macros in SQL

- FmtBCD fields support

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

- Automatic retrieval of default field values

- Deferred detail DataSet refresh in master/detail relationships

- LiteDataAdapter component for WinForms and ASP.NET applications

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 TLiteDump components

Script execution

- Advanced script execution features with TLiteScript component

- Support for executing individual statements in scripts

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

SQL execution monitoring

- Extended SQL tracing capabilities provided by TLiteSQLMonitor 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

- More convenient data source setup with the TLiteDataSource 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 LiteDAC Subscription with Priority Support

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

Request Support   DAC Forum   Provide Feedback

2.3 Requirements

LiteDAC supports two different ways of work with SQLite databases. When the TLiteConnection.Options.Direct property is set to False, an application based on LiteDAC requires the SQLite client library. LiteDAC dynamically loads SQLite client DLL (sqlite3.dll) available on user systems. To locate DLL you can set the TIBCConnection.ClientLibrary property with the path to the client library. By default LiteDAC searches client DLL (sqlite3.dll) in directories specified in the PATH environment variable. When the TLiteConnection.Options.Direct property is set to True, then no additional files for the application are needed.

© 1997-2024 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

2.4 Compatibility

IDE Compatibility

LiteDAC is compatible with the following IDEs:

Embarcadero RAD Studio 12.1 Athens
- Embarcadero Delphi 12.1 Athens for Windows

- Embarcadero Delphi 12.1 Athens for macOS

- Embarcadero Delphi 12.1 Athens for Linux

- Embarcadero Delphi 12.1 Athens for iOS

- Embarcadero Delphi 12.1 Athens for Android

- Embarcadero C++Builder 12.1 Athens for Windows

- Embarcadero C++Builder 12.1 Athens for iOS

- Embarcadero C++Builder 12.1 Athens for Android

Embarcadero RAD Studio 12 Athens
- Embarcadero Delphi 12 Athens for Windows

- Embarcadero Delphi 12 Athens for macOS

- Embarcadero Delphi 12 Athens for Linux

- Embarcadero Delphi 12 Athens for iOS

- Embarcadero Delphi 12 Athens for Android

- Embarcadero C++Builder 12 Athens for Windows

- Embarcadero C++Builder 12 Athens for iOS

- Embarcadero C++Builder 12 Athens for Android

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

- Embarcadero Delphi 11.1 Alexandria for macOS

- Embarcadero Delphi 11.1 Alexandria for Linux

- Embarcadero Delphi 11.1 Alexandria for iOS

- Embarcadero Delphi 11.1 Alexandria for Android

- Embarcadero C++Builder 11.1 Alexandria for Windows

- Embarcadero C++Builder 11.1 Alexandria for iOS

- Embarcadero C++Builder 11.1 Alexandria for Android

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

- Embarcadero Delphi 10.4 Sydney for macOS

- Embarcadero Delphi 10.4 Sydney for Linux

- Embarcadero Delphi 10.4 Sydney for iOS

- Embarcadero Delphi 10.4 Sydney for Android

- Embarcadero C++Builder 10.4 Sydney for Windows

- Embarcadero C++Builder 10.4 Sydney for iOS

- Embarcadero C++Builder 10.4 Sydney for Android

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

- Embarcadero Delphi 10.3 Rio for macOS

- Embarcadero Delphi 10.3 Rio for Linux

- Embarcadero Delphi 10.3 Rio for iOS

- Embarcadero Delphi 10.3 Rio for Android

- Embarcadero C++Builder 10.3 Rio for Windows

- Embarcadero C++Builder 10.3 Rio for macOS

- Embarcadero C++Builder 10.3 Rio for iOS

- Embarcadero C++Builder 10.3 Rio for Android

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

- Embarcadero Delphi 10.2 Tokyo for macOS

- Embarcadero Delphi 10.2 Tokyo for Linux

- Embarcadero Delphi 10.2 Tokyo for iOS

- Embarcadero Delphi 10.2 Tokyo for Android

- Embarcadero C++Builder 10.2 Tokyo for Windows

- Embarcadero C++Builder 10.2 Tokyo for macOS

- Embarcadero C++Builder 10.2 Tokyo for iOS

- Embarcadero C++Builder 10.2 Tokyo for Android

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

- Embarcadero Delphi 10.1 Berlin for macOS

- Embarcadero Delphi 10.1 Berlin for iOS

- Embarcadero Delphi 10.1 Berlin for Android

- Embarcadero C++Builder 10.1 Berlin for Windows

- Embarcadero C++Builder 10.1 Berlin for macOS

- Embarcadero C++Builder 10.1 Berlin for iOS

- Embarcadero C++Builder 10.1 Berlin for Android

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

- Embarcadero Delphi 10 Seattle for macOS

- Embarcadero Delphi 10 Seattle for iOS

- Embarcadero Delphi 10 Seattle for Android

- Embarcadero C++Builder 10 Seattle for Windows

- Embarcadero C++Builder 10 Seattle for macOS

- Embarcadero C++Builder 10 Seattle for iOS

- Embarcadero C++Builder 10 Seattle for Android

Embarcadero RAD Studio XE8
- Embarcadero Delphi XE8 for Windows

- Embarcadero Delphi XE8 for macOS

- Embarcadero Delphi XE8 for iOS

- Embarcadero Delphi XE8 for Android

- Embarcadero C++Builder XE8 for Windows

- Embarcadero C++Builder XE8 for macOS

- Embarcadero C++Builder XE8 for iOS

- Embarcadero C++Builder XE8 for Android

Embarcadero RAD Studio XE7
- Embarcadero Delphi XE7 for Windows

- Embarcadero Delphi XE7 for macOS

- Embarcadero Delphi XE7 for iOS

- Embarcadero Delphi XE7 for Android

- Embarcadero C++Builder XE7 for Windows

- Embarcadero C++Builder XE7 for macOS

- Embarcadero C++Builder XE7 for iOS

- Embarcadero C++Builder XE7 for Android

Embarcadero RAD Studio XE6
- Embarcadero Delphi XE6 for Windows

- Embarcadero Delphi XE6 for macOS

- Embarcadero Delphi XE6 for iOS

- Embarcadero Delphi XE6 for Android

- Embarcadero C++Builder XE6 for Windows

- Embarcadero C++Builder XE6 for macOS

- Embarcadero C++Builder XE6 for iOS

- Embarcadero C++Builder XE6 for Android

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

- Embarcadero Delphi XE5 for macOS

- Embarcadero Delphi XE5 for iOS

- Embarcadero Delphi XE5 for Android

- Embarcadero C++Builder XE5 for Windows

- Embarcadero C++Builder XE5 for macOS

- Embarcadero C++Builder XE5 for iOS

Embarcadero RAD Studio XE4
- Embarcadero Delphi XE4 for Windows

- Embarcadero Delphi XE4 for macOS

- Embarcadero Delphi XE4 for iOS

- Embarcadero C++Builder XE4 for Windows

- Embarcadero C++Builder XE4 for macOS

Embarcadero RAD Studio XE3 (Requires Update 2)
- Embarcadero Delphi XE3 for Windows

- Embarcadero Delphi XE3 for macOS

- Embarcadero C++Builder XE3 for Windows

- Embarcadero C++Builder XE3 for macOS

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

- Embarcadero Delphi XE2 for macOS

- Embarcadero C++Builder XE2 for Windows

- Embarcadero C++Builder XE2 for macOS

Embarcadero RAD Studio XE
- Embarcadero Delphi XE

- Embarcadero C++Builder XE

Embarcadero RAD Studio 2010
- Embarcadero Delphi 2010

- Embarcadero C++Builder 2010

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

- CodeGear C++Builder 2009

CodeGear RAD Studio 2007
- CodeGear Delphi 2007

- CodeGear C++Builder 2007

Borland Developer Studio 2006
- Borland Delphi 2006

- Borland C++Builder 2006

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

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

Professional, Community, and Starter.

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

source code.

Supported Target Platforms
- Windows 32-bit and 64-bit

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

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

- iOS 64-bit

- iOS Simulator ARM 64-bit

- Android 32-bit and 64-bit

Support for Windows 64-bit is available since RAD Studio XE2. Support for iOS 64-bit is

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

Supported GUI Frameworks

- FireMonkey (FMX)

- Visual Component Library (VCL)

- Lazarus Component Library (LCL)

Devart Data Access Components Compatibility

All DAC products are compatible with each other.

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

© 1997-2024
Devart. All Rights
Reserved.

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 SQLite Data Access
Components library. Click on the name of each component for more information. These
components are added to the LiteDAC 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 LiteDAC
components are included in all LiteDAC editions. LiteDAC Professional Edition components
are not included in LiteDAC Standard Edition.

Basic LiteDAC components

	TLiteConnection	Represents an open connection to a SQLite database.
	TLiteQuery	Executes queries and Operates record sets. It also provides flexible way to update data.
	TLiteSQL	Executes SQL statements, which do not return rowsets.
	TLiteTable	Lets you retrieve and update data in a single table without writing SQL statements.
	TLiteUpdateSQL	Lets you tune update operations for the DataSet component.
	TLiteDataSource	Provides an interface between LiteDAC dataset components and data-aware controls on a form.
	TLiteScript	Executes sequences of SQL statements.
	TLiteSQLMonitor	Interface for monitoring dynamic SQL execution in LiteDAC-based applications.
	TLiteConnectDialog	Used to build custom prompts for the database name and encryption key.
	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.

LiteDAC Professional Edition components

	TLiteUserCollation	Provides functionality for working with user-defined collations
	TLiteUserFunction	Provides functionality to define custom functions for future use in SQL-statements
	TLiteLoader	Provides quick loading of external data into the database
	TLiteDump	Serves to store a database or its parts as a script and also to restore database from received script.
	TLiteBackup	Implements SQLite Online Backup API functionality
	TLiteMetaData	Retrieves metadata on specified SQL object.
	TLiteEncryptor	Represents data encryption and decryption in client application.
	TCRBatchMove	Retrieves metadata on database objects from the server.

See Also

- Hierarchy chart

Request Support   DAC Forum   Provide Feedback

2.7   Hierarchy Chart

Many LiteDAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for LiteDAC is shown below. The LiteDAC 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
|        |—TLiteConnection
|—TDataSet
|   |—TMemDataSet
|       |—TCustomDADataSet
|       |   |—TCustomLiteDataSet
|       |   |   |—TLiteQuery
|       |   |   |—TCustomLiteTable
|       |   |        |—TLiteTable
|       |—TDAMetaData
|       |     |—TLiteMetaData
|       |—TVirtualTable
|—TDataSource
|   |—TCRDataSource
|        |—TLiteDataSource
|—TCRBatchMove
|—TCustomConnectDialog
|   |—TLiteConnectDialog
|—TCustomLiteUserFunction
|   |—TLiteUserFunction
|—TCustomDASQL
|   |—TLiteSQL
|—TCustomDASQLMonitor
|   |—TLiteSQLMonitor
|—TDALoader
|   |—TLiteLoader
|—TDAScript
|   |—TLiteScript
|—TDADump
|   |—TLiteDump
|—TCREncryptor
    |—TLiteEncryptor
```

Request Support   DAC Forum   Provide Feedback

2.8   Editions

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

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

The **Professional** edition shows off the full power of LiteDAC, enhancing LiteDAC Standard Edition with support for SQLite-specific functionality, such as access to SQLite in the Direct mode using static linking of the SQLite library in an application, database encryption, and advanced dataset management features.

You can get **Source Access** to LiteDAC Professional Edition by purchasing a special LiteDAC Professional Edition with Source Code, which includes the source code of all component classes. The source code of DataSet Manager is not distributed. The source code of the Direct mode for SQLite is distributed as precompiled packages.

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

LiteDAC Edition Matrix

Feature	Standard	Professional
Direct Mode		
Direct access to SQLite by static linking of the SQLite library	✗	✓
Desktop Application Development		
Windows	✓	✓
macOS	✗	✓
Linux	✗	✓

Mobile Application Development		
iOS	✗	✓
Android	✗	✓
Database Encryption		
SQLite database encryption in Direct mode	✗	✓
Data Access Components		
Base Components: TLiteConnection TLiteQuery TLiteSQL TLiteTable TLiteUpdateSQL TLiteDataSource	✓	✓
Script executing TLiteScript	✗	✓
Fast data loading into the server TLiteLoader	✗	✓
SQLite Specific Components		
Working with user-defined collations TLiteUserCollation	✗	✓
Declaration and execution of user-defined functions TLiteUserFunction	✗	✓
Obtaining metadata about database objects TLiteMetaData	✗	✓
Storing a database as a script TLiteDump	✗	✓
Implements SQLite Online Backup API functionality TLiteBackup	✗	✓
DataBase Activity Monitoring		
Monitoring of per-component SQL execution TLiteSQLMonitor	✓	✓
Additional components		
Advanced connection dialog TLiteConnectDialog	✓	✓

Data encryption and decryption TLiteEncryptor	✕	✔
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	✔	✔
DataSet Manager	✕	✔
Cross IDE Support		
Lazarus and Free Pascal Support	✕	SRC[1]

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

Request Support   DAC Forum   Provide Feedback

2.9 Licensing

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

INTRODUCTION

This Devart end-user license agreement ("Agreement") is a legal agreement between you (either an individual person or a single legal entity) and Devart, for the use of LiteDAC 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 LiteDAC 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 LiteDAC. You can distribute LiteDAC 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.

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

2.10   Getting Support

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

Support Options

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

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

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

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

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

LiteDAC Priority Support

LiteDAC Priority Support is an advanced product support service for getting expedited

individual assistance with LiteDAC-related questions from the LiteDAC 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 LiteDAC Subscription.

To get help through the LiteDAC 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 LiteDAC Registration number.

- Full LiteDAC edition name and version number. You can find both of these from the LiteDAC | LiteDAC About menu in the IDE.

- Versions of the SQLite and client you are using.

- A detailed problem description.

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

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

2.11   Frequently Asked Questions

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

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

Installation and Deployment
1. I am having a problem installing LiteDAC or compiling LiteDAC-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 LiteDAC 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 LiteDAC, IBDAC, SDAC, ODAC, PgDAC or MyDAC 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 LiteDAC-based applications to work?

The minimal configuration of client installation includes the following:

- Copy the SQLite client library file *sqlite3.dll* to the folder available for executable unit of your program. For example, to the folder with your executable file, or to the Windows system folder. For more information, see description of the *LoadLibrary* function and the environment variable *PATH*.

Licensing and Subscriptions

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

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

2. Can I create components using LiteDAC?

You can create your own components that are inherited from LiteDAC or that use the LiteDAC 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. I have a registered version of LiteDAC. Will I need to pay to upgrade to future versions?

All upgrades to future versions are free to users with an active LiteDAC Subscription.

4. What are the benefits of the LiteDAC Subscription Program?

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

Users with a valid LiteDAC Subscription get the following benefits:

- Access to new versions of LiteDAC when they are released

- Access to all LiteDAC updates and bug fixes

- Product support through the LiteDAC Priority Support program

- Notification of new product versions

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

The LiteDAC Subscription Program is available for registered users of LiteDAC.

5. Can I use my version of LiteDAC after my Subscription expires?

Yes, you can. LiteDAC version licenses are perpetual.

6. I want a LiteDAC Subscription! How can I get one?

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

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

How To
1. How can I determine which version of LiteDAC I am using?

You can determine your LiteDAC version number in several ways:

- During installation of LiteDAC, consult the LiteDAC Installer screen.

- After installation, see the *history.html* file in your LiteDAC installation directiory.

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

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

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

- Fill the SQLInsert property with the parametrized query template you want to use.

- Call Insert.

- Initialize field values of the row to insert.

- Call Post.

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

3. Some questions about the visual part of LiteDAC

The following questions usually arise from the same problem:

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

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

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

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

General Questions

1. I would like to develop an application that works with SQLite databases. Which should I use - LiteDAC or dbExpress?

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.

LiteDAC is a specialized set of components to access SQLite databases with advanced design-time and component interface similar to BDE.

We tried to implement maximal SQLite support in LiteDAC. dbExpress technology puts severe restrictions. For example, Unicode fields cannot be passed from the driver to dbExpress.

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 InterBase, it is probably better to use dbExpress. In other cases, especially when migrating from BDE or ADO, you should use LiteDAC.

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

3. I cannot use INT64 fields as key fields in master-detail relationship.

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 (see the protected method TLargeintField.SetVarValue in the DB unit for details). To avoid this problem, you can change the field type to INTEGER, which is usually sufficient for key fields. Alternatively, you can avoid using Value.

4. Can LiteDAC and BDE functions be used side-by-side in a single application?

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

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

3   Getting Started

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

- [What is LiteDAC?](#)

- [Installing LiteDAC](#).

- [Working with the LiteDAC demo projects](#).

- [Compiling and deploying your LiteDAC project](#).

- [Using the LiteDAC documentation](#).

- [How to get help with LiteDAC](#).

What is LiteDAC?

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

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

An introduction to LiteDAC is provided in the [Overview](#) section.

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

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

Installing LiteDAC

To install LiteDAC, complete the following steps.

1. Choose and download the version of the LiteDAC installation program that is compatible with your IDE. For instance, if you are installing LiteDAC 1.00, you should use the following files:

   For BDS 2006 and Turbo - **litedac100d10*.exe**
   For Delphi 7 - **litedac100d7*.exe**

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

2. Close all running Borland applications.

3. Launch the LiteDAC installation program you downloaded in the first step and follow the instructions to install LiteDAC.

By default, the LiteDAC installation program should install compiled LiteDAC libraries automatically on all IDEs.

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

If you have bought LiteDAC Professional Edition with Source Code, you will be able to download both the compiled version of LiteDAC and the LiteDAC source code. The installation process for the compiled version is standard, as described above.The LiteDAC source code must be compiled and installed manually. Consult the supplied *ReadmeSrc.html* file for more details.

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

Working with the LiteDAC demo projects

The LiteDAC installation package includes a number of demo projects that demonstrate LiteDAC capabilities and use patterns. The LiteDAC demo projects are automatically installed in the LiteDAC installation folder.

To quickly get started working with LiteDAC, launch and explore the introductory LiteDAC

demo project, *LiteDACDemo*, from your IDE. This demo project is a collection of demos that show how LiteDAC can be used. The project creates a form which contains an explorer panel for browsing the included demos and a view panel for launching and viewing the selected demo.

LiteDACDemo Walkthrough

1. Launch your IDE.

2. Choose File | Open Project from the menu bar

3. Find the LiteDAC directory and open the *LiteDACDemo* project. This project should be located in the Demos\LiteDACDemo folder.

For example, if you are using Borland Developer Studio 2006, the demo project may be found at

\Program Files\Devart\LiteDAC for Delphi 2006\Demos\Win32\LiteDACDemo \LiteDACDemo.bdsproj

4. Select Run | Run or press F9 to compile and launch the demo project. *LiteDACDemo* should start, and a full-screen LiteDAC Demo window with a toolbar, an explorer panel, and a view panel will open. The explorer panel will contain the list of all demo sub-projects included in *LiteDACDemo,* and the view panel will contain an overview of each included demo.

At this point, you will be able to browse through the available demos, read their descriptions, view their source code, and see the functionality provided by each demo for interacting with SQLite. However, you will not be able to actually retrieve data from SQLite or execute commands until you connect to the database.

5. Click on the "Connect" button on the *LiteDACDemo* toolbar. A Connect dialog box will open. Enter the connection parameters you use to connect to your SQLite database and click "Connect" in the dialog box.

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

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

6. Click on the "Create" button to create all objects that will be used by *LiteDACDemo*. If some of these objects already exist in the database you have connected to, the following error message will appear.

An error has occurred:
#42S01Table 'dept' already exists
You can manually create objects required for demo by using the following file: %LiteDAC%
\Demos\InstallDemoObjects.sql
%LiteDAC% is the LiteDAC installation path on your computer.
Ignore this exception?

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

7. Choose a demo that demonstrates an aspect of working with SQLite that you are interested in, and play with the demo frame in the view window on the right. For example, to find out more about how to work with SQLite tables, select the Table demo from the "Working with Components" folder. A simple SQLite table browser will open in the view panel which will let you open a table in your database by specifying its name and clicking on the Open button.

8. Click on the "Demo source" button in the *LiteDACDemo* toolbar to find out how the demo you selected was implemented. The source code behind the demo project will appear in the view panel. Try to find the places where LiteDAC components are used to connect to the database.

9. Click on the "Form as text" button in the *LiteDACDemo* toolbar to view the code behind the interface to the demo. Try to find the places where LiteDAC components are created on the demo form.

10. Repeat these steps for other demos listed in the explorer window. The available demos are organized in three folders.

Working with components

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

General demos

A collection of projects that show off the LiteDAC technology and demonstrate some ways of working with data.

SQLite-specific demos

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

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

Other LiteDAC demo projects

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

Compiling and deploying your LiteDAC project

Compiling LiteDAC-based projects

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

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

For more information about which library files and environment changes are needed for compiling LiteDAC-based projects, consult the Installation topic.

Deploying LiteDAC-based projects

To deploy an application that uses LiteDAC, you will need to make sure the target workstation has access to the following files.

- The SQLite client library, if connecting using SQLite client.

- The LiteDAC bpl files, if compiling with runtime packages.

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

A list of the files which may need to be deployed with LiteDAC-based applications is included in the Deployment topic.

Using the LiteDAC documentation

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

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

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

How to get help with LiteDAC

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

- If you have a question about LiteDAC installation or licensing, consult the Licensing and FAQ sections.

- You can get community assistance and LiteDAC technical support on the LiteDAC Support Forum.

- To get help through the LiteDAC Priority Support program, send an email to the LiteDAC development team at litedac@devart.com.

- If you have a question about ordering LiteDAC or any other Devart product, contact

  sales@devart.com.

For more information, consult the Getting Support topic.

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

3.1   Installation

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

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

Before installing LiteDAC ...

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

So before installing a new version of LiteDAC, uninstall all previous versions of LiteDAC 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 litedac@devart.com

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

Installed packages

> **Note:** %LiteDAC% denotes the path to your LiteDAC 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
litedacXX.bpl	LiteDAC run-time package	Windows\System32
dcllitedacXX.bpl	LiteDAC design-time package	Delphi\Bin
litedacvclXX.bpl	VCL support package	Delphi\Bin
crcontrolsXX.bpl	TCRDBGrid component	Delphi\Bin

Additional packages for using LiteDAC managers and wizards

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

Environment Changes

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

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

Delphi

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

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

C++Builder

C++Builder 6:

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

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

C++Builder 2006, 2007:

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

Lazarus

The LiteDAC installation program only copies LiteDAC files. You need to install LiteDAC packages to the Lazarus IDE manually. Open %LiteDAC%\Source\Lazarus1\dcllitedac10.lpk (for Trial version %LiteDAC%\Packages\dcllitedac10.lpk) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilded with LiteDAC packages.

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

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

DBMonitor

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

Request Support   DAC Forum   Provide Feedback

3.2 Connecting To SQLite Database

This tutorial describes how to connect to SQLite Database.

Requirements

LiteDAC supports 2 modes of working with SQLite database:

1. Direct Mode - no additional client libraries required;

2. Using sqlite3.dll(.o,.dylib) client library - requires a corresponding client library on a particular PC or device. The library must be located either in the application folder or in the folder specified in the environmental variables.

General information

SQLite works with 3 database types: real DB file, temp DB file, and DB in memory.

To use different DB types, a corresponding value must be set in the TLiteConnection.Database proeprty. To work with a DB file,you have to specify the full, relative or UNC path to the DB file. To work with a database in memory, the ':memory:' value must be set. To work with a temporary database, the property value should be empty.

To switch modes of work with the database, the TLiteConnectionOptions.Direct property is used. The property is set to False by default (client library mode). To use the Direct mode, the proeprty must be set to True.

When connecting to a database file, if it doesn't exist, the file can be created automatically. For this, the TLiteConnectionOptions.ForceCreateDatabase property must be set to True. Its

default value is False. On an attempt to connect to a non-existing DB file with disabled ForceCreateDatabase, an error message will be displayed saying that such a file doesn't exist.

Note: this option doesn't apply to working with a temp database and an in-memory database.

Note: when working with client library, its bitness must match the application bitness, i.e. a 32-bit application can work with a 32-bit library version only, and 64-bit - only with 64-bit.

Creating connection

Design time creation

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

1. Open the Component palette and find the TLiteConnection component in the LiteDAC category.

2. Double-click the component.

Note: a new object appears on the form. If this is the first time you create TLiteConnection in this application, it is named LiteConnection1.

After you have done these steps, you should set up the newly created LiteConnection1 component. You can do this in two ways:

Using TLiteConnection Editor

1. Double-click on the LiteConnection1 object.

2. In the Database edit box specify the database name (for example, test.db3). If Database is not specified, the temporary database is used.

3. If a client library with suppport for encryption or Direct Mode is used, then you can specify the key to the database file in the Encryption Key edit box.

4. If Direct Mode is used, then you can select an encryption algorithm in the Encription Algorithm ComboBox. When using a client library, the standard algorithm built into the library is used.

5. You can specify a particular SQLite3 library in the Client Library edit box.

6. To enable Direct Mode, the Direct CheckBox must be checked.

Using Object Inspector

1. Click on the LiteConnection1 object and press F11 to focus on object's properties.

2. In the Database property specify the database name (for example,test.db3). If Database is not specified, the temporary database is used.

3. If a client library with suppport for encryption or Direct Mode is used, then you can specify the key to the database file in the EncryptionKey poperty.

4. If Direct Mode is used, then you can select an encryption algorithm in the Options.EncriptionAlgorithm proeprty. When using a client library, the standard algorithm built into the library is used.

5. You can specify a particular SQLite3 library in the ClientLibrary property.

6. To enable Direct Mode, the Options.Direct property must be set to True.

Run time creation

Same operations performed in runtime look as follows:

[Delphi]

```
var
  LiteConnection: TLiteConnection;
begin
  LiteConnection := TLiteConnection.Create(nil);
  try
    LiteConnection.Database := 'test.db3';
    LiteConnection.Options.ForceCreateDatabase := True;
    LiteConnection.Options.Direct := True;
    LiteConnection.EncryptionKey := '123';
    LiteConnection.Options.EncryptionAlgorithm := leAES256;
    LiteConnection.LoginPrompt := False; //to prevent showing of the connect
    LiteConnection.Connect;
  finally
    LiteConnection.Free;
  end;
end.
```

Note: To run this code, you have to add the LiteAccess and LiteCall units to the USES clause of your unit.

[C++Builder]

```
{
```

```
  TLiteConnection* LiteConnection = new TLiteConnection(NULL);
  try
  {
    LiteConnection->Database = "test.db3";
    LiteConnection->Options->ForceCreateDatabase = True;
    LiteConnection->Options->Direct = True;
    LiteConnection->EncryptionKey = '123';
    LiteConnection->Options->EncryptionAlgorithm = leAES256;
    LiteConnection->LoginPrompt = False; //to prevent showing of the connect
    LiteConnection->Connect();
  }
  __finally
  {
    LiteConnection->Free();
  }
}
```

Note: To run this code, you have to include the LiteAccess.hpp and LiteCall.hpp header files
to your unit.

And using the ConnectString property:

[Delphi]

```
var
  LiteConnection: TLiteConnection;
begin
  LiteConnection := TLiteConnection.Create(nil);
  try
    LiteConnection.ConnectString := 'Database=test.db3;ForceCreateDatabase=T
    LiteConnection.Connect;
  finally
    LiteConnection.Free;
  end;
end.
```

Note: To run this code, you have to add the LiteAccess and LiteCall units to the USES clause
of your unit.

[C++Builder]

```
{
  TLiteConnection* LiteConnection = new TLiteConnection(NULL);
  try
  {
    LiteConnection->ConnectString = "Database=test.db3;ForceCreateDatabase=T
    LiteConnection->Connect();
  }
  __finally
  {
    LiteConnection->Free();
  }
}
```

> **Note:** To run this code, you have to include the LiteAccess.hpp and LiteCall.hpp header files to your unit.

Opening connection

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

[Delphi]

```
LiteConnection.Connect;
```

[C++Builder]

```
LiteConnection->Connect();
```

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

[Delphi]

```
LiteConnection.Connected := True;
```

[C++Builder]

```
LiteConnection->Connected = True;
```

This way can be used at design-time as well. Of course, LiteConnection1 must have valid connection options assigned earlier. When you call Connect, LiteDAC tries to find the database file and connect to it. If any problem occurs, it raises an exception with brief explanation on what is wrong. If no problem is encountered, LiteDAC tries to establish the connection. Finally, when connection is established, the Connect method returns and the Connected property is changed to True.

Closing connection

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

[Delphi]

```
LiteConnection.Close;
```

[C++Builder]

```
LiteConnection.Close();
```

, or

[Delphi]

```
LiteConnection.Connected := False;
```

[C++Builder]

```
LiteConnection.Connected = False;
```

Modifying connection

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

Request Support   DAC Forum   Provide Feedback

3.3 Creating Database Objects

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

Requirements

In order to create database objects, you have to connect to SQLite DB. This process is described in details in the tutorial Connecting To SQLite Database.

General information

Database objects are created using Data Definition Language (DDL), which is a part of SQL. There are two ways to create database objects. You can build DDL statements manually and execute them using the component like TLiteSQL. Another way is to use console utility sqlite3.exe This topic covers the first way - using components.

There are two ways of executing DDL statements in components like TLiteSQL, in design-time and in run-time. Both these ways are described below.

Note: the following assumes that you have the IDE running, you are currently focused on the form designer, and you have already set up the TLiteConnection on the form.

Creating tables

To create tables, the [TLiteSQL](#) component is used here.

Design time creation

- Open the Component palette and find the TLiteSQL component in the LiteDAC.

- Double-click on the component. Note that new object appears on the form. If this is the first

  time you create TLiteSQL in this application,

it is named LiteSQL1. Note that the LiteSQL1.Connection property is already set to existent

(on the form) connection.

- Double-click on the LiteSQL1 object.

- Type the following lines:

```
CREATE TABLE dept (
  deptno INTEGER PRIMARY KEY,
  dname VARCHAR2(14),
  loc VARCHAR2(13)
);
CREATE TABLE emp (
  empno INTEGER PRIMARY KEY,
  ename VARCHAR2(10),
  job VARCHAR2(9),
  mgr INTEGER,
  hiredate DATE,
  sal FLOAT,
  comm FLOAT,
  deptno INTEGER
);
```

- Press the Execute button. This will create two tables we will use for tutorial purposes.

Run time creation

The same operations performed in runtime look as follows:

[Delphi]

```
var
  LiteSQL: TLiteSQL;
begin
  LiteSQL:= TLiteSQL.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteSQL.Connection := LiteConnection;
    // set SQL script for creating tables
```

```
      LiteSQL.SQL.Clear;
      LiteSQL.SQL.Add('CREATE TABLE dept (');
      LiteSQL.SQL.Add('   deptno INTEGER,');
      LiteSQL.SQL.Add('   dname VARCHAR2(14),');
      LiteSQL.SQL.Add('   loc VARCHAR2(13)');
      LiteSQL.SQL.Add(');');
      LiteSQL.SQL.Add('CREATE TABLE emp (');
      LiteSQL.SQL.Add('   empno INTEGER PRIMARY KEY,');
      LiteSQL.SQL.Add('   ename VARCHAR2(10),');
      LiteSQL.SQL.Add('   job VARCHAR2(9),');
      LiteSQL.SQL.Add('   mgr INTEGER,');
      LiteSQL.SQL.Add('   hiredate DATE,');
      LiteSQL.SQL.Add('   sal FLOAT,');
      LiteSQL.SQL.Add('   comm FLOAT,');
      LiteSQL.SQL.Add('   deptno INTEGER');
      LiteSQL.SQL.Add(');');
      // execute script
      LiteSQL.Execute;
   finally
      LiteSQL.Free;
   end;
end;
```

[C++Builder]

```
{
  TLiteSQL* LiteSQL= new TLiteSQL(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteSQL->Connection = LiteConnection;
    // set SQL script for creating tables
    LiteSQL->SQL->Clear();
    LiteSQL->SQL->Add("CREATE TABLE dept (");
    LiteSQL->SQL->Add("   deptno INTEGER PRIMARY KEY,");
    LiteSQL->SQL->Add("   dname VARCHAR2(14),");
    LiteSQL->SQL->Add("   loc VARCHAR2(13)");
    LiteSQL->SQL->Add(");");
    LiteSQL->SQL->Add("CREATE TABLE emp (");
    LiteSQL->SQL->Add("   empno INTEGER PRIMARY KEY,");
    LiteSQL->SQL->Add("   ename VARCHAR2(10),");
    LiteSQL->SQL->Add("   job VARCHAR2(9),");
    LiteSQL->SQL->Add("   mgr INTEGER,");
    LiteSQL->SQL->Add("   hiredate DATE,");
    LiteSQL->SQL->Add("   sal FLOAT,");
    LiteSQL->SQL->Add("   comm FLOAT,");
    LiteSQL->SQL->Add("   deptno INTEGER");
    LiteSQL->SQL->Add(");");
    // execute script
    LiteSQL->Execute();
  }
  __finally
  {
    LiteSQL->Free();
  }
}
```

Additional Information

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

Request Support   DAC Forum   Provide Feedback

3.4   Deleting Data

This tutorial describes how to delete data from tables using the TLiteQuery and TLiteTable components.

Requirements

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

General information

Data in the database can be deleted using Data Manipulation Language (DML), which is a part of SQL. There are two ways to manipulate a database. You can build DML statements manually and run them within some component like TLiteQuery. Another way is to use the dataset functionality (the Delete method) of the TLiteQuery and TLiteTable 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 TLiteQuery and TLiteTable components allows deleting data without using DML statements. DML statements are generated by LiteDAC components internally. The code below demonstrates using this method:

[Delphi]

```
var
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // retrieve data
    LiteQuery.SQL.Text := 'SELECT * FROM dept';
    LiteQuery.Open;
    // delete the current record
    LiteQuery.Delete;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteQuery->Connection = LiteConnection;
    // retrieve data
    LiteQuery->SQL->Text = "SELECT * FROM dept";
    LiteQuery->Open();
    // delete the current record
    LiteQuery->Delete();
  }
  __finally
  {
    LiteQuery->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
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
```

```
    LiteQuery.Connection := LiteConnection;
    // set SQL query for delete record
    LiteQuery.SQL.Clear;
    LiteQuery.SQL.Add('DELETE FROM dept WHERE deptno = :deptno;');
    // set parameters
    LiteQuery.ParamByName('deptno').AsInteger := 10;
    // execute query
    LiteQuery.Execute;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

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

DML Statements As Plain Text

[Delphi]

```
var
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // set SQL query for delete record
    LiteQuery.SQL.Clear;
    LiteQuery.SQL.Add('DELETE FROM dept WHERE deptno = 10;');
    // execute query
    LiteQuery.Execute;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // con is either TLiteConnection already set up
    LiteQuery->Connection = con;
    // set SQL query for delete record
    LiteQuery->SQL->Clear();
    LiteQuery->SQL->Add("DELETE FROM dept WHERE deptno = 10;");
    // execute query
    LiteQuery->Execute();
  }
  __finally
  {
    LiteQuery->Free();
  }
}
```

Request Support   DAC Forum   Provide Feedback

3.5 Inserting Data Into Tables

This tutorial describes how to insert data into tables using the TLiteQuery and TLiteTable
components.

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To
SQLite Database") and that necessary objects are already created in the database (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 TLiteQuery. Another way is to use the
dataset functionality (the Insert, Append, and Post methods) of the TLiteQuery and TLiteTable
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	ANAL YST	7566	03.12. 1981	3000	NULL	20
MILLE R	CLER K	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 TLiteQuery component in the LiteDAC category.

- Double-click on the component. Note that a new object appears on the form. If this is the first time you create TLiteQuery in this application, it is named MSQuery1. Note that the LiteQuery1.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 TLiteQuery and TLiteTable components allow inserting data not using DML statements. DML statements are generated by LiteDAC 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
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
```

```
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // retrieve data
    LiteQuery.SQL.Text := 'SELECT * FROM dept';
    LiteQuery.Open;
    // append record
    LiteQuery.Append;
    LiteQuery.FieldByName('deptno').AsInteger := 10;
    LiteQuery.FieldByName('dname').AsString := 'ACCOUNTING';
    LiteQuery.FieldByName('loc').AsString := 'NEW YORK';
    LiteQuery.Post;
    // insert record
    LiteQuery.Insert;
    LiteQuery.FieldByName('deptno').AsInteger := 20;
    LiteQuery.FieldByName('dname').AsString := 'RESEARCH';
    LiteQuery.FieldByName('loc').AsString := 'DALLAS';
    LiteQuery.Post;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteQuery->Connection = LiteConnection;
    // retrieve data
    LiteQuery->SQL->Text = "SELECT * FROM dept";
    LiteQuery->Open();
    // append record
    LiteQuery->Append();
    LiteQuery->FieldByName("deptno")->AsInteger = 10;
    LiteQuery->FieldByName("dname")->AsString = "ACCOUNTING";
    LiteQuery->FieldByName("loc")->AsString = "NEW YORK";
    LiteQuery->Post();
    // insert record
    LiteQuery->Insert();
    LiteQuery->FieldByName("deptno")->AsInteger = 20;
    LiteQuery->FieldByName("dname")->AsString = "RESEARCH";
    LiteQuery->FieldByName("loc")->AsString = "DALLAS";
    LiteQuery->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
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // set SQL query for insert record
    LiteQuery.SQL.Clear;
    LiteQuery.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (:deptno,
    // set parameters
    LiteQuery.ParamByName('deptno').AsInteger := 10;
    LiteQuery.ParamByName('dname').AsString := 'ACCOUNTING';
    LiteQuery.ParamByName('loc').AsString := 'NEW YORK';
    // execute query
    LiteQuery.Execute;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteQuery->Connection = LiteConnection;
    // set SQL query for insert record
    LiteQuery->SQL->Clear();
    LiteQuery->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (:deptn
    // set parameters
    LiteQuery->ParamByName("deptno")->AsInteger = 10;
    LiteQuery->ParamByName("dname")->AsString = "ACCOUNTING";
    LiteQuery->ParamByName("loc")->AsString = "NEW YORK";
    // execute query
    LiteQuery->Execute();
  }
  __finally
  {
    LiteQuery->Free();
  }
}
```

DML Statements As Plain Text

[Delphi]

```
vvar
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // set SQL query for insert record
    LiteQuery.SQL.Clear;
    LiteQuery.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (10,''ACC
    // execute query
    LiteQuery.Execute;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteQuery->Connection = LiteConnection; // con is either TLiteConnection
    // set SQL query for insert record
    LiteQuery->SQL->Clear();
    LiteQuery->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (10,'AC
    // execute query
    LiteQuery->Execute();
  }
  __finally
  {
    LiteQuery->Free();
  }
}
```

Additional Information

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

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

3.6 Retrieving Data

This tutorial describes how to retrieve data from tables using the TLiteQuery and TLiteTable components.

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To SQLite Database"), 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. LiteDAC provides several components that can be used for data retrieving, such as TLiteQuery and TLiteTable. We will discuss data retrieving using these components.

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

TLiteQuery

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

[Delphi]

```
var
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // retrieve data
    LiteQuery.SQL.Text := 'SELECT * FROM dept';
    LiteQuery.Open;
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(LiteQuery.RecordCount));
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

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

TMyTable

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

[Delphi]

```
var
  LiteTable: TLiteTable;
begin
  LiteTable := TLiteTable.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteTable.Connection := LiteConnection;
    // retrieve data
    LiteTable.TableName := 'dept';
    LiteTable.Open;
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(LiteTable.RecordCount));
  finally
    LiteTable.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteTable* LiteTable = new TLiteTable(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteTable->Connection = LiteConnection;
    // retrieve data
```

```
    LiteTable->TableName = "dept";
    LiteTable->Open();
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(LiteTable->RecordCount));
  }
  __finally
  {
    LiteTable->Free();
  }
}
```

Request Support   DAC Forum   Provide Feedback

3.7   Modifying Data

This tutorial describes how to modify data in tables using the TLiteQuery and TLiteTable
components.

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To
SQLite Database"), 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 TLiteQuery. Another way is to use the
dataset functionality (the Edit and Post methods) of the TLiteQuery and TLiteTable
components. We will discuss both ways. The goal of this tutorial is to modify the following
record of the table dept:

10	ACCO UNTIN G	NEW YORK

to make it look as follows:

10	RESEA RCH	LOS ANGELE S

Using DataSet Functionality

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

[Delphi]

```
var
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // retrieve data
    LiteQuery.SQL.Text := 'SELECT * FROM dept';
    LiteQuery.Open;
    // to make the record with deptno=10 the current record
    LiteQuery.FindKey([10]);
    // modify record
    LiteQuery.Edit;
    LiteQuery.FieldByName('dname').AsString := 'RESEARCH';
    LiteQuery.FieldByName('loc').AsString := 'LOS ANGELES';
    LiteQuery.Post;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteQuery->Connection = LiteConnection;
    // retrieve data
    LiteQuery->SQL->Text = "SELECT * FROM dept";
    LiteQuery->Open();
    // to make the record with deptno=10 the current record
    LiteQuery->FindKey(ARRAYOFCONST((10)));
    // modify record
    LiteQuery->Edit();
    LiteQuery->FieldByName("dname")->AsString = "RESEARCH";
    LiteQuery->FieldByName("loc")->AsString = "LOS ANGELES";
```

```
      LiteQuery->Post();
    }
  __finally
  {
    LiteQuery->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
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnection is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // set SQL query for update record
    LiteQuery.SQL.Clear;
    LiteQuery.SQL.Add('UPDATE dept SET dname = :dname, loc = :loc WHERE dept
    // set parameters
    LiteQuery.ParamByName('deptno').AsInteger := 10;
    LiteQuery.ParamByName('dname').AsString := 'RESEARCH';
    LiteQuery.ParamByName('loc').AsString := 'LOS ANGELES';
    // execute query
    q.Execute;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteQuery->Connection = LiteConnection;
    // set SQL query for update record
    LiteQuery->SQL->Clear();
    LiteQuery->SQL->Add("UPDATE dept SET dname = :dname, loc = :loc WHERE de
    // set parameters
    LiteQuery->ParamByName("deptno")->AsInteger = 10;
    LiteQuery->ParamByName("dname")->AsString = "RESEARCH";
    LiteQuery->ParamByName("loc")->AsString = "LOS ANGELES";
```

```
      // execute query
      LiteQuery->Execute();
   }
   __finally
   {
      LiteQuery->Free();
   }
}
```

DML Statements As Plain Text

[Delphi]

```
var
  LiteQuery: TLiteQuery;
begin
  LiteQuery := TLiteQuery.Create(nil);
  try
    // LiteConnecton is either TLiteConnection already set up
    LiteQuery.Connection := LiteConnection;
    // set SQL query for update record
    LiteQuery.SQL.Clear;
    LiteQuery.SQL.Add('UPDATE dept SET dname = ''RESEARCH'', loc = ''LOS ANG
    // execute query
    LiteQuery.Execute;
  finally
    LiteQuery.Free;
  end;
end;
```

[C++Builder]

```
{
  TLiteQuery* LiteQuery = new TLiteQuery(NULL);
  try
  {
    // LiteConnection is either TLiteConnection already set up
    LiteQuery->Connection = LiteConnection;
    // set SQL query for update record
    LiteQuery->SQL->Clear();
    LiteQuery->SQL->Add("UPDATE dept SET dname = 'RESEARCH', loc = 'LOS ANGE
    // execute query
    LiteQuery->Execute();
  }
  __finally
  {
    LiteQuery->Free();
  }
}
```

Request Support   DAC Forum   Provide Feedback

3.8 Demo Projects

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

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

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

Where are the LiteDAC demo projects located?

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

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

  |—LiteDACDemo [The main LiteDAC demo project]
  |—ThirdParty
  |   |— [A collection of demo projects on integration with third-
  party components]
  |—Miscellaneous
      |— [Some other demo projects on design technologies]
```

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

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

To explore a LiteDAC demo project,

1. Launch your IDE.

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

3. Find the directory you installed LiteDAC 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 SQLite. 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 LiteDAC demo directory includes two sample SQL scripts for creating and dropping all the test schema objects used in the LiteDAC 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 LiteDAC demo project in the Getting Started topic. The other LiteDAC demo projects include a *ReadMe.txt* file with individual building and launching instructions.

Demo project descriptions

LiteDACDemo

LiteDACDemo 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 LiteDAC components.

General demos

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

SQLite-specific demos

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

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

Working with Components

Name	Description
ConnectDialog	Demonstrates how to customize the LiteDAC connect dialog. Changes the standard LiteDAC connect dialog to a custom connect dialog. The customized sample dialog is inherited from the TForm 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.
CRDBGrid	Demonstrates how to work with the TCRDBGrid component. Shows off the main TCRDBGrid features, like filtering, searching, stretching, using compound headers, and more.
Dump	Demonstrates how to backup data from tables with the TLiteDump component. Shows how to use scripts created during back up to restore table data. This demo lets you back up a table either by specifying the table name or by writing a SELECT query.
Loader	Uses the TLiteLoader component to quickly load data into a server table. This demo also compares the two TLiteLoader data loading handlers: GetColumnData and PutData.
Query	Demonstrates working with TLiteQuery, which is one of the most useful LiteDAC components. Includes many TLiteQuery usage scenarios. Demonstrates how to execute queries in both standard and NonBlocking mode and how to edit data and export it to XML files.
	Note: This is a very good introductory demo. We recommend starting here when first becoming familiar with LiteDAC.

Sql	Uses TLiteSQL to execute SQL statements. Demonstrates how to work in a separate thread, in standard mode, in NonBlocking mode, and how to break long-duration query execution.
Table	Demonstrates how to use TLiteTable 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 TLiteUpdateSQL component to customize update commands. Lets you optionally use T:Devart.SQLiteDAC.TLiteCommand and TLiteQuery objects for carrying out insert, delete, query, and update commands.
VirtualTable	Demonstrates working with the TVirtualTable component. This sample shows how to fill virtual dataset with data from other datasets, filter data by a given criteria, locate specified records, perform file operations, and change data and table structure.

General Demos

Name	Description
CachedUpdates	Demonstrates how to perform the most important tasks of working with data in CachedUpdates mode, including highlighting uncommitted changes, managing transactions, and committing changes in a batch.
FilterAndIndex	Demonstrates LiteDAC'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 LiteDAC 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.
Lock	Demonstrates the recommended approach for managing transactions with the TLiteConnection component. The TLiteConnection interface provides a wrapper for SQLite server commands like START TRANSACTION, COMMIT, ROLLBACK.

SQLite-specific Demos

Name	Description
Pictures	Uses LiteDAC functionality to work with graphics. The sample demonstrates how to retrieve binary data from PgSQL server database and display it on visual components. Sample also shows how to load and save pictures to files and to the database.

Text	Uses LiteDAC functionality to work with text. The sample demonstrates how to retrieve text data from SQL Server database and display it on visual components. Sample also shows how to load and save text to files and to the database.
Functions	Functions Uses LiteDAC functionality to work with SQLite functions. The sample demonstrates how to define custom functions for future use in SQL-statements, how to overload user-defined functions and how to override built-in SQLite functions.

Supplementary Demo Projects

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

Location	Name	Description
Miscellaneous	**Dll**	Demonstrates creating and loading DLLs for LiteDAC-based projects. This demo project consists of two parts - an Pg_Dll project that creates a DLL of a form that sends a query to the server and displays its results, and an Pg_Exe project that can be executed to display a form for loading and running this DLL. Allows you to build a dll for one LiteDAC-based project and load and test it from a separate application.
LiteDACDemo	***LiteDACDemo***	*[Win32 version of the main LiteDAC demo project - see above]*

Request Support   DAC Forum   Provide Feedback

3.9   Deployment

LiteDAC 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 LiteDAC-based applications built without run-time packages, provided you are using a registered version of LiteDAC.

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 LiteDAC Trial Edition, you will need to deploy the following DAC BPL files:

dacXX.bpl	always
litedacXX.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
litedacXX.bpl	always
dacvclXX.bpl	if your application uses the LiteDACVcl unit
litedacvclXX.bpl	if your application uses the LiteDACVcl unit
crcontrolsXX.bpl	if your application uses the CRDBGrid component

Request Support   DAC Forum   Provide Feedback

4 Using LiteDAC

This section describes basics of using SQLite Data Access Components

- [Connecting in Direct Mode](#)

- [Disabling Direct Mode](#)

- [Updating Data with LiteDAC Dataset Components](#)

- [Master/Detail Relationships](#)

- [Data Type Mapping](#)

- [Data Encryption](#)

- [Database File Encryption](#)

- [Disconnected Mode](#)

- [Increasing Performance](#)

- [Working in an Unstable Network](#)

- [Macros](#)

- [DataSet Manager](#)

- [DBMonitor](#)

- [Writing GUI Applications with LiteDAC](#)

- [Connection Pooling](#)

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

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

- [Demo Projects](#)

- [Deployment](#)

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

4.1 Connecting in Direct Mode

LiteDAC Professional Edition allows to connect to SQLite in two ways: with using SQLite client library, or in Direct mode by linking SQLite library statically in an application. The chosen connection mode is regulated by the TLiteConnection.Options.Direct property.

LiteDAC connection modes

By default, LiteDAC, like most applications that work with SQLite, uses the SQLite client library (sqlite3.dll for Windows, libsqlite3.dylib for MacOS and libsqlite3.so for Linux) to connect to a SQLite database. This is referred to as connecting in Client mode, and it is the usual way to develop SQLite applications with a third-generation language. All SQLite API routines are stored in external library, so the executables for applications, that work using SQLite client, are a bit smaller. However, working in Client mode requires SQLite client library to be present on target workstations. It's either may cause compatibility issues of your application and the existent SQLite library, or causes a need to deploy a compatible version of the SQLite client library with your application.

LiteDAC Professional Edition includes an option to connect to SQLite directly, using the embedded SQLite3 engine. This is referred to as connecting in Direct mode. Connecting in Direct mode does not require SQLite client software to be present on target machines and saves your application from compatibility problems. Furthermore, our embedded SQLite3 engine supports built-in database encryption, that gives an ability to work with encrypted databases in Direct mode. The only inconvenience is that the application size increases slightly (about 350 KB).

Setting up Direct mode connections

To connect to SQLite database using Direct mode, set up your Direct property to True. This is all you need to do to enable Direct mode connections in your application. You do not have to rewrite other parts of your code.

To return to working through SQLite client software, just set the Direct property to False.

Note: Direct mode is available in LiteDAC Professional Edition, and can be evaluated with LiteDAC Trial Edition.

Advantages of using Direct mode

- Using SQLite client software is not required.

- Application compatibility problems are eliminated.

- Built-in database encryption support.

Connecting in Direct mode is managed transparently by the TLiteConnection object, and you can easily return to connecting via SQLite client library in Client mode at any time, if the restrictions above become critical for you.

Disabling Direct Mode

If you don't plan to use Direct Mode in your application, you can permanently disable it.

   Request Support   DAC Forum   Provide Feedback

4.2 Disabling Direct Mode

Disabling Direct Mode

As described in the Connecting in Direct Mode article, Direct Mode usage leads to increased size of the application executable file. If you don't plan to use Direct Mode in your application, you can permanently disable it, i.e., the SQLite engine will not be embedded into the executable file. For this, open the Project -> Options menu in the IDE, select the Delphi Compiler node in the options list, and add a NOSTATIC value in the Conditional defines property. Also, add path to the [LiteDAC installation folder]\Source folder in the Search path property. Then build the project.

On disabling Direct Mode, the executable file size decreases. The application will be able to work with SQLite using the client library only. On attempt to set the TLiteConnectionOptions.Direct property to True, an exception will be raised: 'Direct Mode disabled'

   Request Support   DAC Forum   Provide Feedback

Reserved.

4.3 Updating Data with LiteDAC Dataset Components

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

The first approach is to use automatic generation of update SQL statements. When using this approach you should specify Key Fields (the KeyFields property) to avoid requesting KeyFields from the database. When SELECT statement uses multiple tables, you can use the UpdatingTable property to specify which table will be updated. If UpdatingTable is blank, the first table of the FROM clause will be used. In the most cases LiteDAC needs an additional information about updating objects. So LiteDAC executes additional queries to the database. This helps to generate correct updating SQL statements but may result in performance decrease. To disable these additional queries, set the ExtendedFieldsInfo option to False.

Another approach is to set update SQL statements using SQLInsert, SQLUpdate, and SQLDelete properties. Use them to specify SQL statements that will be used for corresponding data modifications. It is useful when generating data modification statements is not possible or you need to execute some specific statements. You may also assign the TLiteUpdateSQL component to the UpdateObject property. TLiteUpdateSQL component holds all updating SQL statements in one place. You can generate all these SQL statements using LiteDAC design time editors. For more careful customization of data update operations you can use InsertObject, ModifyObject and DeleteObject properties of the TLiteUpdateSQL component.

See Also
- TLiteQuery

- TLiteTable

- TLiteUpdateSQL

Request Support   DAC Forum   Provide Feedback

4.4 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 LiteDAC 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.

LiteDAC provides two ways to bind tables. First code example shows how to bind two TCustomLiteDataSet components (TLiteQuery or TLiteTable) into MD relationship via parameters.

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

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

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

Now suppose that detail table "Department" 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: TLiteQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TLiteQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TLiteQuery.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 database. Otherwise detail dataset performs query each time when record is selected in master dataset. Using local MD relationship can reduce database 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**

4.5 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, LiteDAC 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 SQLite database:

```
CREATE TABLE NUMERIC_TYPES
(
 ID INTEGER PRIMARY KEY AUTOINCREMENT,
 VALUE1 NUMERIC(4,0),
 VALUE2 NUMERIC(10,0),
 VALUE3 NUMERIC(15,0),
 VALUE4 NUMERIC(5,2),
 VALUE5 NUMERIC(10,4),
 VALUE6 NUMERIC(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:

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

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

```
uses
  LiteDataTypeMap;
...
var
  DBType: Word;
  MinPrecision: Integer;
  MaxPrecision: Integer;
  MinScale: Integer;
  MaxScale: Integer;
  FieldType: TfieldType;
begin
  DBType        := liteNumeric;
  MinPrecision := 0;
  MaxPrecision := 4;
  MinScale      := 0;
  MaxScale      := 0;
  FieldType     := ftSmallint;
  LiteConnection.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecisio
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
LiteConnection.DataTypeMap.Clear;
// rule for numeric(4,0)
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumeric, 0,      4, 0,      0, f
// rule for numeric(10,0)
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumeric, 5,     10, 0,      0, f
// rule for numeric(15,0)
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumeric, 11, rlAny, 0,      0, f
// rule for numeric(5,2)
```

```
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumeric, 0,      9, 1, rlAny, f
// rule for numeric(10,4)
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumeric, 10, rlAny, 1,    4, f
// rule for numeric(15,6)
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumeric, 10, rlAny, 5, rlAny, f
```

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 a SQLite database:

```
CREATE TABLE NUMBER_TYPES
(
 ID INTEGER PRIMARY KEY AUTOINCREMENT,
 VALUE1 NUMBER(5,2),
 VALUE2 NUMBER(10,4),
 VALUE3 NUMBER(15,6)
)
```

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

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

If rules are set in the following way:

```
LiteConnection.DataTypeMap.Clear;
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumber, 0,      9, rlAny, rlAny,
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumber, 0, rlAny,    0,    4,
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumber, 0, rlAny,    0, rlAny,
```

it will lead to the following result:

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

But if rules are set in the following way:

```
LiteConnection.DataTypeMap.Clear;
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumber, 0, rlAny,    0, rlAny,
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumber, 0, rlAny,    0,    4,
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumber, 0,    9, rlAny, rlAny,
```

it will lead to the following result:

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

This happens because the rule

```
LiteConnection.DataTypeMap.AddDBTypeRule(liteNumber, 0, rlAny, 0, rlAny, ftF
```

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 a SQLite database:

```
CREATE TABLE PERSON
```

```
(
 ID                  INTEGER PRIMARY KEY AUTOINCREMENT,
 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:

```
LiteConnection.DataTypeMap.Clear;
LiteConnection.DataTypeMap.AddDBTypeRule(liteDateTime, 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:

```
LiteQuery.DataTypeMap.Clear;
LiteQuery.DataTypeMap.AddDBTypeRule(liteDateTime, 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:

```
LiteConnection.DataTypeMap.Clear;
LiteConnection.DataTypeMap.AddDBTypeRule(liteDateTime, ftDate);
LiteQuery.DataTypeMap.Clear;
LiteQuery.DataTypeMap.AddDBTypeRule(liteDateTime, ftDateTime);
```

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

```
CREATE TABLE ITEM
(
 ID     INTEGER PRIMARY KEY AUTOINCREMENT,
 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:

```
LiteQuery.DataTypeMap.Clear;
LiteQuery.DataTypeMap.AddDBTypeRule(liteChar, 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:

```
LiteQuery.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:

```
LiteConnection.DataTypeMap.AddDBTypeRule(ibcVarchar, 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

4.6   Data Encryption

LiteDAC 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 database, 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, TLiteEncryptor 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     INTEGER PRIMARY KEY AUTOINCREMENT,
ENAME     VARBINARY(2000),
HIREDATE VARBINARY (200),
SAL       VARBINARY (200),
FOTO      VARBINARY
)
```

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.

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

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

4.7   Database File Encryption

What constitutes Database File Encryption

The SQLite architecture provides the functionality for work with encrypted databases. This means that encoding/decoding is applied to a database file, in the moment of execution of the file read/write operations. This is a low-level encryption "on the fly", it is implemented at the level of the SQLite client library and is completely transparent to the applications working with

the database.

But, the fact is that in the client libraries available at the official SQLite website, the algorithms of database file encryption are not implemented. Therefore, usually, to work with encrypted databases one has to either use a custom-built client library with encryption support, or create an own library from the source code, available on the SQLite website.

LiteDAC functionality for Database File Encryption

LiteDAC provides built-in capabilities for Database File Encryption, which becomes available when working in Direct mode. Database File Encryption, built in LiteDAC, allows to:

- encrypt a database;

- create a new encrypted database;

- connect and work with the encrypted database;

- change the encryption key of the encrypted database;

- decryp the encrypted database.

To encrypt/decrypt the database file, one of the following encryption algorithms can be used:

- the Triple DES encryption algorithm;

- the Blowfish encryption algorithm;

- the AES encryption algorithm with a key size of 128 bits;

- the AES encryption algorithm with a key size of 192 bits;

- the AES encryption algorithm with a key size of 256 bits;

- the Cast-128 encryption algorithm;

- the RC4 encryption algorithm.

Important note: there are no strict standardized requirements for implementation of database file encryption in SQLite. Therefore, implementation of Database File Encryption in LiteDAC is incompatible with other implementations. When using LiteDAC, it is possible to work only with encrypted databases, created with the use of LiteDAC. In turn, no third-party application will be able to work with encrypted databases, created with the use of LiteDAC

The difference between Database File Encryption and

Data Encryption.

The functionality of [Data Encryption](#), which is realized with the help of the P:Devart.SQLiteDac.TLiteEncryptor component, allows to encrypt individual fields in database tables. In this case, the database itself is not encrypted. I.e. on the one hand, the information in this database (with the exception of the encrypted fields) is easily accessible for viewing by any SQLite DB-tools. On the other hand, such database is more simple in terms of modification of data structures.

Database File Encryption encrypts all the data file. Both structure and information on such database becomes unavailable for any third-party applications. An indisputable advantage is the increased level of secrecy of information. The disadvantage is that, for making any changes in the structure of the database, developers will have to use only LiteDAC.

Both Database File Encryption and Data Encryption methods are not mutually exclusive and can be used at the same time.

The usage of Database File Encryption in LiteDAC

To control database encryption in LiteDAC, the following properties and methods of the P:Devart.SQLiteDac.TLiteConnection component are used:

- The TLiteConnection.Options.EncryptionAlgorithm property - specifies the encryption algorithm that will be used to connect to an encrypted database, or to create a new encrypted database.

- The TLiteConnection.EncryptionKey property - specifies the encryption key that will be used to connect to an encrypted database, or to create a new encrypted database.

- The TLiteConnection.EncryptDatabase method - is used to change the encryption key in an encrypted database, or to decrypt the database.

Encrypt a database

The following example shows how to encrypt an existing database:

```
LiteConnection.Database := 'C:\sqlite.db3';                    // the name of the
LiteConnection.Options.ForceCreateDatabase := False;          // to check that t
LiteConnection.Options.Direct := True;                        // database file e
LiteConnection.Options.EncryptionAlgorithm := leBlowfish;     // the database wi
LiteConnection.EncryptionKey := '';                           // no encryption k
LiteConnection.Open;                                          // connect to the
```

```
LiteConnection.EncryptDatabase ('11111');                        // encrypt the dat
```

Creating of a new encrypted database

The following example shows creating a new encrypted database:

```
LiteConnection.Database := 'C:\sqlite_encoded.db3';        // the name of the
LiteConnection.Options.ForceCreateDatabase := True;       // this will allow
LiteConnection.Options.Direct := True;                    // database file e
LiteConnection.Options.EncryptionAlgorithm := leBlowfish; // the database wi
LiteConnection.EncryptionKey := '11111';                  // the encryption
LiteConnection.Open;                                      // create and conn
```

Connecting to an encrypted database

To connect to an existing encrypted database, the following should be performed:

```
LiteConnection.Database := 'C:\sqlite_encoded.db3';        // the name of the
LiteConnection.Options.ForceCreateDatabase := False;      // to check that t
LiteConnection.Options.Direct := True;                    // database file e
LiteConnection.Options.EncryptionAlgorithm := leBlowfish; // the encryption
LiteConnection.EncryptionKey := '11111';                  // the encryption
LiteConnection.Open;                                      // connect to the
```

Changing the encryption key for the database

To change the encryption key in the encrypted database, you must perform the following:

```
LiteConnection.Database := 'C:\sqlite_encoded.db3';        // the name of the
LiteConnection.Options.ForceCreateDatabase := False;      // to check that t
LiteConnection.Options.Direct := True;                    // database file e
LiteConnection.Options.EncryptionAlgorithm := leBlowfish; // the encryption
LiteConnection.EncryptionKey := '11111';                  // the encryption
LiteConnection.Open;                                      // connect to the
LiteConnection.EncryptDatabase ('22222');                 // change the data
```

After changing the encryption key, the database connection remains open and the further work with the database can continue. However, if disconnected from the database and for subsequent connection, the new value of the encryption key should be assigned to the LiteConnection.EncryptionKey property.

Decryption of the database

The encrypted database can be decrypted, after that it becomes available for viewing and editing in third-party applications. To decrypt the database you must first connect to it, as shown in the examples above, and then execute the LiteConnection.EncryptDatabase('') method, specifying an empty string as a new key.

PRAGMA Encryption

- The PRAGMA ENCRYPTION statement specifies the encryption algorithm that will be used to encrypt a previously connected unencrypted database. The statement can be executed only after the database is connected. The statement must not be used on databases encrypted with a different encryption algorithm. The pragma values are the same as the EncryptionAlgorithm attribute values.

Example:

```
PRAGMA ENCRYPTION=TripleDES;
```

The statement can be executed from any database tool that uses Devart LiteDAC, or with the SQLExecuteDirect API function.

- The PRAGMA REKEY statement – is used to encrypt unencrypted database, to change the encryption key of an encrypted database or to decrypt a database. The statement can be executed only after the database is connected.

Example of encryption or changing an encryption key:

```
PRAGMA REKEY='mynewkey';
```

Example of decryption:

```
PRAGMA REKEY='';
```

The statements can be executed from any database tool that uses Devart LiteDAC, or with the SQLExecuteDirect API function.

Request Support   DAC Forum   Provide Feedback

4.8 Disconnected Mode

In disconnected mode a connection opens only when it is required. After performing all database 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 database (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 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 database.

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

- TLiteQuery.LockMode

- TCustomDAConnection.Pooling

- TCustomDAConnection.Connect

- TCustomDAConnection.Disonnect

- [Working in unstable network](#)

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

4.9   Batch Operations

Data amount processed by modern databases grows steadily. In this regard, there is an acute problem – database performance. Insert, Update and Delete operations have to be performed as fast as possible. Therefore Devart provides several solutions to speed up processing of huge amounts of data. So, for example, insertion of a large portion of data to a DB is supported in the [TLiteLoader](#). Unfortunately, [TLiteLoader](#) 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 [TLiteLoader](#), 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       INTEGER,
  F_INTEGER  INTEGER,
  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
  LiteQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F
  // define the parameter types passed to the query :
  LiteQuery1.Params[0].DataType := ftInteger;
  LiteQuery1.Params[1].DataType := ftInteger;
  LiteQuery1.Params[2].DataType := ftFloat;
  LiteQuery1.Params[3].DataType := ftString;
  LiteQuery1.Params[4].DataType := ftDateTime;
  // specify the array dimension:
  LiteQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to LiteQuery1.Params.ValueCount - 1 do begin
    LiteQuery1.Params[0][i].AsInteger := i + 1;
    LiteQuery1.Params[1][i].AsInteger := i + 2000 + 1;
    LiteQuery1.Params[2][i].AsFloat := (i + 1) / 12;
    LiteQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
    LiteQuery1.Params[4][i].AsDateTime := Now;
  end;
  // insert 1000 rows into the BATCH_TEST table
```

```
  LiteQuery1.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:

```
LiteQuery1.Execute(1000);
```

2×500 rows:

```
// insert first 500 rows
LiteQuery1.Execute(500, 0);
// insert next 500 rows
LiteQuery1.Execute(500, 500);
```

500 rows, then 300, and finally 200:

```
// insert 500 rows
LiteQuery1.Execute(500, 0);
// insert next 300 rows starting from 500
LiteQuery1.Execute(300, 500);
// insert next 200 rows starting from 800
LiteQuery1.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
  LiteQuery1.SQL.Text := 'UPDATE BATCH_TEST SET F_INTEGER=:F_INTEGER, F_FLOA
  // define parameter types passed to the query:
  LiteQuery1.Params[0].DataType := ftInteger;
  LiteQuery1.Params[1].DataType := ftFloat;
  LiteQuery1.Params[2].DataType := ftString;
  LiteQuery1.Params[3].DataType := ftDateTime;
  LiteQuery1.Params[4].DataType := ftInteger;
  // specify the array dimension:
  LiteQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to 1000 - 1 do begin
    LiteQuery1.Params[0][i].AsInteger := i - 2000 + 1;
    LiteQuery1.Params[1][i].AsFloat := (i + 1) / 100;
```

```
    LiteQuery1.Params[2][i].AsString := 'New Values ' + IntToStr(i + 1);
    LiteQuery1.Params[3][i].AsDateTime := Now;
    LiteQuery1.Params[4][i].AsInteger := i + 1;
  end;
  // update 1000 rows in the BATCH_TEST table
  LiteQuery1.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
  LiteQuery1.SQL.Text := 'DELETE FROM BATCH_TEST WHERE ID=:ID';
  // define parameter types passed to the query:
  LiteQuery1.Params[0].DataType := ftInteger;
  // specify the array dimension
  LiteQuery1.Params.ValueCount := 1000;
  // populate the arrays with parameter values
  for i := 0 to 1000 - 1 do
    LiteQuery1.Params[0][i].AsInteger := i + 1;
  // delete 1000 rows from the BATCH_TEST table
  LiteQuery1.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	2292	0.92
Update	2535	2.63
Delete	2175	0.44
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
  LiteQuery1.Params.ParamByName('ID')[i].AsInteger := i + 1;
  LiteQuery1.Params.ParamByName('F_INTEGER')[i].AsInteger := i + 2000 + 1;
  LiteQuery1.Params.ParamByName('F_FLOAT')[i].AsFloat := (i + 1) / 12;
  LiteQuery1.Params.ParamByName('F_STRING')[i].AsString := 'Values ' + IntTo
  LiteQuery1.Params.ParamByName('F_DATE')[i].AsDateTime := Now;
end;
```

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

Request Support   DAC Forum   Provide Feedback

4.10   Increasing Performance

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

Connect

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

Execute

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

To execute SQL statements a TLiteSQL component is more preferable than TLiteQuery. 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

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

- FetchRows

- 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 database 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 database.

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

Request Support   DAC Forum   Provide Feedback

4.11   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 database. 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 database.

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

- 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), LiteDAC 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

4.12 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 LiteDAC 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;
```

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

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

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

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

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

See Also

- TMacro

- TCustomDADataSet.MacroByName

- TCustomDADataSet.Macros

Request Support   DAC Forum   Provide Feedback

4.13 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.
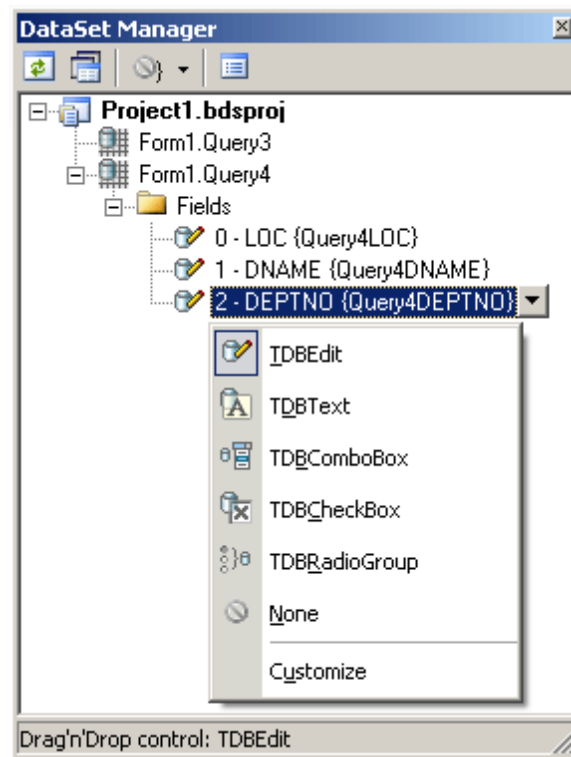
Using the DataSet Manager window, you can:

• 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.
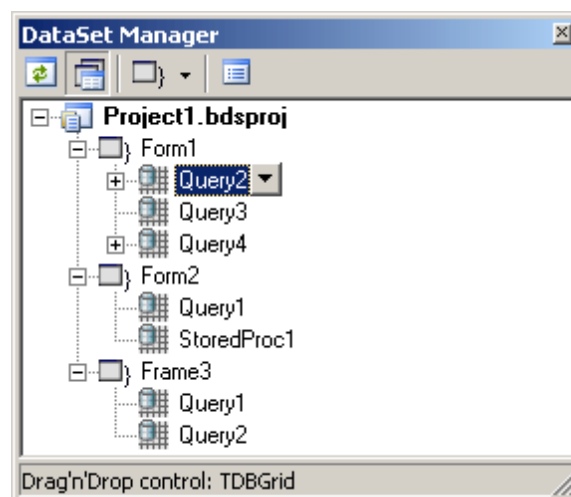
Creating Data-bound Controls

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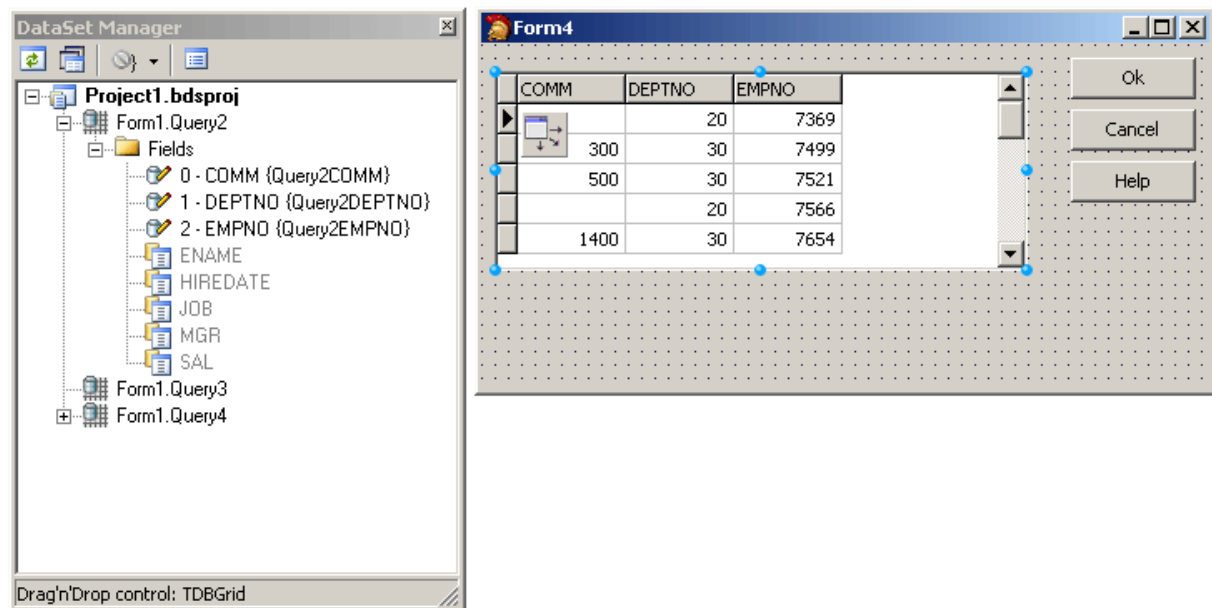
To use Details layout choose Details from the DataSet node drop-down control list in the DataSet Manager window. Then select required controls in the drop-down control list for each DataSet field. DataSet fields must be created. After setting required options you can drag the DataSet to the form from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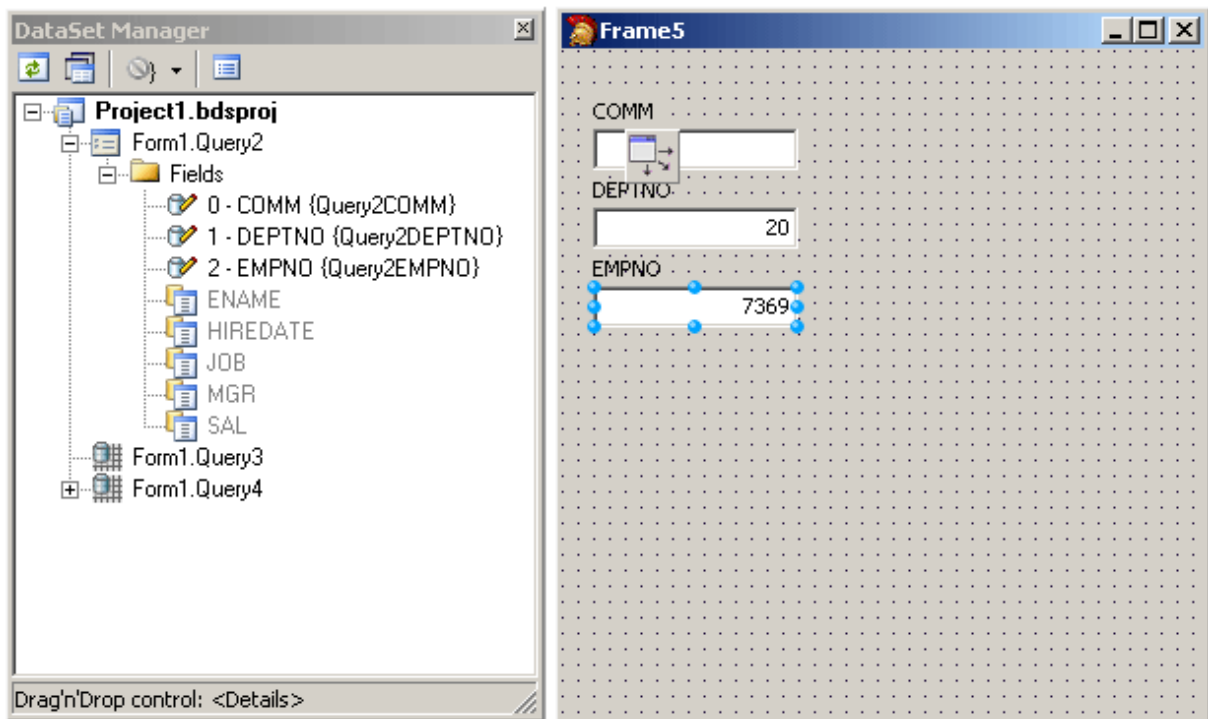
Adding custom controls to the DataSet Manager window

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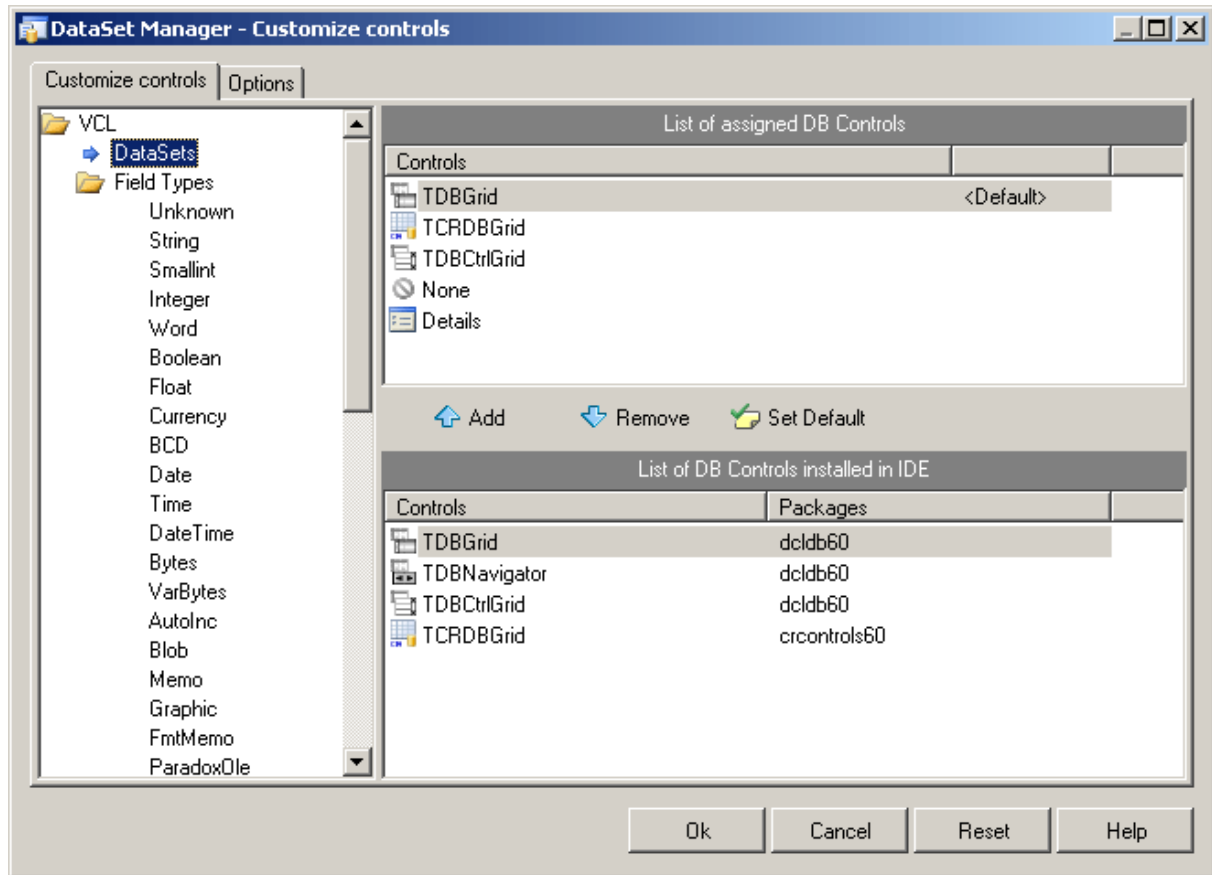
The default configuration can easily be restored by pressing Reset button in the *DataSet Manager - Options* dialog.

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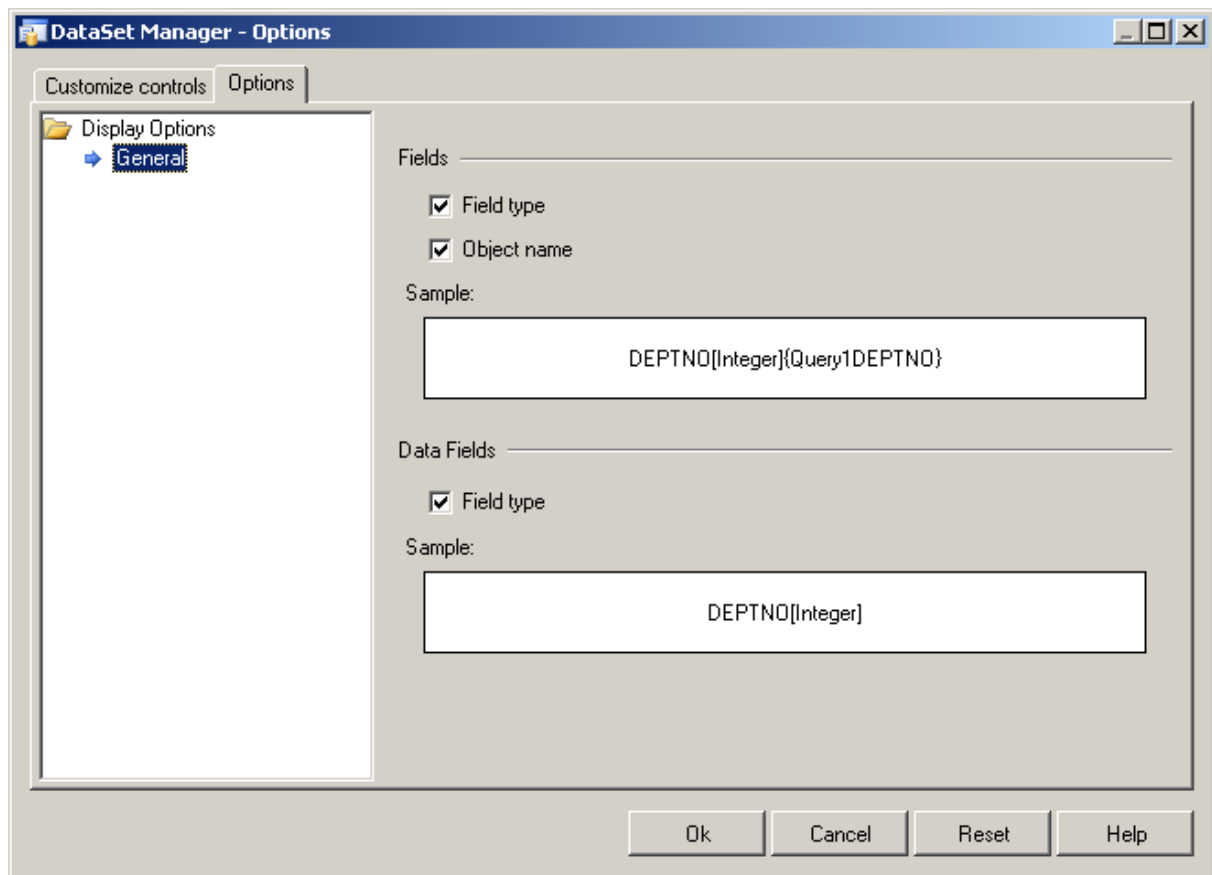
You can chose what information will be added to names of the Field and Data Field objects in the main window of DataSet Manager. Below you can see the example.

Request Support   DAC Forum   Provide Feedback

4.14 DBMonitor

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

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 TLiteSQLMonitor 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

4.15   Writing GUI Applications with LiteDAC

LiteDAC 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 LiteDacVcl unit in your application. This feature is needed for writing console applications.

Delphi and C++Builder

By default LiteDAC does not require Forms, Controls and other GUI related units. Only TLiteConnectDialog component require the Forms unit.

Request Support   DAC Forum   Provide Feedback

4.16   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, P:Devart.Dac.TCustomDAConnection.Database, P:Devart.Dac.TCustomDAConnection.EncryptionKey, 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 TLiteConnectionPoolManager. 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

4.17   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

4.18 Database Specific Aspects of 64-bit Development

SQLite Connectivity Aspects

Using client library:

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 SQLite database at design-time, you must have a 32-bit SQLite 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 SQLite database in run-time (64-bit application), you must have the 64-bit client library placed to the *C:\Windows\System32* directory.

Request Support   DAC Forum   Provide Feedback

5 Reference

This page shortly describes units that exist in LiteDAC.

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	
[DAAlerter](#)	This unit contains the base class for the TLiteAlerter component.
[DADump](#)	This unit contains the base class for the TLiteDump component.
[DALoader](#)	This unit contains the base class for the TLiteLoader component.
[DAScript](#)	This unit contains the base class for the TLiteScript component.
[DASQLMonitor](#)	This unit contains the base class for the TLiteSQLMonitor component.
[DBAccess](#)	This unit contains base classes for most of the components.
[LiteAccess](#)	This unit contains main components of LiteDAC
[LiteDacVcl](#)	This unit contains the visual constituent of LiteDAC.
[LiteDump](#)	This unit contains implementation of the [TLiteDump](#) component
[LiteError](#)	LiteError unit implements the T.Devart.LiteDac.ESQLiteError class.
[LiteLoader](#)	This unit contains implementation of the [TLiteLoader](#) component
[LiteScript](#)	This unit contains implementation of the [TLiteScript](#) component.
[LiteSQLMonitor](#)	This unit contains implementation of the [TLiteSQLMonitor](#) component.
[MemData](#)	This unit contains classes for storing data in memory.
[MemDS](#)	This unit contains implementation of the

	TMemDataSet class.
[VirtualDataSet](#)	This unit contains implementation of the TVirtualDataSet component.
[VirtualTable](#)	This unit contains implementation of the TVirtualTable component.

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

5.1   CRAccess

This unit contains base classes for accessing databases.

Classes

Name	Description
[TCRCursor](#)	A base class for classes that work with database cursors.

Types

Name	Description
[TBeforeFetchProc](#)	This type is used for the [TCustomDADataSet.BeforeFetch](#) event.

Enumerations

Name	Description
[TCRIsolationLevel](#)	Specifies how to handle transactions containing database modifications.
[TCRTransactionAction](#)	Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.
[TCursorState](#)	Used to set cursor state

5.1.1 Classes

Classes in the **CRAccess** unit.

Classes

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

5.1.1.1 TCRCursor Class

A base class for classes that work with database cursors.

For a list of all members of this type, see TCRCursor members.

Unit

CRAccess

Syntax

```
TCRCursor = class(TSharedObject);
```

Remarks

TCRCursor is a base class for classes that work with database cursors.

Inheritance Hierarchy

TSharedObject

  TCRCursor

5.1.1.1.1 Members

TCRCursor class overview.

Properties

Name	Description
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

    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.

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 LiteDAC 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.

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

5.2.1.1.2.6 FieldMappingMode Property

Used to specify the way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.

Class

`TCRBatchMove`

Syntax

```
property FieldMappingMode: TCRFieldMappingMode default
mmFieldIndex;
```

Remarks

Specifies in what way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.

Request Support   DAC Forum   Provide Feedback

5.2.1.1.2.7 KeyViolCount Property

Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.

Class

`TCRBatchMove`

Syntax

```
property KeyViolCount: Integer;
```

Remarks

Use the KeyViolCount property to get the number of records that could not be replaced, added, deleted from the destination dataset because of integrity or key violations.

If AbortOnKeyViol is True, then KeyViolCount will never exceed one, because the operation aborts when the integrity or key violation occurs.

See Also

- AbortOnKeyViol

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.

© 1997-2024 Devart. All Rights Reserved.   **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.

© 1997-2024 Devart. All Rights Reserved.   **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

Reserved.

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.

© 1997-2024 Devart. All Rights Reserved.   [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**

Devart. All Rights
Reserved.

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.

© 1997-2024
Devart. All Rights
Reserved.

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.

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.

5.4 DAAlerter

This unit contains the base class for the TLiteAlerter component.

Classes

Name	Description

[TDAAlerter](#)	A base class that defines functionality for database event notification.

Types

Name	Description
[TAlerterErrorEvent](#)	This type is used for the TDAAlerter.OnError event.

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

5.4.1   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](#)

5.4.1.1   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

5.4.1.1.1 Members

TDAAlerter class overview.

Properties

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever connection opens.
Connection	Used to specify the connection for TDAAlerter.

Methods

Name	Description
SendEvent	Sends an event with Name and content Message.
Start	Starts waiting process.
Stop	Stops waiting process.

Events

Name	Description

OnError	Occurs if an exception occurs in waiting process

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

5.4.1.1.2 Properties

Properties of the **TDAAlerter** class.

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

Public

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever connection opens.
Connection	Used to specify the connection for TDAAlerter.

See Also

- TDAAlerter Class

- TDAAlerter Class Members

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

5.4.1.1.2.1 Active Property

Used to determine if TDAAlerter waits for messages.

Class

TDAAlerter

Syntax

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

Remarks

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

See Also

- [Start](#)

- [Stop](#)

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

5.4.1.1.2.2 AutoRegister Property

Used to automatically register events whenever connection opens.

Class

[TDAAlerter](#)

Syntax

```
property AutoRegister: boolean default False;
```

Remarks

Set the AutoRegister property to True to automatically register events whenever connection opens.

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

5.4.1.1.2.3 Connection Property

Used to specify the connection for TDAAlerter.

Class

[TDAAlerter](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify the connection for TDAAlerter.

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

5.4.1.1.3 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**

5.4.1.1.3.1 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.

5.4.1.1.3.2 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

5.4.1.1.3.3 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](#)

5.4.1.1.4 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](#)

5.4.1.1.4.1 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**

5.4.2 Types

Types in the **DAAlerter** unit.

Types

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

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

5.4.2.1 TAlerterErrorEvent Procedure Reference

This type is used for the TDAAlerter.OnError event.

Unit

DAAlerter

Syntax

```
TAlerterErrorEvent = procedure (Sender: TDAAlerter; E: Exception)
```

```
of object;
```

Parameters

Sender

  An object that raised the event.

E

  Exception object.

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

5.5 DADump

This unit contains the base class for the TLiteDump component.

Classes

Name	Description
[TDADump](#)	A base class that defines functionality for descendant classes that dump database objects to a script.
[TDADumpOptions](#)	This class allows setting up the behaviour of the TDADump class.

Types

Name	Description
[TDABackupProgressEvent](#)	This type is used for the [TDADump.OnBackupProgress](#) event.
[TDARestoreProgressEvent](#)	This type is used for the [TDADump.OnRestoreProgress](#) event.

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

5.5.1 Classes

Classes in the **DADump** unit.

Classes

Name	Description
[TDADump](#)	A base class that defines functionality for descendant classes that dump database objects to a script.
[TDADumpOptions](#)	This class allows setting up the behaviour of the TDADump class.

Request Support   DAC Forum   Provide Feedback

5.5.1.1 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**

5.5.1.1.1 Members

TDADump class overview.

Properties

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Debug	Used to display the statement that is being executed and the values and types of its parameters.
Options	Used to specify the behaviour of a TDADump component.
SQL	Used to set or get the dump script.
TableNames	Used to set the names of the tables to dump.

Methods

Name	Description
Backup	Dumps database objects to the TDADump.SQL property.
BackupQuery	Dumps the results of a particular query.
BackupToFile	Dumps database objects to the specified file.
BackupToStream	Dumps database objects to the stream.
Restore	Executes a script contained in the SQL property.
RestoreFromFile	Executes a script from a file.
RestoreFromStream	Executes a script received

	from the stream.

Events

Name	Description
OnBackupProgress	Occurs to indicate the TDADump.Backup, M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError	Occurs when SQLite 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

5.5.1.1.2 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

5.5.1.1.2.1 Connection Property

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

Class

TDADump

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection

property.

See Also
- TCustomDAConnection

5.5.1.1.2.2 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 LiteDacVcl unit to the uses clause of any unit in your project to make the

Debug property work.

Note: If TLiteSQLMonitor is used in the project and the TLiteSQLMonitor.Active property is set

to False, the debug window is not displayed.

See Also
- TCustomDADataSet.Debug

- TCustomDASQL.Debug

5.5.1.1.2.3 Options Property

Used to specify the behaviour of a TDADump component.

Class

[TDADump](#)

Syntax

```
property Options: TDADumpOptions;
```

Remarks

Use the Options property to specify the behaviour of a TDADump component.

Descriptions of all options are in the table below.

Option Name	Description
[AddDrop](#)	Used to add drop statements to a script before creating statements.
[CompleteInsert](#)	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
[GenerateHeader](#)	Used to add a comment header to a script.
[QuoteNames](#)	Used for TDADump to quote all database object names in generated SQL statements.

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

5.5.1.1.2.4 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

5.5.1.1.2.5 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

5.5.1.1.3 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

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

5.5.1.1.3.1 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](#)

5.5.1.1.3.2 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](#)

5.5.1.1.3.3 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](#)

5.5.1.1.3.4 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

5.5.1.1.3.5 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](#)

5.5.1.1.3.6 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](#)

5.5.1.1.3.7 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](#)

5.5.1.1.4 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.Ba ckupToFile(System.String) or M:Devart.Dac.TDADump.Ba ckupToStream(Borland.Vcl. TStream) method execution progress.
[OnError](#)	Occurs when SQLite raises some error on [TDADump.Restore](#).
[OnRestoreProgress](#)	Occurs to indicate the [TDADump.Restore](#), [TDADump.RestoreFromFile](#), or [TDADump.RestoreFromStr eam](#) method execution progress.

See Also

- [TDADump Class](#)

- [TDADump Class Members](#)

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

5.5.1.1.4.1 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

Request Support   DAC Forum   Provide Feedback

5.5.1.1.4.2 OnError Event

Occurs when SQLite raises some error on Restore.

Class

TDADump

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when SQLite 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

5.5.1.1.4.3 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

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

5.5.1.2   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);
```

© 1997-2024   Request Support   DAC Forum   Provide Feedback

5.5.1.2.1 Members

TDADumpOptions class overview.

Properties

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

Request Support DAC Forum Provide Feedback

5.5.1.2.2 Properties

Properties of the **TDADumpOptions** class.

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

Published

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is

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

See Also

- TDADumpOptions Class

- TDADumpOptions Class Members

Request Support DAC Forum Provide Feedback

5.5.1.2.2.1 AddDrop Property

Used to add drop statements to a script before creating statements.

Class

TDADumpOptions

Syntax

```
property AddDrop: boolean default True;
```

Remarks

Use the AddDrop property to add drop statements to a script before creating statements.

Request Support DAC Forum Provide Feedback

5.5.1.2.2.2 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

5.5.1.2.2.3 GenerateHeader Property

Used to add a comment header to a script.

Class

[TDADumpOptions](#)

Syntax

```
property GenerateHeader: boolean default True;
```

Remarks

Use the GenerateHeader property to add a comment header to a script. It contains script generation date, DAC version, and some other information.

Request Support   DAC Forum   Provide Feedback

5.5.1.2.2.4 QuoteNames Property

Used for TDADump to quote all database object names in generated SQL statements.

Class

[TDADumpOptions](#)

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If the QuoteNames property is True, TDADump quotes all database object names in generated SQL statements.

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

5.5.2 Types

Types in the **DADump** unit.

Types

Name	Description
[TDABackupProgressEvent](#)	This type is used for the [TDADump.OnBackupProgress](#) event.
[TDARestoreProgressEvent](#)	This type is used for the [TDADump.OnRestoreProgress](#) event.

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

5.5.2.1 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

5.5.2.2   **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

5.6   DALoader

This unit contains the base class for the TLiteLoader 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.

© 1997-2024
Devart. All Rights   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)
Reserved.

5.6.1   Classes

Classes in the **DALoader** unit.

Classes

Name	Description
[TDAColumn](#)	Represents the attributes for column loading.
[TDAColumns](#)	Holds a collection of [TDAColumn](#) objects.
[TDALoader](#)	This class allows loading external data into database.
[TDALoaderOptions](#)	Allows loading external data into database.

© 1997-2024   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

Devart. All Rights
Reserved.

5.6.1.1   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

5.6.1.1.1 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**

5.6.1.1.2 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**

5.6.1.1.2.1 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](#)

5.6.1.1.2.2 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](#)

5.6.1.2   **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

5.6.1.2.1 Members

TDAColumns class overview.

Properties

Name	Description
Items	Used to access individual columns.

　　Request Support　　DAC Forum　　Provide Feedback

5.6.1.2.2 Properties

Properties of the **TDAColumns** class.

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

Public

Name	Description
Items	Used to access individual columns.

See Also

- TDAColumns Class

- TDAColumns Class Members

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

5.6.1.2.2.1 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**

5.6.1.3   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.

© 1997-2024 Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback

5.6.1.3.1 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.LoadFromDataSet method is needed.
OnPutData	Occurs when putting loading data by rows is needed.

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

5.6.1.3.2 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**

5.6.1.3.2.1 Columns Property

Used to add a TDAColumn object for each field that will be loaded.

Class

`TDALoader`

Syntax

```
property Columns: TDAColumns stored IsColumnsStored;
```

Remarks

Use the Columns property to add a TDAColumn object for each field that will be loaded.

See Also

- TDAColumns

Request Support   DAC Forum   Provide Feedback

5.6.1.3.2.2 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

5.6.1.3.2.3 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](#)

5.6.1.3.3 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

5.6.1.3.3.1 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

5.6.1.3.3.2 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](#)

5.6.1.3.3.3 LoadFromDataSet Method

Loads data from the specified dataset.

Class

[TDALoader](#)

Syntax

```
procedure LoadFromDataSet(DataSet: TDataSet);
```

Parameters

DataSet
  Holds the dataset to load data from.

Remarks

Call the LoadFromDataSet method to load data from the specified dataset. There is no need

to create columns and write event handlers for OnPutData and OnGetColumnData before calling this method.

Request Support   DAC Forum   Provide Feedback

5.6.1.3.3.4 PutColumnData Method

Puts the value of individual columns.

Class

TDALoader

Overload List

Name	Description
PutColumnData(Col: integer; Row: integer; **const** Value: variant)	Puts the value of individual columns by the column index.
PutColumnData(**const** ColName: **string**; Row: integer; **const** Value: variant)	Puts the value of individual columns by the column name.

Request Support   DAC Forum   Provide Feedback

Puts the value of individual columns by the column index.

Class

TDALoader

Syntax

```
procedure PutColumnData(Col: integer; Row: integer; const Value:
variant); overload; virtual;
```

Parameters

Col
  Holds the index of a loading column. The first column has index 0.

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

Value

Holds the column value.

Remarks

Call the PutColumnData method to put the value of individual columns. 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

5.6.1.3.4 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

5.6.1.3.4.1 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

5.6.1.3.4.2 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

5.6.1.3.4.3 OnPutData Event

Occurs when putting loading data by rows is needed.

Class

TDALoader

Syntax

```
property OnPutData: TDAPutDataEvent;
```

Remarks

Write the OnPutData event handler to put loading data by rows.

Note that rows should be loaded from the first in the ascending order.

To start loading, call the Load method.

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**

5.6.1.4   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**

5.6.1.4.1 Members

TDALoaderOptions class overview.

Properties

Name	Description
UseBlankValues	Forces LiteDAC to fill the buffer with null values after loading a row to the database.

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

5.6.1.4.2 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 LiteDAC 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

5.6.1.4.2.1 UseBlankValues Property

Forces LiteDAC 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 LiteDAC to fill the buffer with null values after loading a row to the database.

Request Support   DAC Forum   Provide Feedback

5.6.2 Types

Types in the **DALoader** unit.

Types

Name	Description
TDAPutDataEvent	This type is used for the TDALoader.OnPutData event.
TGetColumnDataEvent	This type is used for the TDALoader.OnGetColumnData event.
TLoaderProgressEvent	This type is used for the TDALoader.OnProgress event.

Request Support   DAC Forum   Provide Feedback

5.6.2.1 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

5.6.2.2 TGetColumnDataEvent Procedure Reference

This type is used for the TDALoader.OnGetColumnData event.

Unit

DALoader

Syntax

```
TGetColumnDataEvent = procedure (Sender: TObject; Column:
TDAColumn; Row: integer; var Value: variant; var IsEOF: boolean)
of object;
```

Parameters

Sender
  An object that raised the event.

Column
  Points to TDAColumn object that corresponds to the current loading column.

Row
  Indicates the current loading record.

Value
  Holds column values.

IsEOF
  True, if data loading needs to be stopped.

Request Support   DAC Forum   Provide Feedback

5.6.2.3   TLoaderProgressEvent Procedure Reference

This type is used for the TDALoader.OnProgress event.

Unit

DALoader

Syntax

```
TLoaderProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender
  An object that raised the event.

Percent
  Percentage of the load operation progress.

Request Support   DAC Forum   Provide Feedback

5.7 DAScript

This unit contains the base class for the TLiteScript 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**

5.7.1   Classes

Classes in the **DAScript** unit.

Classes

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

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

5.7.1.1   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](#)

5.7.1.1.1 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 SQLite raises

	an error.

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

5.7.1.1.2 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

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

5.7.1.1.2.1 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

5.7.1.1.2.2 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

5.7.1.1.2.3 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 LiteDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: If TLiteSQLMonitor is used in the project and the TLiteSQLMonitor.Active property is set to False, the debug window is not displayed.

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

5.7.1.1.2.4 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](#)

5.7.1.1.2.5 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](#)

5.7.1.1.2.6 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.

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

5.7.1.1.2.7 EndPos Property

Used to get the end position of the current statement.

Class

[TDAScript](#)

Syntax

```
property EndPos: Int64;
```

Remarks

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

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

5.7.1.1.2.8 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**

5.7.1.1.2.9 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.

5.7.1.1.2.10 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.

5.7.1.1.2.11 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.

5.7.1.1.2.12 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

5.7.1.1.2.13 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

5.7.1.1.3 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](#)

5.7.1.1.3.1 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](#)

5.7.1.1.3.2 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

5.7.1.1.3.3 Execute Method

Executes a script.

Class

TDAScript

Syntax

```
procedure Execute; virtual;
```

Remarks

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

See Also

* ExecuteNext

* OnError

* ErrorOffset

5.7.1.1.3.4 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.

5.7.1.1.3.5 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 SQLite raises an error, the OnError event occurs.

See Also

- Execute

- OnError

- ErrorOffset

Request Support   DAC Forum   Provide Feedback

5.7.1.1.3.6 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

5.7.1.1.3.7 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

5.7.1.1.3.8   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](#)

5.7.1.1.4 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 SQLite raises an error.

See Also

- [TDAScript Class](#)

- [TDAScript Class Members](#)

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

5.7.1.1.4.1 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](#)

5.7.1.1.4.2 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

5.7.1.1.4.3 OnError Event

Occurs when SQLite raises an error.

Class

TDAScript

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

Occurs when SQLite 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

5.7.1.2   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

5.7.1.2.1 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.

© 1997-2024
Devart. All Rights
Reserved.

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

5.7.1.2.2 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](#)

5.7.1.2.2.1 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](#)

5.7.1.2.2.2 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**

5.7.1.2.2.3 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**

5.7.1.2.2.4 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**

5.7.1.2.2.5 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

5.7.1.2.2.6 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

5.7.1.2.2.7 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

5.7.1.2.2.8 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.

Request Support   DAC Forum   Provide Feedback

5.7.1.2.2.9 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.

Request Support   DAC Forum   Provide Feedback

5.7.1.2.2.10 StartPos Property

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

Class

TDAStatement

Syntax

```
property StartPos: integer;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.7.1.2.3 Methods

Methods of the **TDAStatement** class.

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

Public

Name	Description
Execute	Executes a statement.

See Also

- TDAStatement Class

- TDAStatement Class Members

Request Support   DAC Forum   Provide Feedback

5.7.1.2.3.1 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

5.7.1.3   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

5.7.1.3.1 Members

TDAStatements class overview.

Properties

Name	Description
Items	Used to access separate script statements.

Request Support   DAC Forum   Provide Feedback

5.7.1.3.2 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

5.7.1.3.2.1 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

5.7.2 Types

Types in the **DAScript** unit.

Types

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

| TOnErrorEvent | This type is used for the TDAScript.OnError event. |

Request Support   DAC Forum   Provide Feedback

5.7.2.1   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

5.7.2.2   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.

5.7.2.3   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.

5.7.3   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

5.7.3.1   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

5.8   DASQLMonitor

This unit contains the base class for the TLiteSQLMonitor 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**

5.8.1 Classes

Classes in the **DASQLMonitor** unit.

Classes

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

5.8.1.1   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.

5.8.1.1.1 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](#)

5.8.1.1.2 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

5.8.1.1.2.1 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

5.8.1.1.2.2 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

Reserved.

5.8.1.1.2.3 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

5.8.1.1.2.4 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**

5.8.1.1.3 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**

5.8.1.1.3.1 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

5.8.1.2 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

5.8.1.2.1 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

5.8.1.2.2 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

5.8.1.2.2.1 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.

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

5.8.1.2.2.2 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.

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

5.8.1.2.2.3 ReconnectTimeout Property

Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.

Class

`TDBMonitorOptions`

Syntax

```
property ReconnectTimeout: integer default
DefaultReconnectTimeout;
```

Remarks

Use the ReconnectTimeout property to set the minimum time (in milliseconds) that should be spent before allowing reconnecting to dbMonitor. If an error occurs when the component sends an event to dbMonitor (dbMonitor is not running), next events are ignored and the component does not restore the connection until ReconnectTimeout is over.

Request Support   DAC Forum   Provide Feedback

5.8.1.2.2.4 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

5.8.2 Types

Types in the **DASQLMonitor** unit.

Types

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

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

5.8.2.1 TDATraceFlags Set

Represents the set of [TDATraceFlag](#).

Unit

`DASQLMonitor`

Syntax

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

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

5.8.2.2 TMonitorOptions Set

Represents the set of [TMonitorOption](#).

Unit

`DASQLMonitor`

Syntax

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

5.8.2.3   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.

5.8.3   Enumerations

Enumerations in the **DASQLMonitor** unit.

Enumerations

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

5.8.3.1   TDATraceFlag Enumeration

Use TraceFlags to specify which database operations the monitor should track in an
application at runtime.

Unit

DASQLMonitor

Syntax

```
TDATraceFlag = (tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt,
tfConnect, tfTransact, tfBlob, tfService, tfMisc, tfParams,
tfObjDestroy, tfPool);
```

Values

Value	Meaning
tfBlob	This option is declared for future use.
tfConnect	Establishing a connection.
tfError	Errors of query execution.
tfMisc	This option is declared for future use.
tfObjDestroy	Destroying of components.
tfParams	Representing parameter values for tfQPrepare and tfQExecute.
tfPool	Connection pool operations.
tfQExecute	Execution of the queries.
tfQFetch	This option is declared for future use.
tfQPrepare	Queries preparation.
tfService	This option is declared for future use.
tfStmt	This option is declared for future use.
tfTransact	Processing transactions.

Request Support   DAC Forum   Provide Feedback

5.8.3.2 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](#)

5.9 DBAccess

This unit contains base classes for most of the components.

Classes

Name	Description
[EDAError](#)	A base class for exceptions that are raised when an error occurs on the server side.
[TCRDataSource](#)	Provides an interface between a DAC dataset components and data-aware controls on a form.

TCustomConnectDialog	A base class for the connect dialog components.
TCustomDAConnection	A base class for components used to establish connections.
TCustomDADataSet	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
TCustomDASQL	A base class for components executing SQL statements that do not return result sets.
TCustomDAUpdateSQL	A base class for components that provide DML statements for more flexible control over data modifications.
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.On](#)

	Error 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**

5.9.1 Classes

Classes in the **DBAccess** unit.

Classes

Name	Description
[EDAError](#)	A base class for exceptions that are raised when an error occurs on the server side.
[TCRDataSource](#)	Provides an interface between a DAC dataset components and data-aware controls on a form.
[TCustomConnectDialog](#)	A base class for the connect dialog components.
[TCustomDAConnection](#)	A base class for components used to establish connections.
[TCustomDADataSet](#)	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
[TCustomDASQL](#)	A base class for components executing SQL statements that do not return result sets.
[TCustomDAUpdateSQL](#)	A base class for components that provide DML statements for more flexible control over data modifications.
[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

5.9.1.1   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

5.9.1.1.1 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

5.9.1.1.2 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

5.9.1.1.2.1 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

5.9.1.1.2.2 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 SQLite. This value is always positive.

Request Support   DAC Forum   Provide Feedback

5.9.1.2 TCRDataSource Class

Provides an interface between a DAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TCRDataSource members.

Unit

DBAccess

Syntax

```
TCRDataSource = class(TDataSource);
```

Remarks

TCRDataSource provides an interface between a DAC dataset components and data-aware controls on a form.

TCRDataSource inherits its functionality directly from the TDataSource component.

At design time assign individual data-aware components' DataSource properties from their drop-down listboxes.

5.9.1.2.1 Members

TCRDataSource class overview.

5.9.1.3　TCustomConnectDialog Class

A base class for the connect dialog components.

For a list of all members of this type, see TCustomConnectDialog members.

Unit

DBAccess

Syntax

```
TCustomConnectDialog = class(TComponent);
```

Remarks

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name before connecting to a database. You can customize captions of buttons and labels by their properties.

5.9.1.3.1 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.
Retries	Used to indicate the number of retries of failed connections.
StoreLogInfo	Used to specify whether the login information should be kept in system registry after a connection was established.

Methods

Name	Description
Execute	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.

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

5.9.1.3.2 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.
[Retries](#)	Used to indicate the number of retries of failed connections.
[StoreLogInfo](#)	Used to specify whether the login information should be kept in system registry after a connection was established.

See Also

- [TCustomConnectDialog Class](#)

- [TCustomConnectDialog Class Members](#)

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

5.9.1.3.2.1 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**

5.9.1.3.2.2 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

5.9.1.3.2.3 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

5.9.1.3.2.4 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 LiteDAC installation directory.

Request Support   DAC Forum   Provide Feedback

5.9.1.3.2.5 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

Devart. All Rights
Reserved.

5.9.1.3.2.6 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

5.9.1.3.2.7 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

Reserved.

5.9.1.3.3 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.

See Also

- TCustomConnectDialog Class

- TCustomConnectDialog Class Members

Request Support   DAC Forum   Provide Feedback

5.9.1.3.3.1 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

5.9.1.4 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

5.9.1.4.1 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.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.

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.
GetKeyFieldNames	Provides a list of available key field names.
GetTableNames	Provides a list of available tables names.
MonitorMessage	Sends a specified message through the TCustomDASQLMonitor component.
Ping	Used to check state of connection to the server.
RemoveFromPool	Marks the connection that should not be returned to the pool after disconnect.

Rollback	Discards all current data changes and ends transaction.
StartTransaction	Begins a new user transaction.

Events

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

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

5.9.1.4.2 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.
[Pooling](#)	Enables or disables using connection pool.
[PoolingOptions](#)	Specifies the behaviour of connection pool.

See Also

- [TCustomDAConnection Class](#)

- [TCustomDAConnection Class Members](#)

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

5.9.1.4.2.1   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](#)

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

5.9.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

[TCustomDAConnection](#)

Syntax

```
property ConnectString: string stored False;
```

Remarks

LiteDAC 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.
[ClientLibrary](#)	Used to set or get the SQLite client library location.
[Database](#)	Used to specify the name of the database to be used once a connection is open.

UseUnicode	Used to enable or disable Unicode support.
Direct	Used to connect to the database directly and without using SQLite3 client library.
ForceCreateDatabase	Used to force TLiteConnection to create a new database before opening a connection, if the database is not exists.

See Also

- Connect

Request Support   DAC Forum   Provide Feedback

5.9.1.4.2.3 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.

Request Support   DAC Forum   Provide Feedback

5.9.1.4.2.4 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

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

5.9.1.4.2.5 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 LiteDacVcl unit appears to the uses clause.

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

5.9.1.4.2.6 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.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

See Also

* Disconnected Mode

* Working in an Unstable Network

Request Support   DAC Forum   Provide Feedback

5.9.1.4.2.7 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

Note: Using Pooling := True can cause errors with working with temporary tables.

See Also
- PoolingOptions

- Connection Pooling

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.9.1.4.2.8 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

5.9.1.4.3 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.
[GetKeyFieldNames](#)	Provides a list of available key field names.
[GetTableNames](#)	Provides a list of available tables names.
[MonitorMessage](#)	Sends a specified message through the [TCustomDASQLMonitor](#) component.
[Ping](#)	Used to check state of connection to the server.
[RemoveFromPool](#)	Marks the connection that should not be returned to the pool after disconnect.
[Rollback](#)	Discards all current data changes and ends transaction.
[StartTransaction](#)	Begins a new user transaction.

See Also

- [TCustomDAConnection Class](#)

- [TCustomDAConnection Class Members](#)

© 1997-2024
Devart. All Rights
Reserved.

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

5.9.1.4.3.1 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](#)

5.9.1.4.3.2 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

- TLiteQuery.FetchAll

Request Support   DAC Forum   Provide Feedback

5.9.1.4.3.3 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 P:Devart.Dac.TCustomDAConnection.Username, P:Devart.Dac.TCustomDAConnection.Password, and P:Devart.Dac.TCustomDAConnection.Server properties.

See Also

- Disconnect

- ConnectDialog

Request Support   DAC Forum   Provide Feedback

5.9.1.4.3.4 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

5.9.1.4.3.5 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 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

5.9.1.4.3.6 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

Request Support   DAC Forum   Provide Feedback

5.9.1.4.3.7 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.

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

5.9.1.4.3.8 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

5.9.1.4.3.9 Ping Method

Used to check state of connection to the server.

Class

TCustomDAConnection

Syntax

```
procedure Ping;
```

Remarks

The method is used for checking server connection state.

Request Support   DAC Forum   Provide Feedback

Reserved.

5.9.1.4.3.10 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](#)

5.9.1.4.3.11 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](#)

- [TLiteQuery.FetchAll](#)

- [TLiteTable.FetchAll](#)

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

5.9.1.4.3.12 StartTransaction Method

Begins a new user transaction.

Class

[`TCustomDAConnection`](#)

Syntax

```
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the [InTransaction](#) property. If InTransaction is True, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling [Commit](#) or [Rollback](#) to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes, or Rollback to cancel them.

See Also

- [Commit](#)

- [Rollback](#)

- [InTransaction](#)

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

5.9.1.4.4 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**

5.9.1.4.4.1 OnConnectionLost Event

This event occurs when connection was lost.

Class

TCustomDAConnection

Syntax

```
property OnConnectionLost: TConnectionLostEvent;
```

Remarks

Write the OnConnectionLost event handler to process fatal errors and perform failover.

Note: To use the OnConnectionLost event handler, you should explicitly add the MemData unit to the 'uses' list and set the TCustomDAConnection.Options.LocalFailover property to True.

Request Support   DAC Forum   Provide Feedback

5.9.1.4.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

`TCustomDAConnection`

Syntax

```
property OnError: TDAConnectionErrorEvent;
```

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

Request Support   DAC Forum   Provide Feedback

5.9.1.5   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](#)

5.9.1.5.1 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.
[CommitUpdates](#) (inherited from **[TMemDataSet](#)**)	Clears the cached updates buffer.
[CreateBlobStream](#)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
[DeferredPost](#) (inherited from **[TMemDataSet](#)**)	Makes permanent changes to the database server.
[DeleteWhere](#)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
[EditRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Enables changing the ending value for an existing range.
[EditRangeStart](#) (inherited from **[TMemDataSet](#)**)	Enables changing the starting value for an existing range.
[Execute](#)	Overloaded. Executes a SQL statement on the server.
[Executing](#)	Indicates whether SQL statement is still being executed.
[Fetched](#)	Used to find out whether TCustomDADataSet has fetched all rows.
[Fetching](#)	Used to learn whether TCustomDADataSet is still

	fetching rows.
[FetchingAll](#)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
[FindKey](#)	Searches for a record which contains specified field values.
[FindMacro](#)	Finds a macro with the specified name.
[FindNearest](#)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
[FindParam](#)	Determines if a parameter with the specified name exists in a dataset.
[GetBlob](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
[GetDataType](#)	Returns internal field types defined in the MemData and accompanying modules.
[GetFieldObject](#)	Returns a multireference shared object from field.
[GetFieldPrecision](#)	Retrieves the precision of a number field.
[GetFieldScale](#)	Retrieves the scale of a number field.
[GetKeyFieldNames](#)	Provides a list of available key field names.
[GetOrderBy](#)	Retrieves an ORDER BY clause from a SQL statement.
[GotoCurrent](#)	Sets the current record in this dataset similar to the current record in another dataset.
[Locate](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Searches a dataset for a specific record and positions the cursor on

	it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a macro with the specified name.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved	Determines if the SQL property value was saved to the BaseSQL property.
UnLock	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute	Occurs after a component has executed a query to database.
AfterFetch	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a

	database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

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

5.9.1.5.2 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**

5.9.1.5.2.1 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

© 1997-2024 Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

TCustomDADataSet

Syntax

```
property Conditions: TDAConditions stored False;
```

See Also

- TDAConditions

5.9.1.5.2.3 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.

5.9.1.5.2.4 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

5.9.1.5.2.5 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 LiteDacVcl unit to the uses clause of any unit in your project to make the

Debug property work.

Note: If TLiteSQLMonitor is used in the project and the TLiteSQLMonitor.Active property is set

to False, the debug window is not displayed.

See Also
* TCustomDASQL.Debug

Request Support   DAC Forum   Provide Feedback

5.9.1.5.2.6 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

5.9.1.5.2.7 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

5.9.1.5.2.8 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

5.9.1.5.2.9 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

5.9.1.5.2.10 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

5.9.1.5.2.11 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](#)

5.9.1.5.2.12 KeyFields Property

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.

Class

[TCustomDADataSet](#)

Syntax

```
property KeyFields: string;
```

Remarks

TCustomDADataset uses the KeyFields property to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database. For this feature KeyFields may hold a list of semicolon-delimited field names. If KeyFields is not defined before opening a dataset, TCustomDADataset requests information about primary keys by sending an additional query to the database.

See Also

- [SQLDelete](#)

- [SQLInsert](#)

- [SQLRefresh](#)

- [SQLUpdate](#)

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

5.9.1.5.2.13 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](#)

5.9.1.5.2.14 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

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

See Also

- TMacro

- MacroByName

- Params

　　　Request Support　　　DAC Forum　　　Provide Feedback

5.9.1.5.2.15 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](#)

5.9.1.5.2.16 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

5.9.1.5.2.17 Options Property

Used to specify the behaviour of TCustomDADataSet object.

Class

`TCustomDADataSet`

Syntax

```
property Options: TDADataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TCustomDADataSet object.

Descriptions of all options are in the table below.

Option Name	Description
AutoPrepare	Used to execute automatic Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the

	DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.
SetFieldsReadOnly	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
[TrimFixedChar](#)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
[UpdateAllFields](#)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
[UpdateBatchSize](#)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

See Also

- [Master/Detail Relationships](#)

- [TMemDataSet.CachedUpdates](#)

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

5.9.1.5.2.18 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

5.9.1.5.2.19 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

5.9.1.5.2.20 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

5.9.1.5.2.21 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

5.9.1.5.2.22 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

5.9.1.5.2.23 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

`TCustomDADataSet`

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.2.24 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

5.9.1.5.2.25 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

5.9.1.5.2.26 SQLInsert Property

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Class

TCustomDADataSet

Syntax

```
property SQLInsert: TStrings;
```

Remarks

Use the SQLInsert property to specify the SQL statement that will be used when applying an insertion to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. Parameters prefixed with OLD_ allow using current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to dataset.

To create a SQLInsert statement at design-time, use the query statements editor.

See Also
- SQL
- SQLUpdate
- SQLDelete
- SQLRefresh

Request Support   DAC Forum   Provide Feedback

5.9.1.5.2.27 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](#)

5.9.1.5.2.28 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

5.9.1.5.2.29 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

5.9.1.5.2.30 SQLUpdate Property

Used to specify a SQL statement that will be used when applying an update to a dataset.

Class

`TCustomDADataSet`

Syntax

```
property SQLUpdate: TStrings;
```

Remarks

Use the SQLUpdate property to specify a SQL statement that will be used when applying an update to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to the dataset.

To create a SQLUpdate statement at design-time, use the query statement editor.

Example

```
UPDATE Orders
   set
      ShipName = :ShipName
   WHERE
      OrderID = :Old_OrderID
```

See Also

- [SQL](#)

- [SQLInsert](#)

- [SQLDelete](#)

- [SQLRefresh](#)

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

5.9.1.5.2.31 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

- [TLiteQuery.FetchAll](#)

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

5.9.1.5.3 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.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.

EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey	Searches for a record which contains specified field values.
FindMacro	Finds a macro with the specified name.
FindNearest	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Determines if a parameter with the specified name exists in a dataset.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject	Returns a multireference shared object from field.
GetFieldPrecision	Retrieves the precision of a number field.

GetFieldScale	Retrieves the scale of a number field.
GetKeyFieldNames	Provides a list of available key field names.
GetOrderBy	Retrieves an ORDER BY clause from a SQL statement.
GotoCurrent	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a macro with the specified name.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.

SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved	Determines if the SQL property value was saved to the BaseSQL property.
UnLock	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TCustomDADataSet Class

- TCustomDADataSet Class Members

5.9.1.5.3.1 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](#)

5.9.1.5.3.2 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. It makes sense to only call BreakExec from another thread.

See Also

- TCustomDADataSet.Execute

- TCustomDASQL.BreakExec

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.3 CreateBlobStream Method

Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

Class

TCustomDADataSet

Syntax

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode):
TStream; override;
```

Parameters

Field
   Holds the BLOB field for reading data from or writing data to from a stream.

Mode
   Holds the stream mode, for which the stream will be used.

Return Value
   The BLOB Stream.

Remarks

Call the CreateBlobStream method to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter. It must be a TBlobField component. You can specify whether the stream will be used for reading, writing, or updating the contents of the field with the Mode parameter.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.4 DeleteWhere Method

Removes WHERE clause from the SQL property and assigns the BaseSQL property.

Class

TCustomDADataSet

Syntax

```
procedure DeleteWhere;
```

Remarks

Call the DeleteWhere method to remove WHERE clause from the the SQL property and assign BaseSQL.

See Also

- AddWhere

- BaseSQL

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.5 Execute Method

Executes a SQL statement on the server.

Class

TCustomDADataSet

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

Request Support   DAC Forum   Provide Feedback

Executes a SQL statement on the server.

Class

`TCustomDADataSet`

Syntax

```
procedure Execute; overload; virtual;
```

Remarks

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

Execute implicitly prepares SQL statement by calling the TCustomDADataSet.Prepare method if the TCustomDADataSet.Options option is set to True and the statement has not been prepared yet. To speed up the performance in case of multiple Execute calls, an application should call Prepare before calling the Execute method for the first time.

See Also
- TCustomDADataSet.AfterExecute

- TCustomDADataSet.Executing

- TCustomDADataSet.Prepare

Request Support   DAC Forum   Provide Feedback

Used to perform Batch operations .

Class

`TCustomDADataSet`

Syntax

```
procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

Iters
Specifies the number of inserted rows.

Offset
Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the Batch operations article for samples.

See Also

- Batch operations

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.6 Executing Method

Indicates whether SQL statement is still being executed.

Class

TCustomDADataSet

Syntax

```
function Executing: boolean;
```

Return Value
True, if SQL statement is still being executed.

Remarks

Check Executing to learn whether TCustomDADataSet is still executing SQL statement.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.7 Fetched Method

Used to find out whether TCustomDADataSet has fetched all rows.

Class

TCustomDADataSet

Syntax

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

Return Value

   True, if all rows have been fetched.

Remarks

Call the Fetched method to find out whether TCustomDADataSet has fetched all rows.

See Also

- Fetching

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.8 Fetching Method

Used to learn whether TCustomDADataSet is still fetching rows.

Class

TCustomDADataSet

Syntax

```
function Fetching: boolean;
```

Return Value

   True, if TCustomDADataSet is still fetching rows.

Remarks

Check Fetching to learn whether TCustomDADataSet is still fetching rows. Use the Fetching method if NonBlocking is True.

See Also

- Executing

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.9 FetchingAll Method

Used to learn whether TCustomDADataSet is fetching all rows to the end.

Class

TCustomDADataSet

Syntax

```
function FetchingAll: boolean;
```

Return Value

   True, if TCustomDADataSet is fetching all rows to the end.

Remarks

Check FetchingAll to learn whether TCustomDADataSet is fetching all rows to the end.

See Also

- Executing

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.10 FindKey Method

Searches for a record which contains specified field values.

Class

TCustomDADataSet

Syntax

```
function FindKey(const KeyValues: array of System.TVarRec):
Boolean;
```

Parameters

KeyValues
  Holds a key.

Remarks

Call the FindKey method to search for a specific record in a dataset. KeyValues holds a comma-delimited array of field values, that is called a key.

This function is provided for BDE compatibility only. It is recommended to use functions [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) for the record search.

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

5.9.1.5.3.11 FindMacro Method

Finds a macro with the specified name.

Class

[TCustomDADataSet](#)

Syntax

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

Parameters

Value
  Holds the name of a macro to search for.

Return Value
  TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the [TMacros.Items](#) property to avoid depending on the order of the items.

See Also

- [TMacro](#)

- [Macros](#)

- MacroByName

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.12 FindNearest Method

Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

Class

`TCustomDADataSet`

Syntax

```
procedure FindNearest(const KeyValues: array of System.TVarRec);
```

Parameters

KeyValues
  Holds the values of the record key fields to which the cursor should be moved.

Remarks

Call the FindNearest method to move the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, the cursor will not move.

This function is provided for BDE compatibility only. It is recommended to use functions TMemDataSet.Locate and TMemDataSet.LocateEx for the record search.

See Also

- TMemDataSet.Locate

- TMemDataSet.LocateEx

- FindKey

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.13 FindParam Method

Determines if a parameter with the specified name exists in a dataset.

Class

TCustomDADataSet

Syntax

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

Parameters

Value
 Holds the name of the param for which to search.

Return Value
 the TDAParam object for the specified Name. Otherwise it returns nil.

Remarks

Call the FindParam method to determine if a specified param component exists in a dataset. Name is the name of the param for which to search. If FindParam finds a param with a matching name, it returns a TDAParam object for the specified Name. Otherwise it returns nil.

See Also

- Params

- ParamByName

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.14 GetDataType Method

Returns internal field types defined in the MemData and accompanying modules.

Class

TCustomDADataSet

Syntax

```
function GetDataType(const FieldName: string): integer; virtual;
```

Parameters

FieldName
  Holds the name of the field.

Return Value

  internal field types defined in MemData and accompanying modules.

Remarks

Call the GetDataType method to return internal field types defined in the MemData and accompanying modules. Internal field data types extend the TFieldType type of VCL by specific database server data types. For example, ftString, ftFile, ftObject.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.15 GetFieldObject Method

Returns a multireference shared object from field.

Class

TCustomDADataSet

Syntax

```
function GetFieldObject(Field: TField): TSharedObject;
overload;function GetFieldObject(Field: TField; RecBuf:
TRecordBuffer): TSharedObject; overload;function
GetFieldObject(FieldDesc: TFieldDesc): TSharedObject;
overload;function GetFieldObject(FieldDesc: TFieldDesc; RecBuf:
TRecordBuffer): TSharedObject; overload;function
GetFieldObject(const FieldName: string): TSharedObject; overload;
```

Parameters

FieldName
  Holds the field name.

Return Value

  multireference shared object.

Remarks

Call the GetFieldObject method to return a multireference shared object from field. If field

does not hold one of the TSharedObject descendants, GetFieldObject raises an exception.

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

5.9.1.5.3.16 GetFieldPrecision Method

Retrieves the precision of a number field.

Class

[TCustomDADataSet](#)

Syntax

```
function GetFieldPrecision(const FieldName: string): integer;
```

Parameters

FieldName
   Holds the existing field name.

Return Value
   precision of number field.

Remarks

Call the GetFieldPrecision method to retrieve the precision of a number field. FieldName is the name of an existing field.

See Also

- [GetFieldScale](#)

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

5.9.1.5.3.17 GetFieldScale Method

Retrieves the scale of a number field.

Class

[TCustomDADataSet](#)

Syntax

```
function GetFieldScale(const FieldName: string): integer;
```

Parameters

FieldName
   Holds the existing field name.

Return Value
   the scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also

• GetFieldPrecision

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.18 GetKeyFieldNames Method

Provides a list of available key field names.

Class

TCustomDADataSet

Syntax

```
procedure GetKeyFieldNames(List: TStrings);
```

Parameters

List
   The list of available key field names

Return Value
   Key field name

Remarks

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also

-

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

5.9.1.5.3.19 GetOrderBy Method

Retrieves an ORDER BY clause from a SQL statement.

Class

TCustomDADataSet

Syntax

```
function GetOrderBy: string;
```

Return Value

an ORDER BY clause from the SQL statement.

Remarks

Call the GetOrderBy method to retrieve an ORDER BY clause from a SQL statement.

Note: GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

See Also

- SetOrderBy

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

5.9.1.5.3.20 GotoCurrent Method

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

Class

TCustomDADataSet

Syntax

```
procedure GotoCurrent(DataSet: TCustomDADataSet);
```

Parameters

DataSet

  Holds the TCustomDADataSet descendant to synchronize the record position with.

Remarks

Call the GotoCurrent method to set the current record in this dataset similar to the current record in another dataset. The key fields in both these DataSets must be coincident.

See Also

- TMemDataSet.Locate
- TMemDataSet.LocateEx

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.21 Lock Method

Locks the current record.

Class

TCustomDADataSet

Syntax

```
procedure Lock; virtual;
```

Remarks

Call the Lock method to lock the current record by executing the statement that is defined in the SQLLock property.

The Lock method sets the savepoint with the name LOCK_ + <component_name>.

See Also

- UnLock

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.22 MacroByName Method

Finds a macro with the specified name.

Class

TCustomDADataSet

Syntax

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

Parameters

Value
 Holds the name of a macro to search for.

Return Value
 TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

Example

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

See Also

- TMacro

- Macros

- FindMacro

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.23 ParamByName Method

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

Class

`TCustomDADataSet`

Syntax

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

Parameters

Value

  Holds the name of the parameter for which to retrieve information.

Return Value

  a TDAParam object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a [TDAParam](#) object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See Also

- Params

- FindParam

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.24 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

[TCustomDADataSet](#)

Syntax

```
procedure Prepare; override;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

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](#)

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

5.9.1.5.3.25 RefreshRecord Method

Actualizes field values for the current record.

Class

[TCustomDADataSet](#)

Syntax

```
procedure RefreshRecord;
```

Remarks

Call the RefreshRecord method to actualize field values for the current record. RefreshRecord performs query to database and refetches new field values from the returned cursor.

See Also
- RefreshOptions

- SQLRefresh

    Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.26 RestoreSQL Method

Restores the SQL property modified by AddWhere and SetOrderBy.

Class
TCustomDADataSet

Syntax

```
procedure RestoreSQL;
```

Remarks

Call the RestoreSQL method to restore the SQL property modified by AddWhere and SetOrderBy.

See Also
- AddWhere

- SetOrderBy

- SaveSQL

- SQLSaved

    Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.27 SaveSQL Method

Saves the SQL property value to BaseSQL.

Class

[TCustomDADataSet](#)

Syntax

```
procedure SaveSQL;
```

Remarks

Call the SaveSQL method to save the SQL property value to the BaseSQL property.

See Also

- [SQLSaved](#)

- [RestoreSQL](#)

- [BaseSQL](#)

© 1997-2024
Devart. All Rights
Reserved.

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

5.9.1.5.3.28 SetOrderBy Method

Builds an ORDER BY clause of a SELECT statement.

Class

[TCustomDADataSet](#)

Syntax

```
procedure SetOrderBy(const Fields: string);
```

Parameters

Fields

Holds the names of the fields which will be added to the ORDER BY clause.

Remarks

Call the SetOrderBy method to build an ORDER BY clause of a SELECT statement. The

fields are identified by the comma-delimited field names.

Note: The GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

Example

```
Query1.SetOrderBy('DeptNo;DName');
```

See Also

- GetOrderBy

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.29 SQLSaved Method

Determines if the SQL property value was saved to the BaseSQL property.

Class

TCustomDADataSet

Syntax

```
function SQLSaved: boolean;
```

Return Value

True, if the SQL property value was saved to the BaseSQL property.

Remarks

Call the SQLSaved method to know whether the SQL property value was saved to the BaseSQL property.

Request Support   DAC Forum   Provide Feedback

5.9.1.5.3.30 UnLock Method

Releases a record lock.

Class

[TCustomDADataSet](#)

Syntax

```
procedure UnLock;
```

Remarks

Call the Unlock method to release the record lock made by the [Lock](#) method before.

Unlock is performed by rolling back to the savepoint set by the Lock method.

See Also

- [Lock](#)

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

5.9.1.5.4 Events

Events of the **TCustomDADataSet** class.

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

Public

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

OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

See Also

- TCustomDADataSet Class

- TCustomDADataSet Class Members

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

5.9.1.5.4.1 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**

5.9.1.5.4.2 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

5.9.1.5.4.3 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

5.9.1.5.4.4 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

5.9.1.5.4.5 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

5.9.1.6 **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](#)

5.9.1.6.1 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.

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

5.9.1.6.2 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](#)

5.9.1.6.2.1 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](#)

5.9.1.6.2.2 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

5.9.1.6.2.3 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 LiteDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: If TLiteSQLMonitor is used in the project and the TLiteSQLMonitor.Active property is set to False, the debug window is not displayed.

See Also
- TCustomDADataSet.Debug

Request Support   DAC Forum   Provide Feedback

5.9.1.6.2.4 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](#)

5.9.1.6.2.5 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](#)

5.9.1.6.2.6 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](#)

5.9.1.6.2.7 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

5.9.1.6.2.8 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

5.9.1.6.2.9 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 LiteSQL 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

5.9.1.6.2.10 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

5.9.1.6.2.11 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

5.9.1.6.2.12 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

TCustomDASQL

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

Request Support   DAC Forum   Provide Feedback

5.9.1.6.2.13 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

5.9.1.6.3 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

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

5.9.1.6.3.1 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

5.9.1.6.3.2 Execute Method

Executes a SQL statement on the server.

Class

TCustomDASQL

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

Request Support   DAC Forum   Provide Feedback

Executes a SQL statement on the server.

Class

TCustomDASQL

Syntax

```
procedure Execute; overload; virtual;
```

Remarks

Call the Execute method to execute a SQL statement on the server. If the SQL statement has OUT parameters, use the TCustomDASQL.ParamByName method or the TCustomDASQL.Params property to get their values. Iters argument specifies the number of times this statement is executed for the DML array operations.

Request Support   DAC Forum   Provide Feedback

Used to perform [Batch operations](#) .

Class

[TCustomDASQL](#)

Syntax

```
procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

Iters
  Specifies the number of inserted rows.

Offset
  Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the [Batch operations](#) article for samples.

See Also

- [Batch operations](#)

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

5.9.1.6.3.3 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

5.9.1.6.3.4 FindMacro Method

Finds a macro with the specified name.

Class

TCustomDASQL

Syntax

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

Parameters

Value
   Holds the name of a macro to search for.

Return Value
   TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

See Also

- TMacro

- Macros

- MacroByName

Request Support   DAC Forum   Provide Feedback

5.9.1.6.3.5 FindParam Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

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

Parameters

Value
  Holds the parameter name to search for.

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

Remarks

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

See Also

- ParamByName

Request Support   DAC Forum   Provide Feedback

5.9.1.6.3.6 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

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

5.9.1.6.3.7 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

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

See Also

- FindParam

Request Support   DAC Forum   Provide Feedback

5.9.1.6.3.8 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 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

5.9.1.6.3.9 UnPrepare Method

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

Class

TCustomDASQL

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free resources allocated for a previously prepared query on the server and client sides.

See Also

• Prepare

Request Support   DAC Forum   Provide Feedback

5.9.1.6.3.10 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

5.9.1.6.4 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

5.9.1.6.4.1 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

5.9.1.7   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

- TLiteQuery.UpdateObject

- TLiteTable.UpdateObject

Request Support   DAC Forum   Provide Feedback

5.9.1.7.1 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**

5.9.1.7.2 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](#)

5.9.1.7.2.1 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

5.9.1.7.2.2 DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class

`TCustomDAUpdateSQL`

Syntax

```
property DeleteObject: TComponent;
```

Remarks

Assign SQL component or a TCustomLiteDataSet 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

5.9.1.7.2.3 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

5.9.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class

`TCustomDAUpdateSQL`

Syntax

```
property InsertObject: TComponent;
```

Remarks

Assign SQL component or TCustomLiteDataSet 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

5.9.1.7.2.5 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

5.9.1.7.2.6 LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

TCustomDAUpdateSQL

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomLiteDataSet 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

5.9.1.7.2.7 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**

5.9.1.7.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class

TCustomDAUpdateSQL

Syntax

```
property ModifyObject: TComponent;
```

Remarks

Assign a SQL component or TCustomLiteDataSet 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**

5.9.1.7.2.9 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

5.9.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

`TCustomDAUpdateSQL`

Syntax

```
property RefreshObject: TComponent;
```

Remarks

Assign a SQL component or TCustomLiteDataSet 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

Reserved.

5.9.1.7.2.11 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

5.9.1.7.2.12 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**

5.9.1.7.3 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**

5.9.1.7.3.1 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

5.9.1.7.3.2   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

5.9.1.8   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

5.9.1.8.1 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**

5.9.1.8.2 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**

5.9.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

`TDACondition`

Syntax

```
property Enabled: Boolean default True;
```

5.9.1.8.2.2 Name Property

The name of the condition

Class

`TDACondition`

Syntax

```
property Name: string;
```

5.9.1.8.2.3 Value Property

The value of the condition

Class

`TDACondition`

Syntax

```
property Value: string;
```

5.9.1.8.3 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

5.9.1.8.3.1 Disable Method

Disables the condition

Class

TDACondition

Syntax

```
procedure Disable;
```

Request Support   DAC Forum   Provide Feedback

5.9.1.8.3.2 Enable Method

Enables the condition

Class

TDACondition

Syntax

```
procedure Enable;
```

Request Support   DAC Forum   Provide Feedback

5.9.1.9   TDAConditions Class

Holds a collection of TDACondition objects.

For a list of all members of this type, see TDAConditions members.

Unit

DBAccess

Syntax

```
TDAConditions = class(TCollection);
```

Remarks

The given example code

```
UniTable1.Conditions.Add('1','JOB="MANAGER"');
UniTable1.Conditions.Add('2','SAL>2500');
UniTable1.Conditions.Enable;
UniTable1.Open;
```

will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
(SAL<2500)
```

Request Support   DAC Forum   Provide Feedback

5.9.1.9.1 Members

TDAConditions class overview.

Properties

Name	Description
Condition	Used to iterate through all the conditions.
Enabled	Indicates whether the condition is enabled
Items	Used to iterate through all conditions.
Text	The property returns condition names and values as CONDITION_NAME=CONDITION
WhereSQL	Returns the SQL WHERE condition added in the Conditions property.

Methods

Name	Description
Add	Overloaded. Adds a condition to the WHERE clause of the query.
Delete	Deletes the condition
Disable	Disables the condition
Enable	Enables the condition
Find	Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.
Get	Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.

[IndexOf](#)	Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.
[Remove](#)	Removes the condition

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

5.9.1.9.2 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](#)

5.9.1.9.2.1 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](#)

5.9.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

[TDAConditions](#)

Syntax

```
property Enabled: Boolean;
```

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

5.9.1.9.2.3 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

5.9.1.9.2.4 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

5.9.1.9.2.5 WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class

`TDAConditions`

Syntax

```
property WhereSQL: string;
```

Request Support   DAC Forum   Provide Feedback

5.9.1.9.3 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

5.9.1.9.3.1 Add Method

Adds a condition to the WHERE clause of the query.

Class

`TDAConditions`

Overload List

Name	Description
Add(**const** Value: **string**; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.
Add(**const** Name: **string**; **const** Value: **string**; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.

Request Support   DAC Forum   Provide Feedback

Adds a condition to the WHERE clause of the query.

Class

`TDAConditions`

Syntax

```
function Add(const Value: string; Enabled: Boolean = True):
TDACondition; overload;
```

Parameters

Value
  The value of the condition

Enabled
  Indicates that the condition is enabled

Remarks

If you want then to access the condition, you should use Add and its name in the Name parameter.

The given example code will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
```

```
and
(SAL<2500)
```

Request Support   DAC Forum   Provide Feedback

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Syntax

```
function Add(const Name: string; const Value: string; Enabled:
Boolean = True): TDACondition; overload;
```

Parameters

Name
   Sets the name of the condition

Value
   The value of the condition

Enabled
   Indicates that the condition is enabled

Remarks

The given example code will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
(SAL<2500)
```

Request Support   DAC Forum   Provide Feedback

5.9.1.9.3.2 Delete Method

Deletes the condition

Class

TDAConditions

Syntax

```
procedure Delete(Index: integer);
```

Parameters

Index

Index of the condition

© 1997-2024
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.9.3.3 Disable Method

Disables the condition

Class

`TDAConditions`

Syntax

```
procedure Disable;
```

© 1997-2024
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.9.3.4 Enable Method

Enables the condition

Class

`TDAConditions`

Syntax

```
procedure Enable;
```

© 1997-2024
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.9.1.9.3.5 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

5.9.1.9.3.6 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

5.9.1.9.3.7 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](#)

5.9.1.9.3.8 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](#)

5.9.1.10 **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](#)

TDAConnectionOptions class overview.

Properties

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

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

Properties of the **TDAConnectionOptions** class.

For a complete list of the **TDAConnectionOptions** class members, see the

TDAConnectionOptions Members topic.

Public

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

KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

Published

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.

See Also

- TDAConnectionOptions Class

- TDAConnectionOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.9.1.10.2.1 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

5.9.1.10.2.2 DisconnectedMode Property

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

Class

TDAConnectionOptions

Syntax

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

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.9.1.10.2.3 KeepDesignConnected Property

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

Class

TDAConnectionOptions

Syntax

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

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.9.1.10.2.4 LocalFailover Property

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

Class

TDAConnectionOptions

Syntax

```
property LocalFailover: boolean default False;
```

Remarks

If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks. Read the Working in an Unstable Network topic for more information about using failover.

Request Support   DAC Forum   Provide Feedback

5.9.1.11 TDAConnectionSSLOptions Class

This class is used to set up the SSL options.

For a list of all members of this type, see TDAConnectionSSLOptions members.

Unit

[DBAccess](#)

Syntax

```
TDAConnectionSSLOptions = class(TPersistent);
```

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

5.9.1.11.1 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**

5.9.1.11.2 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

5.9.1.11.2.1 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

5.9.1.11.2.2 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

5.9.1.11.2.3 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

5.9.1.11.2.4 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

5.9.1.12 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

5.9.1.12.1 Members

TDADataSetOptions class overview.

Properties

Name	Description
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill

	the Origin property of the TField objects by appropriate value when opening a dataset.
[FlatBuffers](#)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
[InsertAllSetFields](#)	Used to include all set dataset fields in the generated INSERT statement
[LocalMasterDetail](#)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
[LongStrings](#)	Used to represent string fields with the length that is greater than 255 as TStringField.
[MasterFieldsNullable](#)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
[NumberRange](#)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
[QueryRecCount](#)	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
[QuoteNames](#)	Used for TCustomDADataSet to quote all database object

	names in autogenerated SQL statements such as update SQL.
[RemoveOnRefresh](#)	Used for a dataset to locally remove a record that can not be found on the server.
[RequiredFields](#)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
[ReturnParams](#)	Used to return the new value of fields to dataset after insert or update.
[SetFieldsReadOnly](#)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
[StrictUpdate](#)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
[TrimFixedChar](#)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
[UpdateAllFields](#)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
[UpdateBatchSize](#)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Request Support   DAC Forum   Provide Feedback

5.9.1.12.2 Properties

Properties of the **TDADataSetOptions** class.

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

Public

Name	Description
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataSet to use

	local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.

SetFieldsReadOnly	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

See Also

- TDADataSetOptions Class

- TDADataSetOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.9.1.12.2.1 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](#)

5.9.1.12.2.2 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](#)

5.9.1.12.2.3 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

5.9.1.12.2.4 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

5.9.1.12.2.5 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

5.9.1.12.2.6 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

5.9.1.12.2.7 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

5.9.1.12.2.8 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

5.9.1.12.2.9 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

5.9.1.12.2.10 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

5.9.1.12.2.11 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](#)

5.9.1.12.2.12 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](#)

5.9.1.12.2.13 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

5.9.1.12.2.14 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

5.9.1.12.2.15 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.

5.9.1.12.2.16 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.

5.9.1.12.2.17 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

5.9.1.12.2.18 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

5.9.1.12.2.19 StrictUpdate Property

Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.

Class

TDADataSetOptions

Syntax

```
property StrictUpdate: boolean default True;
```

Remarks

If True, TCustomDADataSet raises an exception when the number of updated or deleted records is not equal 1. Setting this option also causes the exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you execute SQL query, that doesn't return resultset.

Note: There can be problems if this option is set to True and triggers for UPDATE, DELETE, REFRESH commands that are defined for the table. So it is recommended to disable (set to False) this option with triggers.

TrimFixedChar specifies whether to discard all trailing spaces in the string fields of a dataset.

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

5.9.1.12.2.20 TrimFixedChar Property

Specifies whether to discard all trailing spaces in the string fields of a dataset.

Class

[TDADataSetOptions](#)

Syntax

```
property TrimFixedChar: boolean default True;
```

Remarks

Specifies whether to discard all trailing spaces in the string fields of a dataset.

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

5.9.1.12.2.21 UpdateAllFields Property

Used to include all dataset fields in the generated UPDATE and INSERT statements.

Class

[TDADataSetOptions](#)

Syntax

```
property UpdateAllFields: boolean default False;
```

Remarks

If True, all dataset fields will be included in the generated UPDATE and INSERT statements. Unspecified fields will have NULL value in the INSERT statements. Otherwise, only updated fields will be included to the generated update statements.

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

5.9.1.12.2.22 UpdateBatchSize Property

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Class

TDADataSetOptions

Syntax

```
property UpdateBatchSize: Integer default 1;
```

Remarks

Use the UpdateBatchSize property to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch. Takes effect only when updating dataset in the TMemDataSet.CachedUpdates mode. The default value is 1.

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

5.9.1.13 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**

5.9.1.13.1 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**

5.9.1.13.2 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](#)

5.9.1.13.2.1 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](#)

5.9.1.13.2.2 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

5.9.1.14 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.

Request Support   DAC Forum   Provide Feedback

5.9.1.14.1 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](#)

5.9.1.14.2 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](#)

5.9.1.14.2.1 DBLengthMax Property

Maximum DB field length, until which the rule is applied.

Class

[TDAMapRule](#)

Syntax

```
property DBLengthMax 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

5.9.1.14.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class

TDAMapRule

Syntax

```
property DBLengthMin 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

5.9.1.14.2.3 DBScaleMax Property

Maximum DB field scale, until which the rule is applied to the specified DB field.

Class

TDAMapRule

Syntax

```
property DBScaleMax 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

Reserved.

5.9.1.14.2.4 DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class

[TDAMapRule](#)

Syntax

```
property DBScaleMin 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](#)

5.9.1.14.2.5 DBType Property

DB field type, that the rule is applied to.

Class

[TDAMapRule](#)

Syntax

```
property DBType 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](#)

5.9.1.14.2.6 FieldLength Property

The resultant field length in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldLength default rlAny;
```

Remarks

Setting the Delphi field length after conversion.

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

5.9.1.14.2.7 FieldName Property

DataSet field name, for which the rule is applied.

Class

[TDAMapRule](#)

Syntax

```
property FieldName;
```

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](#)

5.9.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldScale default rlAny;
```

Remarks

Setting the Delphi field Scale after conversion.

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

5.9.1.14.2.9 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](#)

5.9.1.14.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class

[TDAMapRule](#)

Syntax

```
property IgnoreErrors 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](#)

5.9.1.15 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);
```

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

5.9.1.15.1 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](#)

5.9.1.15.2 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](#)

5.9.1.15.2.1 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

5.9.1.16 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](#)

5.9.1.16.1 Members

[TDAMetaData](#) class overview.

Properties

Name	Description
[CachedUpdates](#) (inherited from **[TMemDataSet](#)**)	Used to enable or disable the use of cached updates for a dataset.
[Connection](#)	Used to specify a connection object to use to connect to a data store.
[IndexFieldNames](#) (inherited from **[TMemDataSet](#)**)	Used to get or set the list of fields on which the recordset is sorted.
[KeyExclusive](#) (inherited from **[TMemDataSet](#)**)	Specifies the upper and lower boundaries for a range.
[LocalConstraints](#) (inherited from **[TMemDataSet](#)**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
[LocalUpdate](#) (inherited from **[TMemDataSet](#)**)	Used to prevent implicit update of rows on database

	server.
[MetaDataKind](#)	Used to specify which kind of metainformation to show.
[Prepared](#) (inherited from **[TMemDataSet](#)**)	Determines whether a query is prepared for execution or not.
[Ranged](#) (inherited from **[TMemDataSet](#)**)	Indicates whether a range is applied to a dataset.
[Restrictions](#)	Used to provide one or more conditions restricting the list of objects to be described.
[UpdateRecordTypes](#) (inherited from **[TMemDataSet](#)**)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#) (inherited from **[TMemDataSet](#)**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
[ApplyRange](#) (inherited from **[TMemDataSet](#)**)	Applies a range to the dataset.
[ApplyUpdates](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Writes dataset's pending cached updates to a database.
[CancelRange](#) (inherited from **[TMemDataSet](#)**)	Removes any ranges currently in effect for a dataset.
[CancelUpdates](#) (inherited from **[TMemDataSet](#)**)	Clears all pending cached updates from cache and restores dataset in its prior state.
[CommitUpdates](#) (inherited from **[TMemDataSet](#)**)	Clears the cached updates buffer.
[DeferredPost](#) (inherited from **[TMemDataSet](#)**)	Makes permanent changes to the database server.
[EditRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Enables changing the ending value for an existing range.
[EditRangeStart](#) (inherited from **[TMemDataSet](#)**)	Enables changing the starting value for an existing range.

GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds	Used to get values acceptable in the MetaDataKind property.
GetRestrictions	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values

	specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

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

5.9.1.16.2 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](#)

5.9.1.16.2.1 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

5.9.1.16.2.2 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

5.9.1.16.2.3 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**

5.9.1.16.3 Methods

Methods of the **TDAMetaData** class.

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

Public

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

[GetMetaDataKinds](#)	Used to get values acceptable in the MetaDataKind property.
[GetRestrictions](#)	Used to find out which restrictions are applicable to a certain MetaDataKind.
[Locate](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
[LocateEx](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.
[Prepare](#) (inherited from **[TMemDataSet](#)**)	Allocates resources and creates field components for a dataset.
[RestoreUpdates](#) (inherited from **[TMemDataSet](#)**)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#) (inherited from **[TMemDataSet](#)**)	Cancels changes made to the current record when cached updates are enabled.
[SaveToXML](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetRange](#) (inherited from **[TMemDataSet](#)**)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
[UnPrepare](#) (inherited from **[TMemDataSet](#)**)	Frees the resources allocated for a previously

	prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TDAMetaData Class

- TDAMetaData Class Members

Request Support   DAC Forum   Provide Feedback

5.9.1.16.3.1 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

5.9.1.16.3.2 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](#)

5.9.1.17 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](#)

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

5.9.1.17.1 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](#)

5.9.1.17.2 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**

5.9.1.17.2.1 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

5.9.1.17.2.2 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

5.9.1.17.2.3 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

5.9.1.17.2.4 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

5.9.1.17.2.5 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

5.9.1.17.2.6 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

5.9.1.17.2.7 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

5.9.1.17.2.8 AsSQLTimeStamp Property

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

Class

TDAParam

Syntax

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

Request Support   DAC Forum   Provide Feedback

5.9.1.17.2.9 AsString Property

Used to assign the string value to the parameter.

Class

TDAParam

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to the parameter. Setting AsString will set the DataType property to ftString.

Read the AsString property to determine the value that was assigned to an output parameter represented as a string. The value of the parameter will be converted to a string.

Request Support   DAC Forum   Provide Feedback

5.9.1.17.2.10 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

5.9.1.17.2.11 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](#)

5.9.1.17.2.12 IsNull Property

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

Class

[TDAParam](#)

Syntax

```
property IsNull: boolean;
```

Remarks

Use the IsNull property to indicate whether the value assigned to a parameter is NULL.

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

5.9.1.17.2.13 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.

Request Support   DAC Forum   Provide Feedback

5.9.1.17.2.14 Size Property

Specifies the size of a string type parameter.

Class

TDAParam

Syntax

```
property Size: integer default 0;
```

Remarks

Use the Size property to indicate the maximum number of characters the parameter may contain. Use the Size property only for Output parameters of the **ftString, ftFixedChar, ftBytes, ftVarBytes,** or **ftWideString** type.

Request Support   DAC Forum   Provide Feedback

5.9.1.17.2.15 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.

5.9.1.17.3 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

5.9.1.17.3.1 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.

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

5.9.1.17.3.2 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.

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

5.9.1.17.3.3 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](#)

5.9.1.17.3.4 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

5.9.1.17.3.5 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

5.9.1.18 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](#)

5.9.1.18.1 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**

5.9.1.18.2 Properties

Properties of the **TDAParams** class.

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

Public

Name	Description
Items	Used to interate through all parameters.

See Also

- TDAParams Class

- TDAParams Class Members

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

5.9.1.18.2.1 Items Property(Indexer)

Used to interate through all parameters.

Class

TDAParams

Syntax

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

Parameters

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

Remarks

Use the Items property to iterate through all parameters. Index identifies the index in the range

0..Count - 1. Items can reference a particular parameter by its index, but the ParamByName

method is preferred in order to avoid depending on the order of the parameters.

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

5.9.1.18.3 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](#)

5.9.1.18.3.1 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

5.9.1.18.3.2 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

Reserved.

5.9.1.19 TDATransaction Class

A base class that implements functionality for controlling transactions.

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

Unit

[DBAccess](#)

Syntax

```
TDATransaction = class(TComponent);
```

Remarks

TDATransaction is a base class for components implementing functionality for managing transactions.

Do not create instances of TDATransaction. Use descendants of the TDATransaction class instead.

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

5.9.1.19.1 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**

5.9.1.19.2 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

5.9.1.19.2.1 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

5.9.1.19.2.2 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

5.9.1.19.3 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

5.9.1.19.3.1 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

5.9.1.19.3.2 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

5.9.1.19.3.3 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

5.9.1.19.4 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

5.9.1.19.4.1 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.SQLiteDac.TLiteTransaction.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

5.9.1.19.4.2 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

5.9.1.19.4.3 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

5.9.1.19.4.4 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.SQLiteDac.TLiteTransaction.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

5.9.1.19.4.5 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**

5.9.1.20 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**

5.9.1.20.1 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](#)

5.9.1.20.2 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](#)

5.9.1.20.2.1 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

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

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

5.9.1.20.2.2 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](#)

5.9.1.20.2.3 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](#)

5.9.1.20.2.4 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.

5.9.1.20.2.5 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.

5.9.1.20.2.6 Name Property

Used to identify a particular macro.

Class

TMacro

Syntax

```
property Name: string;
```

Remarks

Use the Name property to identify a particular macro.

5.9.1.20.2.7 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.

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.9.1.21 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

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.9.1.21.1 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

5.9.1.21.2 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

5.9.1.21.2.1 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

5.9.1.21.3 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](#)

5.9.1.21.3.1 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](#)

Reserved.

5.9.1.21.3.2 Expand Method

Changes the macros in the passed SQL statement to their values.

Class

TMacros

Syntax

```
procedure Expand(var SQL: string);
```

Parameters

SQL
  Holds the passed SQL statement.

Remarks

Call the Expand method to change the macros in the passed SQL statement to their values.

Request Support   DAC Forum   Provide Feedback

5.9.1.21.3.3 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

5.9.1.21.3.4 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

5.9.1.21.3.5 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

5.9.1.21.3.6 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.

5.9.1.22 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);
```

5.9.1.22.1 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**

5.9.1.22.2 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**

5.9.1.22.2.1 ConnectionLifetime Property

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

Class

TPoolingOptions

Syntax

```
property ConnectionLifetime: integer default
DefValConnectionLifetime;
```

Remarks

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

Request Support   DAC Forum   Provide Feedback

5.9.1.22.2.2 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

Reserved.

5.9.1.22.2.3 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](#)

5.9.1.22.2.4 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](#)

5.9.1.22.2.5 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

5.9.1.23 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

5.9.1.23.1 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**

5.9.1.23.2 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**

5.9.1.23.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

Class

TSmartFetchOptions

Syntax

```
property Enabled: Boolean default False;
```

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

5.9.1.23.2.2 LiveBlock Property

Used to minimize memory consumption.

Class

TSmartFetchOptions

Syntax

```
property LiveBlock: Boolean default True;
```

Remarks

If LiveBlock is True, then on navigating through a dataset forward or backward, memory will be allocated for records count defined in the the FetchRows propety, and no additional memory will be allocated. But if you return records that were read from the database before, they will be read from the database again, because when you left block with these records, memory was free. So the LiveBlock mode minimizes memory consumption, but can decrease performance, because it can lead to repeated data reading from the database.

The default value of LiveBlock is False.

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

5.9.1.23.2.3 PrefetchedFields Property

List of fields additional to key fields that will be read from the database on dataset open.

Class

TSmartFetchOptions

Syntax

```
property PrefetchedFields: string;
```

Remarks

If you are going to use locate, filter or sort by some fields, then these fields should be added to the prefetched fields list to avoid excessive reading from the database.

    Request Support   DAC Forum   Provide Feedback

5.9.1.23.2.4 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

5.9.2 Types

Types in the **DBAccess** unit.

Types

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecute events.
TAfterFetchEvent	This type is used for the TCustomDADataSet.AfterFetch event.
TBeforeFetchEvent	This type is used for the TCustomDADataSet.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDAConnection.OnConnectionLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDAConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption.
TUpdateExecuteEvent	This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.

© 1997-2024 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

5.9.2.1 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

5.9.2.2 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

5.9.2.3 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

5.9.2.4 TConnectionLostEvent Procedure Reference

This type is used for the TCustomDAConnection.OnConnectionLost event.

Unit

DBAccess

Syntax

```
TConnectionLostEvent = procedure (Sender: TObject; Component:
TComponent; ConnLostCause: TConnLostCause; var RetryMode:
TRetryMode) of object;
```

Parameters

Sender

 An object that raised the event.

Component

ConnLostCause

 The reason of the connection loss.

RetryMode

 The application behavior when connection is lost.

Request Support   DAC Forum   Provide Feedback

5.9.2.5 TDAConnectionErrorEvent Procedure Reference

This type is used for the TCustomDAConnection.OnError event.

Unit

DBAccess

Syntax

```
TDAConnectionErrorEvent = procedure (Sender: TObject; E: EDAError;
var Fail: boolean) of object;
```

Parameters

Sender

An object that raised the event.

E

The error information.

Fail

False, if an error dialog should be prevented from being displayed and EAbort exception should be raised to cancel current operation .

Request Support   DAC Forum   Provide Feedback

5.9.2.6 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

5.9.2.7 TRefreshOptions Set

Represents the set of TRefreshOption.

Unit

DBAccess

Syntax

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

Request Support   DAC Forum   Provide Feedback

5.9.2.8 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

Reserved.

5.9.3 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](#)

5.9.3.1 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

5.9.3.2   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

5.9.3.3   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

5.9.3.4 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.

Request Support   DAC Forum   Provide Feedback

5.9.4 Variables

Variables in the **DBAccess** unit.

Variables

Name	Description

	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.
[BaseSQLOldBehavior](#)	
[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**

5.9.4.1   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 LiteDAC . To restore old

behavior, set the BaseSQLOldBehavior variable to True.

Request Support DAC Forum Provide Feedback

5.9.4.2 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

5.9.4.3 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, LiteDAC 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 LiteDAC . To restore the old behavior,

set the BaseSQLOldBehavior variable to True.

Request Support   DAC Forum   Provide Feedback

5.10   LiteAccess

This unit contains main components of LiteDAC

Classes

Name	Description
[TCustomLiteDataSet](#)	A base class that defines SQLite functionality for a dataset.
[TCustomLiteTable](#)	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
[TCustomLiteUserCollation](#)	A base class that provides SQLite functionality for working with user-defined collations.
[TCustomLiteUserFunction](#)	A base class that defines SQLite functionality for working with user-defined functions.
[TLiteBackup](#)	Implements SQLite Online Backup API functionality for performing backup (copying) the contents of one database into another database, overwriting the original content of the target database.
[TLiteConnection](#)	Represents an open connection to a SQLite database.
[TLiteConnectionOptions](#)	This class allows setting up the behaviour of the TLiteConnection class.
[TLiteDataSetOptions](#)	This class allows setting up the behaviour of the

	TCustomLiteDataSet class.
[TLiteDataSource](#)	TLiteDataSource provides an interface between a LiteDAC dataset components and data-aware controls on a form.
[TLiteEncryptor](#)	The class that performs encrypting and decrypting of data.
[TLiteMetaData](#)	A component for obtaining metainformation about database objects.
[TLiteQuery](#)	A component for executing queries and operating record sets. It also provides flexible way to update data.
[TLiteSQL](#)	A component for executing SQL statements.
[TLiteTable](#)	Accesses data in a single table without writing SQL statements.
[TLiteUpdateSQL](#)	Lets you tune update operations for the DataSet component.
[TLiteUserCollation](#)	A base class that provides SQLite functionality for working with user-defined collations.
[TLiteUserFunction](#)	A component for defining a custom function for future use in SQL-statements. Also, can be used for overriding built-in SQLite functions.

Types

Name	Description
[TLiteCollationEvent](#)	This type is used for the [TCustomLiteUserCollation.OnCollate](#) event.
[TLiteFunctionExecuteEvent](#)	This type is used for the [TCustomLiteUserFunction.OnExecute](#) event.

[TLiteFunctionFinalEvent](#)	This type is used for the [TCustomLiteUserFunction.OnFinal](#) event.
[TLiteFunctionStepEvent](#)	This type is used for the TCustomLiteUserFunction.OnStep event.

Enumerations

Name	Description
[TLiteFunctionKind](#)	Specifies the kind of the used-defined function.
[TLiteIsolationLevel](#)	Specifies the way the transactions containing database modifications are handled.

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

5.10.1 Classes

Classes in the **LiteAccess** unit.

Classes

Name	Description
[TCustomLiteDataSet](#)	A base class that defines SQLite functionality for a dataset.
[TCustomLiteTable](#)	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
[TCustomLiteUserCollation](#)	A base class that provides SQLite functionality for working with user-defined collations.
[TCustomLiteUserFunction](#)	A base class that defines SQLite functionality for working with user-defined functions.

[TLiteBackup](#)	Implements SQLite Online Backup API functionality for performing backup (copying) the contents of one database into another database, overwriting the original content of the target database.
[TLiteConnection](#)	Represents an open connection to a SQLite database.
[TLiteConnectionOptions](#)	This class allows setting up the behaviour of the TLiteConnection class.
[TLiteDataSetOptions](#)	This class allows setting up the behaviour of the TCustomLiteDataSet class.
[TLiteDataSource](#)	TLiteDataSource provides an interface between a LiteDAC dataset components and data-aware controls on a form.
[TLiteEncryptor](#)	The class that performs encrypting and decrypting of data.
[TLiteMetaData](#)	A component for obtaining metainformation about database objects.
[TLiteQuery](#)	A component for executing queries and operating record sets. It also provides flexible way to update data.
[TLiteSQL](#)	A component for executing SQL statements.
[TLiteTable](#)	Accesses data in a single table without writing SQL statements.
[TLiteUpdateSQL](#)	Lets you tune update operations for the DataSet component.
[TLiteUserCollation](#)	A base class that provides SQLite functionality for working with user-defined collations.

	A component for defining a custom function for future use in SQL-statements. Also, can be used for overriding built-in SQLite functions.
[TLiteUserFunction](#)	

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

5.10.1.1 TCustomLiteDataSet Class

A base class that defines SQLite functionality for a dataset.

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

Unit

[LiteAccess](#)

Syntax

```
TCustomLiteDataSet = class(TCustomDADataSet);
```

Remarks

TCustomLiteDataSet is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomLiteDataSet objects directly. Instead they use descendants of TCustomLiteDataSet, such as [TLiteQuery](#) and [TLiteTable](#) that inherit its dataset-related properties and methods.

Inheritance Hierarchy

[TMemDataSet](#)

  [TCustomDADataSet](#)

    TCustomLiteDataSet

See Also

- [TLiteQuery](#)

- [TCustomLiteTable](#)

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

Devart. All Rights
Reserved.

5.10.1.1.1 Members

TCustomLiteDataSet 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.
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.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT

	statement and reopen a query.
[FinalSQL](#) (inherited from **[TCustomDADataSet](#)**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
[IndexFieldNames](#) (inherited from **[TMemDataSet](#)**)	Used to get or set the list of fields on which the recordset is sorted.
[IsQuery](#) (inherited from **[TCustomDADataSet](#)**)	Used to check whether SQL statement returns rows.
[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	Specifies the behaviour of TCustomLiteDataSet 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.
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.
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.
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 **TCustomDADataSet**)	Locks the current record.

MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
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.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to

	the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

Reserved.

5.10.1.1.2 Properties

Properties of the **TCustomLiteDataSet** class.

For a complete list of the **TCustomLiteDataSet** class members, see the
TCustomLiteDataSet 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.
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.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.

FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from **TCustomDADataSet**)	Used to check whether SQL statement returns rows.
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	Specifies the behaviour of TCustomLiteDataSet 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.
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

- TCustomLiteDataSet Class

- TCustomLiteDataSet Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.1.2.1 Connection Property

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

Class

[TCustomLiteDataSet](#)

Syntax

```
property Connection: TLiteConnection;
```

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 [TLiteConnection](#) objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

See Also

- [TLiteConnection](#)

© 1997-2024 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.10.1.1.2.2 Encryption Property

Used to specify encryption options in a dataset.

Class

[TCustomLiteDataSet](#)

Syntax

```
property Encryption: TLiteEncryption;
```

Remarks

Set the Encryption options for using encryption in a dataset.

© 1997-2024 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.10.1.1.2.3 Options Property

Specifies the behaviour of TCustomLiteDataSet object.

Class

[TCustomLiteDataSet](#)

Syntax

```
property Options: TLiteDataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a [TCustomLiteDataSet](#) object.

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

5.10.1.1.2.4 SmartFetch Property

The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.

Class

[TCustomLiteDataSet](#)

Syntax

```
property SmartFetch: TSmartFetchOptions;
```

See Also

- [TSmartFetchOptions](#)

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

5.10.1.2 TCustomLiteTable 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 [TCustomLiteTable](#) members.

Unit

[LiteAccess](#)

Syntax

```
TCustomLiteTable = class(TCustomLiteDataSet);
```

Remarks

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

Inheritance Hierarchy

TMemDataSet

  TCustomDADataSet

    TCustomLiteDataSet

      TCustomLiteTable

See Also

- **TLiteTable**

- **TLiteQuery**

- **TCustomLiteDataSet**

Request Support   DAC Forum   Provide Feedback

5.10.1.2.1 Members

TCustomLiteTable 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.

Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomLiteDataSet**)	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 **TCustomLiteDataSet**)	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.
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 table.
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	Used to allow retrieving data from the database starting from the specified row.
Options (inherited from **TCustomLiteDataSet**)	Specifies the behaviour of TCustomLiteDataSet 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 **TCustomLiteDataSet**)	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.
[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.
[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.
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 **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
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.
[RefreshRecord](#) (inherited from **[TCustomDADataSet](#)**)	Actualizes field values for the current record.
[RestoreSQL](#) (inherited from **[TCustomDADataSet](#)**)	Restores the SQL property modified by AddWhere and SetOrderBy.
[RestoreUpdates](#) (inherited from **[TMemDataSet](#)**)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#) (inherited from **[TMemDataSet](#)**)	Cancels changes made to the current record when cached updates are enabled.
[SaveSQL](#) (inherited from **[TCustomDADataSet](#)**)	Saves the SQL property value to BaseSQL.
[SaveToXML](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetOrderBy](#) (inherited from **[TCustomDADataSet](#)**)	Builds an ORDER BY clause of a SELECT statement.
[SetRange](#) (inherited from **[TMemDataSet](#)**)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
[SQLSaved](#) (inherited from **[TCustomDADataSet](#)**)	Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.
[UnLock](#) (inherited from **[TCustomDADataSet](#)**)	Releases a record lock.
[UnPrepare](#) (inherited from **[TMemDataSet](#)**)	Frees the resources allocated for a previously

	prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

5.10.1.2.2 Properties

Properties of the **TCustomLiteTable** class.

For a complete list of the **TCustomLiteTable** class members, see the TCustomLiteTable 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.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomLiteDataSet**)	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 **TCustomLiteDataSet**)	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.
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 table.
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	Used to allow retrieving data from the database starting from the specified row.
Options (inherited from **TCustomLiteDataSet**)	Specifies the behaviour of TCustomLiteDataSet 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 **TCustomLiteDataSet**)	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.
[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

- [TCustomLiteTable Class](#)

- [TCustomLiteTable Class Members](#)

5.10.1.2.2.1 Limit Property

Used to set the number of rows retrieved from the table.

Class

`TCustomLiteTable`

Syntax

```
property Limit: integer default - 1;
```

Remarks

Use the Limit property to set the number of rows retrieved from the table. If Limit is -1, all records will be obtained. The default value is -1.

See Also

- Offset

5.10.1.2.2.2 Offset Property

Used to allow retrieving data from the database starting from the specified row.

Class

`TCustomLiteTable`

Syntax

```
property Offset: integer default 0;
```

Remarks

Use the Offset property to allow retrieving data from the database starting from the specified row. The default value is 0.

See Also

- Limit

Request Support   DAC Forum   Provide Feedback

5.10.1.2.2.3 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

`TCustomLiteTable`

Syntax

```
property OrderFields: string;
```

Remarks

TCustomLiteTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas. TCustomLiteTable is reopened when OrderFields is being changed.

See Also

- TCustomLiteTable

Request Support   DAC Forum   Provide Feedback

5.10.1.2.2.4 TableName Property

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

Class

`TCustomLiteTable`

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to specify the name of the database table this component encapsulates. If TCustomLiteDataSet.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

5.10.1.3 TCustomLiteUserCollation Class

A base class that provides SQLite functionality for working with user-defined collations.

For a list of all members of this type, see TCustomLiteUserCollation members.

Unit

LiteAccess

Syntax

```
TCustomLiteUserCollation = class(TComponent);
```

Remarks

TCustomLiteUserCollation is a base component that provides functionality for classes derived from it. Applications never use TCustomLiteUserCollation objects directly. Instead they use TLiteUserCollation that inherit its properties and methods.

Request Support   DAC Forum   Provide Feedback

5.10.1.3.1 Members

TCustomLiteUserCollation class overview.

Properties

Name	Description
CollationName	Used to specify the name of the collation.
Connection	Used to specify a connection object that will be

	used to connect to a data store.

Events

Name	Description
OnCollate	Occurs when the collation is called in the SQL-statement.

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

5.10.1.3.2 Properties

Properties of the **TCustomLiteUserCollation** class.

For a complete list of the **TCustomLiteUserCollation** class members, see the

TCustomLiteUserCollation Members topic.

Public

Name	Description
CollationName	Used to specify the name of the collation.
Connection	Used to specify a connection object that will be used to connect to a data store.

See Also

- **TCustomLiteUserCollation Class**

- **TCustomLiteUserCollation Class Members**

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

5.10.1.3.2.1 CollationName Property

Used to specify the name of the collation.

Class

TCustomLiteUserCollation

Syntax

```
property CollationName: string;
```

Remarks

Use the CollationName property to specify the name of the collation. The name is case-insensitive.

Request Support   DAC Forum   Provide Feedback

5.10.1.3.2.2 Connection Property

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

Class

TCustomLiteUserCollation

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

Request Support   DAC Forum   Provide Feedback

5.10.1.3.3 Events

Events of the **TCustomLiteUserCollation** class.

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

Public

Name	Description
OnCollate	Occurs when the collation is called in the SQL-statement.

See Also

- TCustomLiteUserCollation Class

- TCustomLiteUserCollation Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.3.3.1 OnCollate Event

Occurs when the collation is called in the SQL-statement.

Class

`TCustomLiteUserCollation`

Syntax

```
property OnCollate: TLiteCollationEvent;
```

Remarks

Use the OnCollate event handler to define the implementation of the collating function. The collating function must return an integer that is negative, zero, or positive if the first string is less than, equal to, or greater than the second, respectively.

Request Support   DAC Forum   Provide Feedback

5.10.1.4 TCustomLiteUserFunction Class

A base class that defines SQLite functionality for working with user-defined functions.

For a list of all members of this type, see TCustomLiteUserFunction members.

Unit

`LiteAccess`

Syntax

```
TCustomLiteUserFunction = class(TComponent);
```

Remarks

TCustomLiteUserFunction is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomLiteUserFunction objects directly. Instead they use TLiteUserFunction that inherit its properties and methods.

See Also

- TLiteUserFunction

Request Support   DAC Forum   Provide Feedback

5.10.1.4.1 Members

TCustomLiteUserFunction class overview.

Properties

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
FunctionKind	Used to specify the kind of the function that will be defined.
FunctionName	Used to specify the name of the function that will be defined.
Params	Used to specify the list of function parameters.

Events

Name	Description
OnExecute	Occurs when the user function is called in the SQL-statement.

[OnFinal](#)	Used to return a result of the user-defined aggregate function in the SQL statement.
[OnStep](#)	Occurs during execution of the user-defined aggregate function in the SQL statement.

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

5.10.1.4.2 Properties

Properties of the **TCustomLiteUserFunction** class.

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

Public

Name	Description
[Connection](#)	Used to specify a connection object that will be used to connect to a data store.
[FunctionKind](#)	Used to specify the kind of the function that will be defined.
[FunctionName](#)	Used to specify the name of the function that will be defined.
[Params](#)	Used to specify the list of function parameters.

See Also

- [TCustomLiteUserFunction Class](#)

- [TCustomLiteUserFunction Class Members](#)

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

5.10.1.4.2.1 Connection Property

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

Class

`TCustomLiteUserFunction`

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

See Also

* TLiteConnection

Request Support   DAC Forum   Provide Feedback

5.10.1.4.2.2 FunctionKind Property

Used to specify the kind of the function that will be defined.

Class

`TCustomLiteUserFunction`

Syntax

```
property FunctionKind: TLiteFunctionKind default fkScalar;
```

Remarks

Use the FunctionKind property to specify the kind of the function that will be defined for future use in SQL statements. The list of function parameters can be set using the Params property. When fkScalar, the implementation of the function has to be defined in the OnExecute event handler. When fkAggregate, the implementation of the function has to be defined in the OnStep event handler, and a result has to be passed back in the OnFinal event

handler. Default value is fkScalar.

See Also

- FunctionName

- Params

Request Support   DAC Forum   Provide Feedback

5.10.1.4.2.3 FunctionName Property

Used to specify the name of the function that will be defined.

Class

TCustomLiteUserFunction

Syntax

```
property FunctionName: string;
```

Remarks

Use the FunctionName property to specify the name of the function that will be defined for future use in SQL statements. The list of function parameters can be set using the Params property. The implementation of the function has to be defined in the OnExecute event handler.

See Also

- Params

- FunctionKind

Request Support   DAC Forum   Provide Feedback

5.10.1.4.2.4 Params Property

Used to specify the list of function parameters.

Class

TCustomLiteUserFunction

Syntax

```
property Params: TLiteUserFunctionParams;
```

Remarks

At design-time use the Parameters editor to set the list of parameters of the function. Use the Params property in the OnExecute event handler at runtime to get parameter values that were passed to the function. 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

- OnExecute

Request Support   DAC Forum   Provide Feedback

5.10.1.4.3 Events

Events of the **TCustomLiteUserFunction** class.

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

Public

Name	Description
OnExecute	Occurs when the user function is called in the SQL-statement.
OnFinal	Used to return a result of the user-defined aggregate function in the SQL statement.
OnStep	Occurs during execution of the user-defined aggregate function in the SQL statement.

See Also
- [TCustomLiteUserFunction Class](#)
- [TCustomLiteUserFunction Class Members](#)

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

5.10.1.4.3.1 OnExecute Event

Occurs when the user function is called in the SQL-statement.

Class

`TCustomLiteUserFunction`

Syntax

```
property OnExecute: TLiteFunctionExecuteEvent;
```

Remarks

Use the OnExecute event handler to define the implementation of the function. Parameters that were passed to the function when it was called in an SQL statement, are accessible through the Params property. The resulting value of the function has to be passed back using the ResultValue variable.

See Also
- [Params](#)
- [OnExecute](#)

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

5.10.1.4.3.2 OnFinal Event

Used to return a result of the user-defined aggregate function in the SQL statement.

Class

`TCustomLiteUserFunction`

Syntax

```
property OnFinal: TLiteFunctionFinalEvent;
```

Remarks

Use the OnFinal event handler to return a result of the function. The resulting value of the function has to be passed back using the ResultValue variable.

See Also

- OnStep

Request Support   DAC Forum   Provide Feedback

5.10.1.4.3.3 OnStep Event

Occurs during execution of the user-defined aggregate function in the SQL statement.

Class

TCustomLiteUserFunction

Syntax

```
property OnStep: TLiteFunctionStepEvent;
```

Remarks

Use the OnStep event handler to define the implementation of the function. Parameters that were passed to the function when it was executed in a SQL-statement, are accessible through the Params argument. The resulting value of the function has to be passed back using the OnFinal event handler.

See Also

- OnFinal

Request Support   DAC Forum   Provide Feedback

5.10.1.5 TLiteBackup Class

Implements SQLite Online Backup API functionality for performing backup (copying) the contents of one database into another database, overwriting the original content of the target database.

For a list of all members of this type, see TLiteBackup members.

Unit

LiteAccess

Syntax

```
TLiteBackup = class(TComponent);
```

Request Support   DAC Forum   Provide Feedback

5.10.1.5.1 Members

TLiteBackup class overview.

Properties

Name	Description
DestinationConnection	Specifies the destination connection for the backup.
DestinationDatabaseName	Specifies the name of the destination database.
PagesPerStep	Specifies the number of database pages to be copied in one step.
SourceConnection	Specifies the source connection for the backup.
SourceDatabaseName	Specifies the name of the source database.
WaitDelay	Specifies the wait delay during which the backup process waits until the source database lock is released.
WaitTimeout	Specifies the total wait time for which the backup

	process waits until the source database lock is released.
[WaitWhenLocked](#)	Specifies, whether the backup process waits until the source database becomes unlocked.

Methods

Name	Description
[Backup](#)	Starts the backup process

Events

Name	Description
[OnProgress](#)	Occurs during the backup process after copying a portion of database pages.

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

5.10.1.5.2 Properties

Properties of the **TLiteBackup** class.

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

Published

Name	Description
[DestinationConnection](#)	Specifies the destination connection for the backup.
[DestinationDatabaseName](#)	Specifies the name of the destination database.
[PagesPerStep](#)	Specifies the number of database pages to be copied in one step.
[SourceConnection](#)	Specifies the source connection for the backup.

SourceDatabaseName	Specifies the name of the source database.
WaitDelay	Specifies the wait delay during which the backup process waits until the source database lock is released.
WaitTimeout	Specifies the total wait time for which the backup process waits until the source database lock is released.
WaitWhenLocked	Specifies, whether the backup process waits until the source database becomes unlocked.

See Also

- TLiteBackup Class

- TLiteBackup Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.5.2.1 DestinationConnection Property

Specifies the destination connection for the backup.

Class

TLiteBackup

Syntax

```
property DestinationConnection: TLiteConnection;
```

Remarks

Set the DestinationConnection property to specify the destination TLiteConnection for the backup (the connection to the target database). Read the property value to access the destination connection.

The actual name of the target database is specified using the DestinationDatabaseName property.

The source connection (the connection to the database to be copied) for the backup is accessible using the SourceConnection property.

Both source and destination connection have to be specified and connected to perform a backup. Otherwise, the appropriate exception will be raised.

See Also

- TLiteConnection

- DestinationDatabaseName

- SourceConnection

Request Support   DAC Forum   Provide Feedback

5.10.1.5.2.2 DestinationDatabaseName Property

Specifies the name of the destination database.

Class

`TLiteBackup`

Syntax

```
property DestinationDatabaseName: string;
```

Remarks

Set the DestinationDatabaseName property to specify the name of the destination (target) database. Read the property value to determine the destination database name.

The database name is 'main' or an empty string for the main database, 'temp' for the temporary database, or the name specified after the AS keyword in an ATTACH statement for an attached database. The default value is an empty string.

The name of the source database for the backup is accessible using the SourceDatabaseName property.

See Also
- SourceDatabaseName

Request Support   DAC Forum   Provide Feedback

5.10.1.5.2.3 PagesPerStep Property

Specifies the number of database pages to be copied in one step.

Class

TLiteBackup

Syntax

```
property PagesPerStep: integer default - 1;
```

Remarks

Set the PagesPerStep property to specify the number of database pages to be copied between the source and destination databases in one step of the backup process. If there are still more pages to be copied, then the OnProgress event is triggered, and the backup process continues. Otherwise, the backup process is finished. The copying process is described in detail in the Backup article.

The default property value is -1, which means that the entire database will be copied at once.

See Also
- OnProgress
- Backup

Request Support   DAC Forum   Provide Feedback

5.10.1.5.2.4 SourceConnection Property

Specifies the source connection for the backup.

Class

TLiteBackup

Syntax

```
property SourceConnection: TLiteConnection;
```

Remarks

Set the SourceConnection property to specify the source TLiteConnection for the backup (the connection to the database to be copied). Read the property value to access the source connection.

The actual name of the database to be copied is specified using the SourceDatabaseName property.

The destination connection (the connection to the target database) for the backup is accessible using the DestinationConnection property.

Both source and destination connection have to be specified and connected to perform a backup. Otherwise, the appropriate exception will be raised.

See Also

- DestinationConnection

- SourceDatabaseName

- TLiteConnection

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

5.10.1.5.2.5 SourceDatabaseName Property

Specifies the name of the source database.

Class

TLiteBackup

Syntax

```
property SourceDatabaseName: string;
```

Remarks

Set the SourceDatabaseName property to specify the name of the source database (the database to be copied). Read the property value to determine the source database name.

The database name is 'main' or an empty string for the main database, 'temp' for the temporary database, or the name specified after the AS keyword in an ATTACH statement for an attached database. The default value is an empty string.

The name of the destination database for the backup is accessible using the DestinationDatabaseName property.

See Also

- DestinationDatabaseName

Request Support   DAC Forum   Provide Feedback

5.10.1.5.2.6 WaitDelay Property

Specifies the wait delay during which the backup process waits until the source database lock is released.

Class

`TLiteBackup`

Syntax

```
property WaitDelay: integer default 250;
```

Remarks

Set the WaitDelay property value to specify the number of milliseconds during which the backup process waits until the source database lock is released.

When during the backup process the source database becomes locked by an outside connection, then if the WaitWhenLocked property is set to True, the backup process waits for WaitDelay milliseconds until WaitTimeout is reached. If the source database lock is released, the backup process continues. Otherwise, the appropriate exception is raised.

The default value is 250.

See Also

- [WaitWhenLocked](#)

- [WaitTimeout](#)

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

5.10.1.5.2.7 WaitTimeout Property

Specifies the total wait time for which the backup process waits until the source database lock is released.

Class

`TLiteBackup`

Syntax

```
property WaitTimeout: integer default 0;
```

Remarks

Set the WaitTimeout property value to specify the total number of milliseconds during which the backup process waits until the source database lock is released. When during the backup process the source database becomes locked by an outside connection, then if the [WaitWhenLocked](#) property is set to True, the backup process waits for [WaitDelay](#) milliseconds until WaitTimeout is reached. If the source database lock is released, the backup process continues. Otherwise, the appropriate exception is raised. The default value is 0.

See Also

- [WaitDelay](#)

- [WaitWhenLocked](#)

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

5.10.1.5.2.8 WaitWhenLocked Property

Specifies, whether the backup process waits until the source database becomes unlocked.

Class

[TLiteBackup](#)

Syntax

```
property WaitWhenLocked: boolean default True;
```

Remarks

Set the WaitWhenLocked property value to specify, whether the backup process will wait until the source database becomes unlocked, if it is locked by an outside connection.

When the property is True, the backup process waits for [WaitDelay](#) milliseconds until [WaitTimeout](#) is reached. If the source database lock is released, the backup process continues. Otherwise, the appropriate exception is raised.

When the property is False, the backup process finishes immediately.

The default value is True.

See Also

- [WaitDelay](#)

- [WaitTimeout](#)

© 1997-2024
Devart. All Rights
Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.10.1.5.3 Methods

Methods of the **TLiteBackup** class.

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

Public

Name	Description
[Backup](#)	Starts the backup process

See Also

- TLiteBackup Class

- TLiteBackup Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.5.3.1 Backup Method

Starts the backup process

Class

`TLiteBackup`

Syntax

```
procedure Backup;
```

Remarks

Use the method to starts the process of copying the source database into the destination database.

Before starting the backup process, the connection to the source database has to be specified in the SourceConnection property, and the source database name in the SourceDatabaseName property. The connection to the destination database has to be specified in the DestinationConnection property, and the destination database name in the DestinationDatabaseName property.

Both source and destination connection have to be connected to perform a backup. Otherwise, the appropriate exception will be raised.

When the backup is started, the backup mechanism tries to copy up to PagesPerStep database pages between the source and destination databases. If there are still more pages to be copied, then the OnProgress event is triggered, and the backup process continues. Otherwise, the backup process is finished.

When during the backup process the source database becomes locked by an outside connection, then if the WaitWhenLocked property is set to True, the backup process waits

for WaitDelay milliseconds until WaitTimeout is reached. If the source database lock is released, the backup process continues. Otherwise, the appropriate exception is raised.

See Also

- SourceConnection

- SourceDatabaseName

- DestinationDatabaseName

- PagesPerStep

- WaitWhenLocked

- WaitDelay

- WaitTimeout

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

5.10.1.5.4 Events

Events of the **TLiteBackup** class.

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

Published

Name	Description
OnProgress	Occurs during the backup process after copying a portion of database pages.

See Also

- TLiteBackup Class

- TLiteBackup Class Members

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

5.10.1.5.4.1 OnProgress Event

Occurs during the backup process after copying a portion of database pages.

Class

`TLiteBackup`

Syntax

```
property OnProgress: TLiteBackupProgressEvent;
```

Remarks

Write the OnProgress event handler to be able to respond on the backup process progress. When the backup is started, the backup mechanism tries to copy up to PagesPerStep database pages between the source and destination databases. If there are still more pages to be copied, then the OnProgress event is triggered, and the backup process continues. Otherwise, the backup process is finished. The copying process is described in detail in the Backup article.

Parameters

Sender

The TLiteBackup component that triggers the event

PagesTotal

Total number of pages in the source database

PagesRemaining

Number of pages left to copy

See Also

- PagesPerStep

- Backup

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.10.1.6 TLiteConnection Class

Represents an open connection to a SQLite database.

For a list of all members of this type, see TLiteConnection members.

Unit

LiteAccess

Syntax

```
TLiteConnection = class(TCustomDAConnection);
```

Remarks

The TLiteConnection component is used to maintain connection to the SQLite database. After setting the Database and EncryptionKey (if needed) properties, you can establish a connection to the database by calling Open or Connect methods or setting the Connected property to True. All components which are dedicated to perform data access, such as TLiteTable, TLiteQuery, TLiteSQL, must have their Connection property assigned with one of the TLiteConnection instances.

Inheritance Hierarchy

TCustomDAConnection

  TLiteConnection

Request Support   DAC Forum   Provide Feedback

5.10.1.6.1 Members

TLiteConnection class overview.

Properties

Name	Description
ClientLibrary	Used to set or get the SQLite client library location.
ClientLibraryVersion	Indicates the version of the SQLite3 client library.

[ConnectDialog](#)	Allows to link a TCustomConnectDialog component.
[Connected](#)	Used to establish a database 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 name of the database to be used once a connection is open.
[Debug](#)	Used to display SQL statements being executed with their parameter values and data types.
[EncryptionKey](#)	Used to specify the encryption key for working with an encrypted database.
[InTransaction](#) (inherited from **[TCustomDAConnection](#)**)	Indicates whether the transaction is active.
[LoginPrompt](#)	Specifies whether a login dialog appears immediately before opening a new connection.
[Options](#)	Used to specify the behaviour of the TLiteConnection object.
[Pooling](#)	Enables or disables using connection pool.
[PoolingOptions](#)	Specifies the behaviour of connection pool.

Methods

Name	Description
[ApplyUpdates](#) (inherited from **[TCustomDAConnection](#)**)	Overloaded. Applies changes in datasets.
[Commit](#) (inherited from **[TCustomDAConnection](#)**)	Commits current transaction.

Connect (inherited from **TCustomDAConnection**)	Establishes a connection to the server.
CreateDataSet	Returns a new instance of TLiteQuery class and associates it with the connection object.
CreateMetaData	Returns a new instance of TLiteMetaData class and associates it with the connection object.
CreateSQL	Returns a new instance of TLiteSQL class and associates it with the connection object.
CreateTransaction	Returns a new instance of TLiteTransaction class and associates it with the connection object.
Disconnect (inherited from **TCustomDAConnection**)	Performs disconnect.
EncryptDatabase	Encrypt the connected database with the encryption key specified.
GetKeyFieldNames (inherited from **TCustomDAConnection**)	Provides a list of available key field names.
GetTableNames (inherited from **TCustomDAConnection**)	Provides a list of available tables names.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
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](#)	Overloaded. Begins a new user transaction.

Events

Name	Description
[OnConnectionLost](#)	This event occurs when connection was lost.
[OnError](#)	This event occurs when an error has arisen in the connection.

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

5.10.1.6.2 Properties

Properties of the **TLiteConnection** class.

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

Public

Name	Description
[ClientLibraryVersion](#)	Indicates the version of the SQLite3 client library.
[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.

Published

Name	Description
ClientLibrary	Used to set or get the SQLite client library location.
ConnectDialog	Allows to link a TCustomConnectDialog component.
Connected	Used to establish a database connection.
Database	Used to specify the name of the database to be used once a connection is open.
Debug	Used to display SQL statements being executed with their parameter values and data types.
EncryptionKey	Used to specify the encryption key for working with an encrypted database.
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Used to specify the behaviour of the TLiteConnection object.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.

See Also

- TLiteConnection Class

- TLiteConnection Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.6.2.1 ClientLibrary Property

Used to set or get the SQLite client library location.

Class

[TLiteConnection](#)

Syntax

```
property ClientLibrary: string;
```

Remarks

Use the ClientLibrary property to manually set the SQLite client library location. When the ClientLibrary property is empty, TLiteConnection tries to find the client library on the system path. When working in Direct mode, the ClientLibrary property is not used. For more information about using direct access in LiteDAC, read the Working In Direct Mode article.

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

5.10.1.6.2.2 ClientLibraryVersion Property

Indicates the version of the SQLite3 client library.

Class

[TLiteConnection](#)

Syntax

```
property ClientLibraryVersion: string;
```

Remarks

The ClientLibraryVersion property is used to indicate the version of the SQLite3 client library. When working in the Direct mode, the property indicates the version of the SQLite3 engine that integrated in LiteDAC.

See Also

- [TLiteConnectionOptions.Direct](#)

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

5.10.1.6.2.3 ConnectDialog Property

Allows to link a TCustomConnectDialog component.

Class

`TLiteConnection`

Syntax

```
property ConnectDialog: TCustomConnectDialog;
```

Remarks

Use the ConnectDialog property to assign to connection a TCustomConnectDialog component.

See Also

- TCustomDAConnection.ConnectDialog

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

5.10.1.6.2.4 Connected Property

Used to establish a database connection.

Class

`TLiteConnection`

Syntax

```
property Connected;
```

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**

Reserved.

5.10.1.6.2.5 Database Property

Used to specify the name of the database to be used once a connection is open.

Class

`TLiteConnection`

Syntax

```
property Database: string;
```

Remarks

Use the Database property to specify the name of the database to be used once a connection is open.

If the Database property points to an existent database, the database will be opened.

If the Database property points to a non-existent database in a correct system path, further behavior depends on the value of the TLiteConnectionOptions.ForceCreateDatabase property.

If the Database property is set to ":memory:", a new temporary in-memory database will be created and opened.

If the Database property is empty, a new temporary on-disk database will be created and opened.

See Also

- TLiteConnectionOptions.ForceCreateDatabase

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

5.10.1.6.2.6 Debug Property

Used to display SQL statements being executed with their parameter values and data types.

Class

`TLiteConnection`

Syntax

```
property Debug: boolean;
```

Remarks

Set the Debug property to True to display SQL statements being executed with their parameter values and data types. Note: To use this property you should explicitly include unit LiteDacVcl (LiteDacClx under Linux) to your project.

Request Support   DAC Forum   Provide Feedback

5.10.1.6.2.7 EncryptionKey Property

Used to specify the encryption key for working with an encrypted database.

Class

`TLiteConnection`

Syntax

```
property EncryptionKey: string stored False;
```

Remarks

Use the EncryptionKey property to specify the encryption key for working with an encrypted database .

When working in Direct mode , the property is used to connect to an encrypted database or to encrypt a database using the EncryptDatabase method.

Otherwize, the appropriate encryption support has to be implemented in the SQLite client library that is used by TLiteConnection.

See Also

- TLiteConnection

- EncryptDatabase

- TLiteConnectionOptions.Direct

- Connecting in the Direct mode

- [Database File Encryption](#)

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

5.10.1.6.2.8 LoginPrompt Property

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

Class

`TLiteConnection`

Syntax

```
property LoginPrompt;
```

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 LiteDacVcl unit appears to the uses clause.

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

5.10.1.6.2.9 Options Property

Used to specify the behaviour of the TLiteConnection object.

Class

`TLiteConnection`

Syntax

```
property Options: TLiteConnectionOptions;
```

Remarks

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

See Also

- TCustomDAConnection.Options

Request Support   DAC Forum   Provide Feedback

5.10.1.6.2.10 Pooling Property

Enables or disables using connection pool.

Class

`TLiteConnection`

Syntax

```
property Pooling: boolean;
```

Remarks

Normally, when TLiteConnection establishes connection with the database it takes server memory and time resources for allocating new database 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. TLiteConnection 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, TLiteConnection.Database, TLiteConnection.ClientLibrary, TLiteConnection.EncryptionKey, TLiteConnectionOptions.ASCIIDatabase, TLiteConnectionOptions.DateFormat, TLiteConnectionOptions.TimeFormat, TLiteConnectionOptions.EnableSharedCache, TLiteConnectionOptions.BusyTimeout, TLiteConnectionOptions.ReadUncommitted, TLiteConnectionOptions.DefaultCollations, TLiteConnectionOptions.ForeignKeys, TLiteConnectionOptions.UseUnicode. Note: Using Pooling := True can cause errors with working with temporary tables.

See Also

- PoolingOptions

Request Support   DAC Forum   Provide Feedback

5.10.1.6.2.11 PoolingOptions Property

Specifies the behaviour of connection pool.

Class

TLiteConnection

Syntax

```
property PoolingOptions: TPoolingOptions;
```

Remarks

Set the properties of PoolingOptions to specify the behaviour of connection pool.

See Also

- Pooling

Request Support   DAC Forum   Provide Feedback

5.10.1.6.3 Methods

Methods of the **TLiteConnection** class.

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

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
Commit (inherited from **TCustomDAConnection**)	Commits current transaction.

Connect (inherited from **TCustomDAConnection**)	Establishes a connection to the server.
CreateDataSet	Returns a new instance of TLiteQuery class and associates it with the connection object.
CreateMetaData	Returns a new instance of TLiteMetaData class and associates it with the connection object.
CreateSQL	Returns a new instance of TLiteSQL class and associates it with the connection object.
CreateTransaction	Returns a new instance of TLiteTransaction class and associates it with the connection object.
Disconnect (inherited from **TCustomDAConnection**)	Performs disconnect.
EncryptDatabase	Encrypt the connected database with the encryption key specified.
GetKeyFieldNames (inherited from **TCustomDAConnection**)	Provides a list of available key field names.
GetTableNames (inherited from **TCustomDAConnection**)	Provides a list of available tables names.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
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](#)	Overloaded. Begins a new user transaction.

See Also

- [TLiteConnection Class](#)

- [TLiteConnection Class Members](#)

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

5.10.1.6.3.1 CreateDataSet Method

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

Class

[TLiteConnection](#)

Syntax

```
function CreateDataSet(AOwner: TComponent = nil):
TCustomDADataSet; override;
```

Return Value

A new instance of TLiteQuery class.

Remarks

The CreateDataSet method returns a new instance of TLiteQuery class and associates it with the connection object.

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

5.10.1.6.3.2 CreateMetaData Method

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

Class

TLiteConnection

Syntax

```
function CreateMetaData: TDAMetaData; override;
```

Return Value
  A new instance of TLiteMetaData class.

Remarks

The CreateMetaData method returns a new instance of TLiteMetaData class and associates it with the connection object.

Request Support   DAC Forum   Provide Feedback

5.10.1.6.3.3 CreateSQL Method

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

Class

TLiteConnection

Syntax

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

Return Value
  A new instance of TLiteSQL class.

Remarks

The CreateSQL method returns a new instance of TLiteSQL class and associates it with the connection object.

Request Support   DAC Forum   Provide Feedback

5.10.1.6.3.4 CreateTransaction Method

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

Class

TLiteConnection

Syntax

```
function CreateTransaction: TDATransaction; override;
```

Return Value

A new instance of TLiteTransaction class.

Remarks

The CreateTransaction method returns a new instance of TLiteTransaction class and associates it with the connection object.

Request Support   DAC Forum   Provide Feedback

5.10.1.6.3.5 EncryptDatabase Method

Encrypt the connected database with the encryption key specified.

Class

TLiteConnection

Syntax

```
procedure EncryptDatabase(NewEncryptionKey: string);
```

Parameters

NewEncryptionKey

Holds the encryption key to encrypt the database.

Remarks

When working in the Direct mode , the TLiteConnectionOptions.EncryptionAlgorithm property should be used to specify the encryption algorithm to encrypt the database .

Otherwize, the appropriate encryption support has to be implemented in the SQLite client library that is used by TLiteConnection.

Note: The EncryptDatabase method can be used only for the already connected database.

See Also
- TLiteConnection
- TLiteConnectionOptions.EncryptionAlgorithm
- Connecting in the Direct mode
- Database File Encryption

Request Support   DAC Forum   Provide Feedback

5.10.1.6.3.6 ReleaseSavepoint Method

Releases the specified savepoint without affecting any work that has been performed after its creation.

Class
TLiteConnection

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
- Savepoint

Request Support   DAC Forum   Provide Feedback

Reserved.

5.10.1.6.3.7 RollbackToSavepoint Method

Cancels all updates for the current transaction.

Class

`TLiteConnection`

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

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.10.1.6.3.8 Savepoint Method

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

Class

`TLiteConnection`

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**

5.10.1.6.3.9 StartTransaction Method

Begins a new user transaction.

Class

[TLiteConnection](#)

Overload List

Name	**Description**
[StartTransaction](#)	Begins a new user transaction.
[StartTransaction(IsolationLevel: TLiteIsolationLevel)](#)	Begins a new user transaction with the defined isolation level.

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

Begins a new user transaction.

Class

[TLiteConnection](#)

Syntax

```
procedure StartTransaction; overload; override;
```

Remarks

Call the StartTransaction method to begin a new user transaction against the database. 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 database until an application calls Commit to save the changes, or Rollback to cancel them.

See Also

- TCustomDAConnection.Commit

- TCustomDAConnection.Rollback

- TCustomDAConnection.InTransaction

Request Support   DAC Forum   Provide Feedback

Begins a new user transaction with the defined isolation level.

Class

TLiteConnection

Syntax

```
procedure StartTransaction(IsolationLevel: TLiteIsolationLevel);
reintroduce; overload;
```

Parameters

IsolationLevel

Remarks

Call the StartTransaction method to begin a new user transaction against the database and to

simultaneously define the isolation level for it. 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 database until an application calls Commit to save the changes, or Rollback to cancel them.

See Also

- TCustomDAConnection.Commit

- TCustomDAConnection.Rollback

- TCustomDAConnection.InTransaction

Request Support   DAC Forum   Provide Feedback

5.10.1.6.4 Events

Events of the **TLiteConnection** class.

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

Published

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

See Also

- TLiteConnection Class

- TLiteConnection Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.6.4.1 OnConnectionLost Event

This event occurs when connection was lost.

Class

[TLiteConnection](#)

Syntax

```
property OnConnectionLost: TConnectionLostEvent;
```

Remarks

Write the OnConnectionLost event handler to process fatal errors and perform failover.

Note: you should explicitly add the MemData unit to the 'uses' list to use the

OnConnectionLost event handler.

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

5.10.1.6.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

[TLiteConnection](#)

Syntax

```
property OnError: TDAConnectionErrorEvent;
```

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

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

5.10.1.7 TLiteConnectionOptions Class

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

For a list of all members of this type, see TLiteConnectionOptions members.

Unit

LiteAccess

Syntax

TLiteConnectionOptions = **class**(TDAConnectionOptions);

Inheritance Hierarchy

TDAConnectionOptions

  TLiteConnectionOptions

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.1 Members

TLiteConnectionOptions class overview.

Properties

Name	Description
AllowImplicitConnect (inherited from **TDAConnectionOptions**)	Specifies whether to allow or not implicit connection opening.
ASCIIDataBase	Used to enable or disable ASCII support.
BusyTimeout	Used to set the timeout for a locked resource (database or table).
CipherLicense	Used to specify a license key for SQLCipher Commercial Edition.
ConnectMode	Used to specify the connection mode.
DateFormat	Used to specify the format for storing dates in the

	database.
[DefaultCollations](#)	Enables or disables automatic default collations registration on connection establishing. List of available default collations: UniNoCase - allows to compare unicode strings case-insensitively. Default value of this option is False.
[Direct](#)	Used to connect to the database directly, without the SQLite3 client library.
[DisconnectedMode](#)	Enables or disables direct access to a database, without using of the SQLite client library. For more information about using direct access in LiteDAC, read the Working In Direct Mode article. Default value of this option is False.
[EnableBCD](#)	Used to enable or disable the currency type.
[EnableFMTBCD](#)	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
[EnableLoadExtension](#)	Enables loading and using an SQLite extension:
[EnableSharedCache](#)	Used to enable or disable the shared-cache mode.
[EncryptionAlgorithm](#)	Used to specify the encryption algorithm for an encrypted database.
[ForceCreateDatabase](#)	Used to force TLiteConnection to create a new database before opening a connection, if the database doesn't already exist.
[ForeignKeys](#)	Used to enable or disable foreign key constraints.
[IntegerAsLargeInt](#)	Used to map INT (INTEGER) columns to TLargeIntField

	fields.
JournalMode	Used to specify the journal mode.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
LockingMode	Used to specify the database locking mode: Exclusive or Normal.
NativeDate	Defines the format for date and time stored in the database.
ReadUncommitted	Used to enable or disable the Read Uncommitted isolation mode.
Synchronous	Used to specify the database synchronization mode when writing to disk.
TimeFormat	Used to specify the format for storing time in the database.
UseUnicode	Used to enable or disable Unicode support.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2 Properties

Properties of the **TLiteConnectionOptions** class.

For a complete list of the **TLiteConnectionOptions** class members, see the

TLiteConnectionOptions Members topic.

Public

Name	Description
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection.On ConnectionLost event occurs and a failover operation can be performed after connection breaks.

Published

Name	Description
AllowImplicitConnect (inherited from **TDAConnectionOptions**)	Specifies whether to allow or not implicit connection opening.
ASCIIDataBase	Used to enable or disable ASCII support.
BusyTimeout	Used to set the timeout for a locked resource (database or table).
CipherLicense	Used to specify a license key for SQLCipher Commercial Edition.
ConnectMode	Used to specify the connection mode.
DateFormat	Used to specify the format for storing dates in the database.
DefaultCollations	Enables or disables automatic default collations registration on connection establishing. List of available default collations: UniNoCase - allows to compare unicode strings case-insensitively. Default value of this option is False.
Direct	Used to connect to the database directly, without the SQLite3 client library.

[DisconnectedMode](#)	Enables or disables direct access to a database, without using of the SQLite client library. For more information about using direct access in LiteDAC, read the Working In Direct Mode article. Default value of this option is False.
[EnableBCD](#)	Used to enable or disable the currency type.
[EnableFMTBCD](#)	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
[EnableLoadExtension](#)	Enables loading and using an SQLite extension:
[EnableSharedCache](#)	Used to enable or disable the shared-cache mode.
[EncryptionAlgorithm](#)	Used to specify the encryption algorithm for an encrypted database.
[ForceCreateDatabase](#)	Used to force TLiteConnection to create a new database before opening a connection, if the database doesn't already exist.
[ForeignKeys](#)	Used to enable or disable foreign key constraints.
[IntegerAsLargeInt](#)	Used to map INT (INTEGER) columns to TLargeIntField fields.
[JournalMode](#)	Used to specify the journal mode.
[LockingMode](#)	Used to specify the database locking mode: Exclusive or Normal.
[NativeDate](#)	Defines the format for date and time stored in the database.
[ReadUncommitted](#)	Used to enable or disable the Read Uncommitted isolation mode.

[Synchronous](#)	Used to specify the database synchronization mode when writing to disk.
[TimeFormat](#)	Used to specify the format for storing time in the database.
[UseUnicode](#)	Used to enable or disable Unicode support.

See Also

- [TLiteConnectionOptions Class](#)

- [TLiteConnectionOptions Class Members](#)

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

5.10.1.7.2.1 ASCIIDataBase Property

Used to enable or disable ASCII support.

Class

[TLiteConnectionOptions](#)

Syntax

```
property ASCIIDataBase: boolean default False;
```

Remarks

Use the ASCIIDataBase property to enable or disable ASCII support. The default value is False.

Note: Set the [UseUnicode](#) property to False before enabling ASCII support.

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

5.10.1.7.2.2 BusyTimeout Property

Used to set the timeout for a locked resource (database or table).

Class

TLiteConnectionOptions

Syntax

```
property BusyTimeout: integer default 0;
```

Remarks

Use the BusyTimeout property to set the timeout in milliseconds for a locked resource (database or table). If the resource is not unlocked during this time, SQLite returns the SQLITE_BUSY error. The default value is 0.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.3 CipherLicense Property

Used to specify a license key for SQLCipher Commercial Edition.

Class

TLiteConnectionOptions

Syntax

```
property CipherLicense: string;
```

Remarks

Use the CipherLicense property to specify your license key for SQLCipher Commercial Edition.

Note: SQLCipher is not supported in the Direct mode.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.4 ConnectMode Property

Used to specify the connection mode.

Class

[TLiteConnectionOptions](#)

Syntax

```
property ConnectMode: TConnectMode default DefValConnectMode;
```

Remarks

Use the ConnectMode property to specify the connection mode.

cmDefault - The default value. The database is opened for reading and writing. Corresponds to the SQLite default behavior.

cmReadOnly - The database is opened in read-only mode.

cmReadWrite - The database is opened for reading and writing.

Note that the [ForceCreateDatabase](#) property can be used to enable the automatic creation of the database when it doesn't already exist.

© 1997-2024
Devart. All Rights
Reserved. [Request Support](#) [DAC Forum](#) [Provide Feedback](#)

5.10.1.7.2.5 DateFormat Property

Used to specify the format for storing dates in the database.

Class

[TLiteConnectionOptions](#)

Syntax

```
property DateFormat: string;
```

Remarks

Use the DateFormat property to specify the format for storing dates in the database. If no format is specified, the default yyyy-mm-dd format will be used. The default value is an empty

string.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.6 DefaultCollations Property

Enables or disables automatic default collations registration on connection establishing. List of available default collations: UniNoCase - allows to compare unicode strings case-insensitively. Default value of this option is False.

Class

`TLiteConnectionOptions`

Syntax

```
property DefaultCollations: boolean default True;
```

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.7 Direct Property

Used to connect to the database directly, without the SQLite3 client library.

Class

`TLiteConnectionOptions`

Syntax

```
property Direct: boolean default DefValDirect;
```

Remarks

Use the Direct property to connect to the database directly using the embedded SQLite3 engine, without the SQLite3 client library.

The Direct mode also enables you to work with an encrypted database using the EncryptionAlgorithm and TLiteConnection.EncryptionKey properties and the TLiteConnection.EncryptDatabase method.

See Also

- [EncryptionAlgorithm](#)

- [TLiteConnection.EncryptionKey](#)

- [TLiteConnection.EncryptDatabase](#)

- [Connecting in the Direct mode](#)

- [Database File Encryption](#)

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.8 DisconnectedMode Property

Enables or disables direct access to a database, without using of the SQLite client library. For more information about using direct access in LiteDAC, read the Working In Direct Mode article. Default value of this option is False.

Class

`TLiteConnectionOptions`

Syntax

```
property DisconnectedMode: boolean;
```

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.9 EnableBCD Property

Used to enable or disable the currency type.

Class

`TLiteConnectionOptions`

Syntax

```
property EnableBCD: boolean;
```

Remarks

Use the EnableBCD property to enable or disable the currency type. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.10 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

`TLiteConnectionOptions`

Syntax

```
property EnableFMTBCD: boolean;
```

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.11 EnableLoadExtension Property

Enables loading and using an SQLite extension:

Class

`TLiteConnectionOptions`

Syntax

```
property EnableLoadExtension: boolean default False;
```

Example

```
LiteConnection.ExecSQL('SELECT load_extension(''C:\ext.dll'', ''sqlite3_ext_
```

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.12 EnableSharedCache Property

Used to enable or disable the shared-cache mode.

Class

TLiteConnectionOptions

Syntax

```
property EnableSharedCache: boolean default False;
```

Remarks

Use the EnableSharedCache property to enable or disable the SQLite shared-cache mode.

The default value is False.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.13 EncryptionAlgorithm Property

Used to specify the encryption algorithm for an encrypted database.

Class

TLiteConnectionOptions

Syntax

```
property EncryptionAlgorithm: TLiteEncryptionAlgorithm default
DefaultEncryptionAlgorithm;
```

Remarks

Use the EncryptionAlgorithm property to specify the encryption algorithm when connecting to an encrypted database , or the encryption algorithm that will be used to encrypt a database using the TLiteConnection.EncryptDatabase method. Also, the TLiteConnection.EncryptionKey property is used to specify the encryption key to open/encrypt a database.

Interacting with encrypted database only available in the Direct mode .

See Also

- Direct

- TLiteConnection.EncryptionKey

- TLiteConnection.EncryptDatabase

- [Connecting in the Direct mode](#)

- [Database File Encryption](#)

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

5.10.1.7.2.14 ForceCreateDatabase Property

Used to force TLiteConnection to create a new database before opening a connection, if the database doesn't already exist.

Class

[TLiteConnectionOptions](#)

Syntax

```
property ForceCreateDatabase: boolean default
DefValForceCreateDatabase;
```

Remarks

By default, when connecting to a database, SQLite3 does not check whether there exists the specified file. If the [TLiteConnection.Database](#) property points to a non-existent database in correct system path, a new empty database will be created and opened, and no warning message will be displayed. In the case if an incorrect database name was entered by mistake, this behavior can lead to misunderstandings and errors in the operation of the software.

If the TLiteConnectionOptions.ForceDatabaseCreate property is set to False, before establishing a connection to the database, [TLiteConnection](#) will check whether the specified file exists. If the file does not exist, an appropriate exception will be raised.

If the TLiteConnectionOptions.ForceDatabaseCreate property is set to True, no checking will be performed and a new database will be created.

The default value of the TLiteConnectionOptions.ForceDatabaseCreate property is False.

See Also
- [TLiteConnection.Database](#)

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

5.10.1.7.2.15 ForeignKeys Property

Used to enable or disable foreign key constraints.

Class

TLiteConnectionOptions

Syntax

```
property ForeignKeys: boolean default True;
```

Remarks

By default, foreign key constraints are disabled in SQLite. Use the ForeignKeys property to enable or disable foreign keys constraints without explicitly executing the "PRAGMA foreign_keys = ON;" and "PRAGMA foreign_keys = OFF;" statements. The default value if True.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.16 IntegerAsLargeInt Property

Used to map INT (INTEGER) columns to TLargeIntField fields.

Class

TLiteConnectionOptions

Syntax

```
property IntegerAsLargeInt: boolean default False;
```

Remarks

Use the IntegerAsLargeInt property to map INT (INTEGER) columns to TLargeIntField fields. The default value is False.

If True, INT (INTEGER) columns are mapped to TLargeIntField fields.

If False, INT (INTEGER) columns are mapped to TIntegerField fields.

5.10.1.7.2.17 JournalMode Property

Used to specify the journal mode.

Class

TLiteConnectionOptions

Syntax

```
property JournalMode: TJournalMode default DefValJournalMode;
```

Remarks

Use the JournalMode property to specify the journal mode.

- **jmDefault** - The default value. If the database was previously opened in the WAL mode, then Default will open the database in the WAL mode; otherwise, the database will be opened in the Delete mode.

- **jmDelete** - The rollback journal is deleted at the conclusion of each transaction.

- **jmMemory** - The rollback journal is stored in volatile RAM. This reduces the disk I/O, but decreases database safety and integrity. If the application using SQLite crashes in the middle of a transaction in this mode, the database file is likely to become corrupt.

- **jmOff** - The rollback journal is completely disabled. No rollback journal is created, thus there is no rollback journal to delete. The ROLLBACK command does not work -- it behaves in an undefined way. Avoid using the ROLLBACK command when the journal mode is disabled.

- **jmPersist** - The rollback journal file is not deleted when the transaction is commited. The header of the journal is filled with zeroes to prevent other connections rolling back from the journal. This mode optimizes performance on platforms where deleting or truncating a file is much more expensive than overwriting the first block of a file with zeros.

- **jmTruncate** - The rollback journal is stored in volatile RAM. It reduces disk I/O, but decreases database safety and integrity. If the application using SQLite crashes in the middle of a transaction, the database file may become corrupt.

- **jmWAL** - A write-ahead log is used instead of a rollback journal to implement transactions. When data database is updated, the original content is preserved in the database file and the changes are appended in a separate WAL file. All the transactions that are appended in the WAL file are eventually transferred back into the original database.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.18 LockingMode Property

Used to specify the database locking mode: Exclusive or Normal.

Class

`TLiteConnectionOptions`

Syntax

```
property LockingMode: TLockingMode default DefValLockingMode;
```

Remarks

Use the LockingMode property to specify the database locking mode: Exclusive or Normal.

- **lmExclusive** - The database connection never releases file locks. The first time the database is read or written in this mode, a shared lock is obtained and held. Use this mode if you want to prevent other processes from accessing the database file, reduce the number of filesystem operations, or access WAL databases without using the shared memory.

- **lmNormal** - The database connection unlocks the database file at the conclusion of each read or write transaction.

Note: Keep the default LockingMode=lmExclusive and Synchronous=smOff for the best perfomance.

See Also

- Synchronous

Request Support   DAC Forum   Provide Feedback

Reserved.

5.10.1.7.2.19 NativeDate Property

Defines the format for date and time stored in the database.

Class

TLiteConnectionOptions

Syntax

```
property NativeDate: boolean default True;
```

Remarks

If the property is set to True, the date and time values will be stored in the database in the native SQLite format, and when retrieved, they will be converted to the TDateTime type. If the property is set to False, then there is no conversion to the TDateTime type. The default value is True.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.20 ReadUncommitted Property

Used to enable or disable the Read Uncommitted isolation mode.

Class

TLiteConnectionOptions

Syntax

```
property ReadUncommitted: boolean default False;
```

Remarks

Use the ReadUncommitted property to enable or disable the Read Uncommitted isolation mode. A database connection in this mode doesn't attempt to acquire a read lock on the table before reading from it. This can lead to inconsistent query results if another database connection modifies data in the table while it is being read, but it also means that a read transaction opened by a connection in the Read Uncommitted mode can neither block nor be

blocked by another connection. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.21 Synchronous Property

Used to specify the database synchronization mode when writing to disk.

Class

TLiteConnectionOptions

Syntax

```
property Synchronous: TSynchronous default DefValSynchronous;
```

Remarks

Use the Synchronous property to specify the database synchronization mode when writing to disk.

- **smExtra** - This mode is similar to the FULL mode, but in the DELETE mode, the directory containing the rollback journal is synced after that journal is unlinked to commit a transaction. This provides additional durability if a power outage occurs right after the commit.

- **smFull** - The database engine ensures that all content is safely written to disk before continuing. This preserves database integrity even in case of an operating system failure or power outage. It is a safe, but slower mode, and is most commonly used when not in the WAL mode.

- **smNormal** - The database engine still syncs at the most critical moments, but less often than in the FULL mode. The Normal mode is faster than the Full mode. When using the WAL mode (and probably the DELETE mode) with synchronous=NORMAL, data is safe from corruption. The synchronous=NORMAL setting is a reasonable choice for most applications running in the WAL mode.

- **smOff** - The database engine continues without syncing after handing data off to the operating system. If the application running SQLite crashes, the data will safe, unless the

operating system crashes or the computer loses power before data has been written to disk, in which case the database might become corrupted. This is the fastest mode.

Note: Keep the default Synchronous=smOff and LockingMode=lmExclusive for the best perfomance.

See Also
- LockingMode

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.22 TimeFormat Property

Used to specify the format for storing time in the database.

Class

TLiteConnectionOptions

Syntax

```
property TimeFormat: string;
```

Remarks

Used to specify the format for storing time in a database. If no format is specified, the default hh:mm:ss format will be used. The default value is an empty string.

Request Support   DAC Forum   Provide Feedback

5.10.1.7.2.23 UseUnicode Property

Used to enable or disable Unicode support.

Class

TLiteConnectionOptions

Syntax

```
property UseUnicode: boolean default DefValUseUnicode;
```

Remarks

Use the UseUnicode property to enable or disable Unicode support. When set to True, all character data is stored as WideString, and TStringField is used instead of TWideStringField. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.10.1.8 TLiteDataSetOptions Class

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

For a list of all members of this type, see TLiteDataSetOptions members.

Unit

LiteAccess

Syntax

```
TLiteDataSetOptions = class(TDADataSetOptions);
```

Inheritance Hierarchy

TDADataSetOptions

   TLiteDataSetOptions

Request Support   DAC Forum   Provide Feedback

5.10.1.8.1 Members

TLiteDataSetOptions class overview.

Properties

Name	Description
AdvancedTypeDetection	Allows advanced detection of a field data type in case if metadata from the DB cannot be used.

AutoPrepare (inherited from **TDADataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
DefaultValues (inherited from **TDADataSetOptions**)	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo	Used to perform an additional query to get information about returned fields and the tables they belong to.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
FullRefresh	Used to specify the fields to include in the automatically generated SQL statement

	when calling the method.
[InsertAllSetFields](#) (inherited from **[TDADataSetOptions](#)**)	Used to include all set dataset fields in the generated INSERT statement
[LocalMasterDetail](#) (inherited from **[TDADataSetOptions](#)**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
[LongStrings](#) (inherited from **[TDADataSetOptions](#)**)	Used to represent string fields with the length that is greater than 255 as TStringField.
[MasterFieldsNullable](#) (inherited from **[TDADataSetOptions](#)**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
[NumberRange](#) (inherited from **[TDADataSetOptions](#)**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
[PrepareUpdateSQL](#)	Used to automatically prepare update queries before execution.
[QueryRecCount](#) (inherited from **[TDADataSetOptions](#)**)	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
[QuoteNames](#) (inherited from **[TDADataSetOptions](#)**)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.

RemoveOnRefresh (inherited from **TDADataSetOptions**)	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from **TDADataSetOptions**)	Used to return the new value of fields to dataset after insert or update.
SetEmptyStrToNull	Force replace of empty strings with NULL values in data. Default value is False.
SetFieldsReadOnly (inherited from **TDADataSetOptions**)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar (inherited from **TDADataSetOptions**)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
TrimVarChar	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
UnknownAsString	Used to map fields of unknown data types to TStringField (TWideStringField).
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.

5.10.1.8.2 Properties

Properties of the **TLiteDataSetOptions** class.

For a complete list of the **TLiteDataSetOptions** class members, see the

TLiteDataSetOptions Members topic.

Public

Name	Description
AutoPrepare (inherited from **TDADataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
DefaultValues (inherited from **TDADataSetOptions**)	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes

	fields.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from **TDADataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh (inherited from **TDADataSetOptions**)	Used for a dataset to locally remove a record that can not be found on the server.

RequiredFields (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from **TDADataSetOptions**)	Used to return the new value of fields to dataset after insert or update.
SetFieldsReadOnly (inherited from **TDADataSetOptions**)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar (inherited from **TDADataSetOptions**)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields (inherited from **TDADataSetOptions**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from **TDADataSetOptions**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Published

Name	Description
AdvancedTypeDetection	Allows advanced detection of a field data type in case if metadata from the DB cannot be used.
EnableBCD	Used to enable currency type. Default value of this option is False.

[EnableFMTBCD](#)	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
[ExtendedFieldsInfo](#)	Used to perform an additional query to get information about returned fields and the tables they belong to.
[FullRefresh](#)	Used to specify the fields to include in the automatically generated SQL statement when calling the method.
[PrepareUpdateSQL](#)	Used to automatically prepare update queries before execution.
[SetEmptyStrToNull](#)	Force replace of empty strings with NULL values in data. Default value is False.
[TrimVarChar](#)	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.
[UnknownAsString](#)	Used to map fields of unknown data types to TStringField (TWideStringField).

See Also

- [TLiteDataSetOptions Class](#)

- [TLiteDataSetOptions Class Members](#)

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

5.10.1.8.2.1 AdvancedTypeDetection Property

Allows advanced detection of a field data type in case if metadata from the DB cannot be used.

Class

TLiteDataSetOptions

Syntax

```
property AdvancedTypeDetection: boolean default False;
```

Remarks

If the property is set to False, then when detecting the field type, standard metadata retrieval from the database is performed. If the property is True, then a number of table records is prefetched, and the field type is detected based on the type of data stored in the corresponding column of the table.The default value is False.

Request Support   DAC Forum   Provide Feedback

5.10.1.8.2.2 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

TLiteDataSetOptions

Syntax

```
property EnableBCD: boolean;
```

Request Support   DAC Forum   Provide Feedback

5.10.1.8.2.3 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

TLiteDataSetOptions

Syntax

```
property EnableFMTBCD: boolean;
```

Request Support   DAC Forum   Provide Feedback

Reserved.

5.10.1.8.2.4 ExtendedFieldsInfo Property

Used to perform an additional query to get information about returned fields and the tables they belong to.

Class

`TLiteDataSetOptions`

Syntax

```
property ExtendedFieldsInfo: boolean default True;
```

Remarks

If True, an additional query is performed to get information about returned fields and the tables they belong to.

Request Support   DAC Forum   Provide Feedback

5.10.1.8.2.5 FullRefresh Property

Used to specify the fields to include in the automatically generated SQL statement when calling the method.

Class

`TLiteDataSetOptions`

Syntax

```
property FullRefresh: boolean;
```

Remarks

Use the FullRefresh property to specify what fields to include in the automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. If the FullRefresh property is True, all fields from a query are included into SQL statement to refresh a single record. If FullRefresh is False, only fields from TLiteQuery.UpdatingTable are included.

Note: If FullRefresh is True, the refresh of SQL statement for complex queries and views may be generated with errors. The default value is False.

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

5.10.1.8.2.6 PrepareUpdateSQL Property

Used to automatically prepare update queries before execution.

Class

`TLiteDataSetOptions`

Syntax

```
property PrepareUpdateSQL: boolean;
```

Remarks

If True, update queries are automatically prepared before executing.

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

5.10.1.8.2.7 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. Default value is False.

Class

`TLiteDataSetOptions`

Syntax

```
property SetEmptyStrToNull: boolean;
```

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

5.10.1.8.2.8 TrimVarChar Property

Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

Class

TLiteDataSetOptions

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

5.10.1.8.2.9 Unknow nAsString Property

Used to map fields of unknown data types to TStringField (TWideStringField).

Class

TLiteDataSetOptions

Syntax

```
property UnknownAsString: boolean default False;
```

Remarks

Use the UnknownAsString to map fields of unknown data types to TStringField (TWideStringField). If False, fields of unknown data types (for example the ifnull function result) are mapped to TMemoField or TWideMemoField depending on the value of the UseUnicode option. Memo is used because maximum length of values from such fields is unknown. If True, fields of unknown data types are mapped to TStringField or TWideStringField depending on the value of the UseUnicode option. Size of fields is set to 8192. Values larger than this size are truncated. The default value is False.

Request Support   DAC Forum   Provide Feedback

5.10.1.9 TLiteDataSource Class

TLiteDataSource provides an interface between a LiteDAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TLiteDataSource members.

Unit

`LiteAccess`

Syntax

```
TLiteDataSource = class(TCRDataSource);
```

Remarks

TLiteDataSource provides an interface between a LiteDAC dataset components and data-aware controls on a form. TLiteDataSource 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

   TLiteDataSource

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

5.10.1.9.1 Members

TLiteDataSource class overview.

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

5.10.1.10 TLiteEncryptor Class

The class that performs encrypting and decrypting of data.

For a list of all members of this type, see TLiteEncryptor members.

Unit

LiteAccess

Syntax

```
TLiteEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

TCREncryptor

TLiteEncryptor

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

5.10.1.10.1 Members

TLiteEncryptor 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**

5.10.1.11 TLiteMetaData Class

A component for obtaining metainformation about database objects.

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

Unit

[LiteAccess](#)

Syntax

```
TLiteMetaData = class(TDAMetaData);
```

Remarks

The TLiteMetaData component is used to obtain metainformation about objects in the database, such as tables, table columns, indexes, etc.

Inheritance Hierarchy

[TMemDataSet](#)

   [TDAMetaData](#)

     TLiteMetaData

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

5.10.1.11.1 Members

[TLiteMetaData](#) 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 that will be used to connect to a data

	store.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MetaDataKind (inherited from **TDAMetaData**)	Used to specify which kind of metainformation to show.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Restrictions (inherited from **TDAMetaData**)	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.

CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds (inherited from **TDAMetaData**)	Used to get values acceptable in the MetaDataKind property.
GetRestrictions (inherited from **TDAMetaData**)	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.

SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

Reserved.

5.10.1.11.2 Properties

Properties of the **TLiteMetaData** class.

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

Public

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
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.

Published

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

See Also

- TLiteMetaData Class

- TLiteMetaData Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.11.2.1 Connection Property

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

Class

`TLiteMetaData`

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

See Also

- TLiteConnection

Request Support   DAC Forum   Provide Feedback

5.10.1.12 TLiteQuery 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 TLiteQuery members.

Unit

LiteAccess

Syntax

```
TLiteQuery = class(TCustomLiteDataSet);
```

Remarks

TLiteQuery is a direct descendant of the TCustomLiteDataSet component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.

Use TLiteQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TLiteQuery 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, TLiteQuery will retrieve primary keys for UpdatingTable from metadata. TLiteQuery 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 TLiteQuery.BeforeUpdateExecute event to assign the value to additional parameters and the TLiteQuery.AfterUpdateExecute event to read them.

Inheritance Hierarchy

TMemDataSet

  TCustomDADataSet

    TCustomLiteDataSet

      TLiteQuery

See Also

- TLiteTable

Request Support   DAC Forum   Provide Feedback

5.10.1.12.1   Members

TLiteQuery 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.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomLiteDataSet**)	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 **TCustomLiteDataSet**)	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.
[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 **[TCustomLiteDataSet](#)**)	Specifies the behaviour of TCustomLiteDataSet 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 **TCustomLiteDataSet**)	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.
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.
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.
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 **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
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.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and

	SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

5.10.1.12.2 Properties

Properties of the **TLiteQuery** class.

For a complete list of the **TLiteQuery** class members, see the TLiteQuery 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.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomLiteDataSet**)	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 **TCustomLiteDataSet**)	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.
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 **TCustomLiteDataSet**)		Specifies the behaviour of TCustomLiteDataSet 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 **TCustomLiteDataSet**)	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.
[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.
[UpdateObject](#)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
[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

- [TLiteQuery Class](#)

- [TLiteQuery Class Members](#)

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

5.10.1.12.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

`TLiteQuery`

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](#)

5.10.1.12.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

`TLiteQuery`

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

- TLiteTable.LockMode

Request Support   DAC Forum   Provide Feedback

5.10.1.12.2.3 UpdateObject Property

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

Class

TLiteQuery

Syntax

```
property UpdateObject: TCustomDAUpdateSQL;
```

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

5.10.1.12.2.4 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

TLiteQuery

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 TCustomLiteDataSet.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.

Example

Below are two examples for the query, where:

1. the only allowed value for UpdatingTable property is 'Dept';

2. allowed values for UpdatingTable are 'Dept' and 'Emp'.

In the first case (or by default) editable field is ShipName, in the second - all fields from Emp.

```
1)Example 1.
   SELECT * FROM Dept
2) Example 2.
    SELECT * FROM Dept, Emp
     WHERE Dept.DeptNo = Emp.DeptNo
```

Request Support   DAC Forum   Provide Feedback

5.10.1.13 TLiteSQL Class

A component for executing SQL statements.

For a list of all members of this type, see TLiteSQL members.

Unit

LiteAccess

Syntax

```
TLiteSQL = class(TCustomDASQL);
```

Remarks

The TLiteSQL component is a direct descendant of the TCustomDASQL class. Use The TLiteSQL component when a client application must execute SQL statement. The SQL statement should not retrieve rows from the database.

Inheritance Hierarchy

TCustomDASQL

  TLiteSQL

© 1997-2024
Devart. All Rights
Reserved.

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

5.10.1.13.1 Members

TLiteSQL class overview.

Properties

Name	Description
ChangeCursor (inherited from **TCustomDASQL**)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
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.
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

5.10.1.13.2 Properties

Properties of the **TLiteSQL** class.

For a complete list of the **TLiteSQL** class members, see the TLiteSQL 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.
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
Connection	Used to specify a connection object that will be used to connect to a data store.

See Also

- TLiteSQL Class

- TLiteSQL Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.13.2.1 Connection Property

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

Class

TLiteSQL

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

See Also

- TLiteConnection

Request Support   DAC Forum   Provide Feedback

5.10.1.14 TLiteTable Class

Accesses data in a single table without writing SQL statements.

For a list of all members of this type, see TLiteTable members.

Unit

LiteAccess

Syntax

```
TLiteTable = class(TCustomLiteTable);
```

Inheritance Hierarchy

TMemDataSet

  TCustomDADataSet

    TCustomLiteDataSet

      TCustomLiteTable

        TLiteTable

    Request Support   DAC Forum   Provide Feedback

5.10.1.14.1 Members

TLiteTable 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.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomLiteDataSet**)	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 **TCustomLiteDataSet**)	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.
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 **TCustomLiteTable**)	Used to set the number of rows retrieved from the table.
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 **TCustomLiteTable**)	Used to allow retrieving data from the database starting from the specified row.
Options (inherited from **TCustomLiteDataSet**)	Specifies the behaviour of TCustomLiteDataSet 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 **TCustomLiteDataSet**)		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	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.
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.
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 **[TCustomDADataSet](#)**)	Locks the current record.
[MacroByName](#) (inherited from **[TCustomDADataSet](#)**)	Finds a macro with the specified name.
[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.
[RefreshRecord](#) (inherited from **[TCustomDADataSet](#)**)	Actualizes field values for the current record.
[RestoreSQL](#) (inherited from **[TCustomDADataSet](#)**)	Restores the SQL property modified by AddWhere and SetOrderBy.
[RestoreUpdates](#) (inherited from **[TMemDataSet](#)**)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#) (inherited from **[TMemDataSet](#)**)	Cancels changes made to the current record when cached updates are enabled.
[SaveSQL](#) (inherited from **[TCustomDADataSet](#)**)	Saves the SQL property value to BaseSQL.
[SaveToXML](#) (inherited from **[TMemDataSet](#)**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetOrderBy](#) (inherited from **[TCustomDADataSet](#)**)	Builds an ORDER BY clause of a SELECT statement.
[SetRange](#) (inherited from **[TMemDataSet](#)**)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the start of the range

	of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not

| | handle the updates. |

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

5.10.1.14.2 Properties

Properties of the **TLiteTable** class.

For a complete list of the **TLiteTable** class members, see the TLiteTable 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.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomLiteDataSet**)	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 **TCustomLiteDataSet**)	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.
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 **TCustomLiteTable**)	Used to set the number of rows retrieved from the table.
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 **TCustomLiteTable**)	Used to allow retrieving data from the database starting from the specified row.
Options (inherited from **TCustomLiteDataSet**)	Specifies the behaviour of TCustomLiteDataSet 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 **TCustomLiteDataSet**)	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.
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.
UpdateObject	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.

See Also

- TLiteTable Class

- TLiteTable Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.14.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TLiteTable

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

5.10.1.14.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

`TLiteTable`

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

- TLiteQuery.LockMode

Request Support   DAC Forum   Provide Feedback

5.10.1.14.2.3 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

[TLiteTable](#)

Syntax

```
property OrderFields: string;
```

Remarks

TLiteTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas.

TLiteTable is reopened when OrderFields is being changed.

See Also

- [TLiteTable](#)

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

5.10.1.14.2.4 TableName Property

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

Class

[TLiteTable](#)

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](#)

5.10.1.14.2.5 UpdateObject Property

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

Class

TLiteTable

Syntax

```
property UpdateObject: TCustomDAUpdateSQL;
```

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

5.10.1.15 TLiteUpdateSQL Class

Lets you tune update operations for the DataSet component.

For a list of all members of this type, see TLiteUpdateSQL members.

Unit

LiteAccess

Syntax

```
TLiteUpdateSQL = class(TCustomDAUpdateSQL);
```

Inheritance Hierarchy

TCustomDAUpdateSQL

  TLiteUpdateSQL

Request Support   DAC Forum   Provide Feedback

5.10.1.15.1 Members

TLiteUpdateSQL 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**

5.10.1.16 TLiteUserCollation Class

A base class that provides SQLite functionality for working with user-defined collations.

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

Unit

LiteAccess

Syntax

```
TLiteUserCollation = class(TCustomLiteUserCollation);
```

Remarks

TCustomLiteUserCollation is a base component that provides functionality for classes derived from it. Applications never use TCustomLiteUserCollation objects directly. Instead they use [TLiteUserCollation](#) that inherit its properties and methods.

Inheritance Hierarchy

[TCustomLiteUserCollation](#)

  TLiteUserCollation

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

5.10.1.16.1 Members

TLiteUserCollation class overview.

Properties

Name	Description
CollationName	Used to specify the name of the collation.
Connection	Used to specify a connection object that will be used to connect to a data store.

Events

Name	Description
OnCollate	Occurs when the collation is called in the SQL-statement.

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

5.10.1.16.2 Properties

Properties of the **TLiteUserCollation** class.

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

Published

Name	Description
CollationName	Used to specify the name of the collation.
Connection	Used to specify a connection object that will be used to connect to a data store.

See Also

- **TLiteUserCollation Class**

- **TLiteUserCollation Class Members**

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

5.10.1.16.2.1 CollationName Property

Used to specify the name of the collation.

Class

TLiteUserCollation

Syntax

```
property CollationName: string;
```

Remarks

Use the CollationName property to specify the name of the collation. The name is case-insensitive.

Request Support   DAC Forum   Provide Feedback

5.10.1.16.2.2 Connection Property

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

Class

TLiteUserCollation

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

Request Support   DAC Forum   Provide Feedback

5.10.1.16.3 Events

Events of the **TLiteUserCollation** class.

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

Published

Name	Description
OnCollate	Occurs when the collation is called in the SQL-statement.

See Also

- TLiteUserCollation Class

- TLiteUserCollation Class Members

Request Support   DAC Forum   Provide Feedback

5.10.1.16.3.1 OnCollate Event

Occurs when the collation is called in the SQL-statement.

Class

TLiteUserCollation

Syntax

```
property OnCollate: TLiteCollationEvent;
```

Remarks

Use the OnCollate event handler to define the implementation of the collating function. The collating function must return an integer that is negative, zero, or positive if the first string is less than, equal to, or greater than the second, respectively.

Request Support   DAC Forum   Provide Feedback

5.10.1.17 TLiteUserFunction Class

A component for defining a custom function for future use in SQL-statements. Also, can be used for overriding built-in SQLite functions.

For a list of all members of this type, see TLiteUserFunction members.

Unit

LiteAccess

Syntax

```
TLiteUserFunction = class(TCustomLiteUserFunction);
```

Inheritance Hierarchy

TCustomLiteUserFunction

TLiteUserFunction

Request Support   DAC Forum   Provide Feedback

5.10.1.17.1 Members

TLiteUserFunction class overview.

Properties

Name	Description
Connection (inherited from **TCustomLiteUserFunction**)	Used to specify a connection object that will be used to connect to a data store.
FunctionKind (inherited from **TCustomLiteUserFunction**)	Used to specify the kind of the function that will be defined.
FunctionName (inherited from **TCustomLiteUserFunction**)	Used to specify the name of the function that will be defined.
Params (inherited from **TCustomLiteUserFunction**)	Used to specify the list of function parameters.

Events

Name	Description
OnExecute (inherited from **TCustomLiteUserFunction**)	Occurs when the user function is called in the SQL-statement.
OnFinal (inherited from **TCustomLiteUserFunction**)	Used to return a result of the user-defined aggregate function in the SQL statement.
OnStep (inherited from **TCustomLiteUserFunction**)	Occurs during execution of the user-defined aggregate function in the SQL statement.

Request Support   DAC Forum   Provide Feedback

5.10.2 Types

Types in the **LiteAccess** unit.

Types

Name	Description
TLiteCollationEvent	This type is used for the TCustomLiteUserCollation.OnCollate event.
TLiteFunctionExecuteEvent	This type is used for the TCustomLiteUserFunction.OnExecute event.
TLiteFunctionFinalEvent	This type is used for the TCustomLiteUserFunction.OnFinal event.
TLiteFunctionStepEvent	This type is used for the TCustomLiteUserFunction.OnStep event.

Request Support   DAC Forum   Provide Feedback

5.10.2.1 TLiteCollationEvent Procedure Reference

This type is used for the TCustomLiteUserCollation.OnCollate event.

Unit

LiteAccess

Syntax

```
TLiteCollationEvent = procedure (Sender: TObject; const Str1:
string; const Str2: string; var ResultValue: integer) of object;
```

Parameters

Sender
  An object that raised the event.

Str1
  The first string to be compared.

Str2
  The second string to be compared.

ResultValue
  Must return a negative integer value if the first string is less than the second. Zero if strings are equal. Positive if the first string is greater than the second.

Request Support   DAC Forum   Provide Feedback

5.10.2.2 TLiteFunctionExecuteEvent Procedure Reference

This type is used for the TCustomLiteUserFunction.OnExecute event.

Unit

LiteAccess

Syntax

```
TLiteFunctionExecuteEvent = procedure (Sender: TObject; Params:
TDAParams; var ResultValue: Variant) of object;
```

Parameters

Sender
  An object that raised the event.

Params

ResultValue
   The resulting value of the function that will be passed back.

See Also

- TCustomLiteUserFunction.OnStep

Request Support   DAC Forum   Provide Feedback

5.10.2.3 TLiteFunctionFinalEvent Procedure Reference

This type is used for the TCustomLiteUserFunction.OnFinal event.

Unit

`LiteAccess`

Syntax

```
TLiteFunctionFinalEvent = procedure (Sender: TObject; var
ResultValue: Variant) of object;
```

Parameters

Sender
   An object that raised the event.

ResultValue
   The resulting value of the function that will be passed back.

Request Support   DAC Forum   Provide Feedback

5.10.2.4 TLiteFunctionStepEvent Procedure Reference

This type is used for the

TCustomLiteUserFunction.OnStep event.

Unit

`LiteAccess`

Syntax

```
TLiteFunctionStepEvent = procedure (Sender: TObject; Params:
```

```
TDAParams) of object;
```

Parameters

Sender
   An object that raised the event.

Params
   Parameters that were passed to the function when it was executed in a SQL statement.

Request Support   DAC Forum   Provide Feedback

5.10.3 Enumerations

Enumerations in the **LiteAccess** unit.

Enumerations

Name	Description
TLiteFunctionKind	Specifies the kind of the used-defined function.
TLiteIsolationLevel	Specifies the way the transactions containing database modifications are handled.

Request Support   DAC Forum   Provide Feedback

5.10.3.1 TLiteFunctionKind Enumeration

Specifies the kind of the used-defined function.

Unit

LiteAccess

Syntax

```
TLiteFunctionKind = (fkScalar, fkAggregate);
```

Values

Value	Meaning

fkAggregate	Indicates that the function will be an aggregate function.
fkScalar	Indicates that the function will be a scalar function. The default value.

Request Support   DAC Forum   Provide Feedback

5.10.3.2 TLiteIsolationLevel Enumeration

Specifies the way the transactions containing database modifications are handled.

Unit

LiteAccess

Syntax

```
TLiteIsolationLevel = (lilDeferred, lilImmediate);
```

Values

Value	**Meaning**
lilDeferred	If the transaction contains DML that requires row locks held by another transaction, then the DML statement waits until the row locks are released. The default SQLite behavior.
lilImmediate	Specifies serializable transaction isolation mode as defined in the SQL92 standard. If a serializable transaction contains data manipulation language (DML) that attempts to update any resource that may have been updated in a transaction uncommitted at the start of the serializable transaction, then the DML statement fails.

Request Support   DAC Forum   Provide Feedback

5.11   LiteDacVcl

This unit contains the visual constituent of LiteDAC.

Classes

Name	**Description**

TLiteConnectDialog	A class that provides a dialog box for user to supply his login information.

Request Support   DAC Forum   Provide Feedback

5.11.1 Classes

Classes in the **LiteDacVcl** unit.

Classes

Name	Description
TLiteConnectDialog	A class that provides a dialog box for user to supply his login information.

Request Support   DAC Forum   Provide Feedback

5.11.1.1 TLiteConnectDialog 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 TLiteConnectDialog members.

Unit

LiteDacVcl

Syntax

```
TLiteConnectDialog = class(TCustomConnectDialog);
```

Remarks

The TLiteConnectDialog component is a direct descendant of TCustomConnectDialog class. Use TLiteConnectDialog to provide dialog box for user to supply database name and password (encryption key for the encrypted database). You may want to customize appearance of dialog box using this class's properties.

Inheritance Hierarchy

TCustomConnectDialog

   TLiteConnectDialog

See Also

- TLiteConnection.ConnectDialog

Request Support   DAC Forum   Provide Feedback

5.11.1.1.1 Members

TLiteConnectDialog 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.
DatabaseLabel	Used to specify a prompt for the 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.
Retries (inherited from **TCustomConnectDialog**)	Used to indicate the number of retries of failed connections.
ShowDatabase	Used to hide or show the database edit when the connect dialog appears.
ShowPassword	Used to hide or show the encryption key edit when the connect dialog appears.
StoreLogInfo (inherited from **TCustomConnectDialog**)	Used to specify whether the login information should be kept in system registry after a connection was

	established.

Methods

Name	Description
Execute (inherited from **TCustomConnectDialog**)	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.

Request Support   DAC Forum   Provide Feedback

5.11.1.1.2 Properties

Properties of the **TLiteConnectDialog** class.

For a complete list of the **TLiteConnectDialog** class members, see the TLiteConnectDialog 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.
DatabaseLabel	Used to specify a prompt for the 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.
Retries (inherited from **TCustomConnectDialog**)	Used to indicate the number of retries of failed connections.
ShowDatabase	Used to hide or show the database edit when the connect dialog appears.

[ShowPassword](#)	Used to hide or show the encryption key edit when the connect dialog appears.
[StoreLogInfo](#) (inherited from **[TCustomConnectDialog](#)**)	Used to specify whether the login information should be kept in system registry after a connection was established.

See Also

- [TLiteConnectDialog Class](#)

- [TLiteConnectDialog Class Members](#)

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

5.11.1.1.2.1 DatabaseLabel Property

Used to specify a prompt for the database edit.

Class

`TLiteConnectDialog`

Syntax

```
property DatabaseLabel: string;
```

Remarks

Use the DatabaseLabel property to specify a prompt for the database edit.

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

5.11.1.1.2.2 Show Database Property

Used to hide or show the database edit when the connect dialog appears.

Class

`TLiteConnectDialog`

Syntax

```
property ShowDatabase: boolean default True;
```

Remarks

Use the ShowDatabase property to specify whether the database edit will be shown or not when the connect dialog appears.

5.11.1.1.2.3 Show Password Property

Used to hide or show the encryption key edit when the connect dialog appears.

Class

[TLiteConnectDialog](#)

Syntax

```
property ShowPassword: boolean default True;
```

Remarks

Use the ShowPassword property to specify whether the encryption key edit will be shown or not when the connect dialog appears.

5.12   LiteDump

This unit contains implementation of the TLiteDump component

Classes

Name	Description
[TLiteDump](#)	A component for storing database or its parts as a script and also for restoring database from the received

	script.

Types

Name	Description
[TLiteDumpObjects](#)	Represents the set of TLiteDumpObject.

Enumerations

Name	Description
[TLiteDumpMode](#)	Specifies the mode of backup performed by TLiteDump.
[TLiteDumpObject](#)	Specifies the types of objects that are backuped by TLiteDump.

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

5.12.1 Classes

Classes in the **LiteDump** unit.

Classes

Name	Description
[TLiteDump](#)	A component for storing database or its parts as a script and also for restoring database from the received script.

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

5.12.1.1 TLiteDump 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 TLiteDump members.

Unit

LiteDump

Syntax

```
TLiteDump = class(TDADump);
```

Inheritance Hierarchy

TDADump

   TLiteDump

Request Support   DAC Forum   Provide Feedback

5.12.1.1.1 Members

TLiteDump 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.
Mode	Used to specify the mode of backup performed by TLiteDump.Used to specify the mode of backup performed by TLiteDump.
ObjectTypes	Used to specify the types of objects that are backuped by TLiteDump.
Options (inherited from **TDADump**)	Used to specify the behaviour of a TDADump component.

SQL (inherited from **TDADump**)	Used to set or get the dump script.
TableNames (inherited from **TDADump**)	Used to set the names of the tables to dump.

Methods

Name	Description
Backup (inherited from **TDADump**)	Dumps database objects to the TDADump.SQL property.
BackupQuery (inherited from **TDADump**)	Dumps the results of a particular query.
BackupToFile (inherited from **TDADump**)	Dumps database objects to the specified file.
BackupToStream (inherited from **TDADump**)	Dumps database objects to the stream.
Restore (inherited from **TDADump**)	Executes a script contained in the SQL property.
RestoreFromFile (inherited from **TDADump**)	Executes a script from a file.
RestoreFromStream (inherited from **TDADump**)	Executes a script received from the stream.

Events

Name	Description
OnBackupProgress (inherited from **TDADump**)	Occurs to indicate the TDADump.Backup, M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError (inherited from **TDADump**)	Occurs when SQLite raises some error on TDADump.Restore.
OnRestoreProgress (inherited from **TDADump**)	Occurs to indicate the TDADump.Restore, TDADump.RestoreFromFile, or

	TDADump.RestoreFromStream method execution progress.

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2 Properties

Properties of the **TLiteDump** class.

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

Public

Name	Description
Options (inherited from **TDADump**)	Used to specify the behaviour of a TDADump component.

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.
Mode	Used to specify the mode of backup performed by TLiteDump.Used to specify the mode of backup performed by TLiteDump.
ObjectTypes	Used to specify the types of objects that are backuped by TLiteDump.
SQL (inherited from **TDADump**)	Used to set or get the dump script.
TableNames (inherited from **TDADump**)	Used to set the names of the tables to dump.

See Also
- TLiteDump Class

- TLiteDump Class Members

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.1 Connection Property

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

Class

`TLiteDump`

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

See Also
- TLiteConnection

Request Support   DAC Forum   Provide Feedback

5.12.1.1.2.2 Mode Property

Used to specify the mode of backup performed by TLiteDump.Used to specify the mode of backup performed by TLiteDump.

Class

`TLiteDump`

Syntax

```
property Mode: TLiteDumpMode default dmAll;
```

Remarks

Use the Mode property to specify the mode of backup performed by TLiteDump. The default value is dmAll.

Request Support DAC Forum Provide Feedback

5.12.1.1.2.3 ObjectTypes Property

Used to specify the types of objects that are backuped by TLiteDump.

Class

TLiteDump

Syntax

```
property ObjectTypes: TLiteDumpObjects default [doTables,
doViews, doTriggers, doIndexes];
```

Remarks

Use the ObjectTypes property to specify the types of objects that are backuped by TLiteDump.

Request Support DAC Forum Provide Feedback

5.12.2 Types

Types in the **LiteDump** unit.

Types

Name	Description
TLiteDumpObjects	Represents the set of TLiteDumpObject.

Request Support DAC Forum Provide Feedback

5.12.2.1 TLiteDumpObjects Set

Represents the set of TLiteDumpObject.

Unit

[LiteDump](#)

Syntax

```
TLiteDumpObjects = set of TLiteDumpObject;
```

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

5.12.3 Enumerations

Enumerations in the **LiteDump** unit.

Enumerations

Name	Description
[TLiteDumpMode](#)	Specifies the mode of backup performed by TLiteDump.
[TLiteDumpObject](#)	Specifies the types of objects that are backuped by TLiteDump.

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

5.12.3.1 TLiteDumpMode Enumeration

Specifies the mode of backup performed by TLiteDump.

Unit

[LiteDump](#)

Syntax

```
TLiteDumpMode = (dmAll, dmData, dmSchema);
```

Values

Value	Meaning
dmAll	Backup of schema objects and table data is performed. The default value.
dmData	Backup of table data only is performed.
dmSchema	Backup of schema only is performed.

Request Support   DAC Forum   Provide Feedback

5.12.3.2 TLiteDumpObject Enumeration

Specifies the types of objects that are backuped by TLiteDump.

Unit

LiteDump

Syntax

```
TLiteDumpObject = (doTables, doViews, doTriggers, doIndexes);
```

Values

Value	Meaning
doIndexes	Backup of indexes is performed.
doTables	Backup of tables is performed.
doTriggers	Backup of triggers is performed.
doViews	Backup of views is performed.

Request Support   DAC Forum   Provide Feedback

5.13 LiteError

LiteError unit implements the class.

Classes

Name	Description

| ESQLiteError | A base class for exceptions raised when a component detects an SQLite error. |

Request Support   DAC Forum   Provide Feedback

5.13.1 Classes

Classes in the **LiteError** unit.

Classes

Name	Description
ESQLiteError	A base class for exceptions raised when a component detects an SQLite error.

Request Support   DAC Forum   Provide Feedback

5.13.1.1 ESQLiteError Class

A base class for exceptions raised when a component detects an SQLite error.

For a list of all members of this type, see ESQLiteError members.

Unit

LiteError

Syntax

```
ESQLiteError = class(EDAError);
```

Remarks

ESQLiteError is raised when a component detects an SQLite error. Use ESQLiteError in an exception handling block.

Inheritance Hierarchy

EDAError

ESQLiteError

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

5.13.1.1.1 Members

ESQLiteError 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.

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

5.14 LiteLoader

This unit contains implementation of the TLiteLoader component

Classes

Name	Description
TLiteLoader	TLiteLoader allows to load external data into the database.
TLiteLoaderOptions	This class allows setting up the behaviour of the TLiteLoader class.

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

5.14.1 Classes

Classes in the **LiteLoader** unit.

Classes

Name	Description
[TLiteLoader](#)	TLiteLoader allows to load external data into the database.
[TLiteLoaderOptions](#)	This class allows setting up the behaviour of the [TLiteLoader](#) class.

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

5.14.1.1 TLiteLoader Class

TLiteLoader allows to load external data into the database.

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

Unit

[LiteLoader](#)

Syntax

```
TLiteLoader = class(TDALoader);
```

Remarks

TLiteLoader allows to load external data into the SQLite database. To specify the name of the loading table set the TableName property. Use the Columns property to access individual columns. Write the OnGetColumnData or OnPutData event handler to read external data and pass it to the database. Call the Load method to start loading data.

Inheritance Hierarchy

[TDALoader](#)

　TLiteLoader

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

5.14.1.1.1 Members

[TLiteLoader](#) class overview.

Properties

Name	Description
AutoCommit	Used to automatically perform a commit after loading a certain amount of records.
AutoCommitRowCount	Used to specify the number of records, after which the transaction will be commited automatically.
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.
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**

5.14.1.1.2 Properties

Properties of the **TLiteLoader** class.

For a complete list of the **TLiteLoader** class members, see the TLiteLoader 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
AutoCommit	Used to automatically perform a commit after loading a certain amount of records.
AutoCommitRowCount	Used to specify the number of records, after which the transaction will be commited automatically.
Connection	Used to specify a connection object that will be used to connect to a data store.

See Also

- TLiteLoader Class

- TLiteLoader Class Members

Request Support | DAC Forum | Provide Feedback

5.14.1.1.2.1 AutoCommit Property

Used to automatically perform a commit after loading a certain amount of records.

Class

TLiteLoader

Syntax

```
property AutoCommit: boolean;
```

Remarks

Use the AutoCommit property to control whether a commit will be performed automatically after loading a number of records, specified by the TLiteLoader.AutoCommitRowCount property. When the property is set to True, a transaction implicitly starts before loading the block of records and commits automatically after records were loaded. This significantly increases the performance in the case of loading a large amount of records, as it is described in the SQLite documentation: http://www.sqlite.org/faq.html#q19 .

The default value is True.

See Also

- AutoCommitRowCount

Request Support | DAC Forum | Provide Feedback

5.14.1.1.2.2 AutoCommitRow Count Property

Used to specify the number of records, after which the transaction will be commited automatically.

Class

TLiteLoader

Syntax

```
property AutoCommitRowCount: integer default 10000;
```

Remarks

Use the AutoCommitRowCount property to specify the number of records, after which the transaction will be commited automatically when the TLiteLoader.AutoCommit property is set to True.

The default value is 1000.

Request Support   DAC Forum   Provide Feedback

5.14.1.1.2.3 Connection Property

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

Class

TLiteLoader

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

See Also

- TLiteConnection

Request Support   DAC Forum   Provide Feedback

5.14.1.2 TLiteLoaderOptions Class

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

For a list of all members of this type, see TLiteLoaderOptions members.

Unit

LiteLoader

Syntax

```
TLiteLoaderOptions = class(TDALoaderOptions);
```

Inheritance Hierarchy

TDALoaderOptions

   TLiteLoaderOptions

Request Support   DAC Forum   Provide Feedback

5.14.1.2.1 Members

TLiteLoaderOptions class overview.

Properties

Name	Description
QuoteNames	Used for TDALoader to quote table and column names in Insert SQL statements.
UseBlankValues (inherited from **TDALoaderOptions**)	Forces LiteDAC to fill the buffer with null values after loading a row to the database.

Request Support   DAC Forum   Provide Feedback

5.14.1.2.2 Properties

Properties of the **TLiteLoaderOptions** class.

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

Public

Name	Description
UseBlankValues (inherited from **TDALoaderOptions**)	Forces LiteDAC to fill the buffer with null values after loading a row to the database.

Published

Name	Description
QuoteNames	Used for TDALoader to quote table and column names in Insert SQL statements.

See Also

- TLiteLoaderOptions Class

- TLiteLoaderOptions Class Members

Request Support   DAC Forum   Provide Feedback

5.14.1.2.2.1 QuoteNames Property

Used for TDALoader to quote table and column names in Insert SQL statements.

Class

`TLiteLoaderOptions`

Syntax

```
property QuoteNames: boolean;
```

Request Support   DAC Forum   Provide Feedback

Devart. All Rights
Reserved.

5.15   LiteScript

This unit contains implementation of the component.

Classes

Name	Description
TLiteScript	Executes sequences of SQL statements.

Request Support   DAC Forum   Provide Feedback

5.15.1   Classes

Classes in the **LiteScript** unit.

Classes

Name	Description
TLiteScript	Executes sequences of SQL statements.

Request Support   DAC Forum   Provide Feedback

5.15.1.1   TLiteScript Class

Executes sequences of SQL statements.

For a list of all members of this type, see TLiteScript members.

Unit

LiteScript

Syntax

```
TLiteScript = class(TDAScript);
```

Inheritance Hierarchy

[TDAScript](#)

TLiteScript

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

5.15.1.1.1 Members

[**TLiteScript**](#) class overview.

Properties

Name	Description
[Connection](#)	Used to specify a connection object that will be used to connect to a data store.
[DataSet](#)	Refers to a dataset that holds the result set of query execution.
[Debug](#) (inherited from [**TDAScript**](#))	Used to display the script execution and all its parameter values.
[Delimiter](#) (inherited from [**TDAScript**](#))	Used to set the delimiter string that separates script statements.
[EndLine](#) (inherited from [**TDAScript**](#))	Used to get the current statement last line number in a script.
[EndOffset](#) (inherited from [**TDAScript**](#))	Used to get the offset in the last line of the current statement.
[EndPos](#) (inherited from [**TDAScript**](#))	Used to get the end position of the current statement.
[Macros](#) (inherited from [**TDAScript**](#))	Used to change SQL script text in design- or run-time easily.
[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.

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 SQLite raises an error.

Request Support   DAC Forum   Provide Feedback

Reserved.

5.15.1.1.2 Properties

Properties of the **TLiteScript** class.

For a complete list of the **TLiteScript** class members, see the [TLiteScript 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](#)	Refers to a dataset that holds the result set of query execution.
[Debug](#) (inherited from **[TDAScript](#)**)	Used to display the script execution and all its

	parameter values.
[Delimiter](#) (inherited from **[TDAScript](#)**)	Used to set the delimiter string that separates script statements.
[Macros](#) (inherited from **[TDAScript](#)**)	Used to change SQL script text in design- or run-time easily.
[SQL](#) (inherited from **[TDAScript](#)**)	Used to get or set script text.

See Also

- **[TLiteScript Class](#)**

- **[TLiteScript Class Members](#)**

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

5.15.1.1.2.1 Connection Property

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

Class

`TLiteScript`

Syntax

```
property Connection: TLiteConnection;
```

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 TLiteConnection objects. At runtime, set the Connection property to reference an existing TLiteConnection object.

See Also

- [TLiteConnection](#)

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

5.15.1.1.2.2 DataSet Property

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

Class

[TLiteScript](#)

Syntax

```
property DataSet: TCustomLiteDataSet;
```

Remarks

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

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

5.16 LiteSQLMonitor

This unit contains implementation of the component.

Classes

Name	Description
[TLiteSQLMonitor](#)	This component serves for monitoring dynamic SQL execution in LiteDAC-based applications.

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

5.16.1 Classes

Classes in the **LiteSQLMonitor** unit.

Classes

Name	Description
[TLiteSQLMonitor](#)	This component serves for

	monitoring dynamic SQL execution in LiteDAC-based applications.

Request Support   DAC Forum   Provide Feedback

5.16.1.1 TLiteSQLMonitor Class

This component serves for monitoring dynamic SQL execution in LiteDAC-based applications.

For a list of all members of this type, see TLiteSQLMonitor members.

Unit

LiteSQLMonitor

Syntax

```
TLiteSQLMonitor = class(TCustomDASQLMonitor);
```

Remarks

Use TLiteSQLMonitor to monitor dynamic SQL execution in LiteDAC-based applications. TLiteSQLMonitor provides two ways of displaying debug information: with dialog window, Using 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 TLiteSQLMonitor object. If an application has no TLiteSQLMonitor instance, the Debug window is available to display SQL statements to be sent.

Inheritance Hierarchy

TCustomDASQLMonitor
   TLiteSQLMonitor

Request Support   DAC Forum   Provide Feedback

5.16.1.1.1 Members

TLiteSQLMonitor class overview.

Properties

Name	Description
Active (inherited from **TCustomDASQLMonitor**)	Used to activate monitoring of SQL.
DBMonitorOptions (inherited from **TCustomDASQLMonitor**)	Used to set options for dbMonitor.
Options (inherited from **TCustomDASQLMonitor**)	Used to include the desired properties for TCustomDASQLMonitor.
TraceFlags (inherited from **TCustomDASQLMonitor**)	Used to specify which database operations the monitor should track in an application at runtime.

Events

Name	Description
OnSQL (inherited from **TCustomDASQLMonitor**)	Occurs when tracing of SQL activity on database components is needed.

Request Support   DAC Forum   Provide Feedback

5.17   MemData

This unit contains classes for storing data in memory.

Classes

Name	Description
TAttribute	TAttribute is not used in LiteDAC.
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.

5.17.1 Classes

Classes in the **MemData** unit.

Classes

Name	Description
[TAttribute](#)	TAttribute is not used in LiteDAC.
[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.

5.17.1.1 TAttribute Class

TAttribute is not used in LiteDAC.

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

5.17.1.1.1 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**

5.17.1.1.2 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](#)

5.17.1.1.2.1 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](#)

5.17.1.1.2.2 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.

5.17.1.1.2.3   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.

5.17.1.1.2.4   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

5.17.1.1.2.5 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

5.17.1.1.2.6 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

5.17.1.1.2.7 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**

5.17.1.1.2.8 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**

5.17.1.1.2.9 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

5.17.1.2 TBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types.

For a list of all members of this type, see TBlob members.

Unit

MemData

Syntax

```
TBlob = class(TSharedObject);
```

Remarks

Object TBlob holds large object value for the field and parameter dtBlob, dtMemo, dtWideMemo data types.

Inheritance Hierarchy

TSharedObject

    TBlob

See Also

- TMemDataSet.GetBlob

Request Support   DAC Forum   Provide Feedback

5.17.1.2.1 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](#)

5.17.1.2.2 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**

5.17.1.2.2.1 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**

5.17.1.2.2.2 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](#)

5.17.1.2.2.3 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](#)

5.17.1.2.2.4 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**

5.17.1.2.3 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

5.17.1.2.3.1 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

5.17.1.2.3.2 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.

5.17.1.2.3.3 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

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

5.17.1.2.3.4 LoadFromStream Method

Copies the contents of a stream into the TBlob object.

Class

TBlob

Syntax

```
procedure LoadFromStream(Stream: TStream); virtual;
```

Parameters

Stream
  Holds the specified stream from which the field's value is copied.

Remarks

Call the LoadFromStream method to copy the contents of a stream into the TBlob object.

Specify the stream from which the field's value is copied as the value of the Stream parameter.

See Also

- SaveToStream

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

5.17.1.2.3.5 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

5.17.1.2.3.6 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

5.17.1.2.3.7 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

5.17.1.2.3.8 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](#)

5.17.1.2.3.9 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

5.17.1.3 TCompressedBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.

For a list of all members of this type, see TCompressedBlob members.

Unit

MemData

Syntax

```
TCompressedBlob = class(TBlob);
```

Remarks

TCompressedBlob is a descendant of the TBlob class. It holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data. For more information about using BLOB compression see TCustomDADataSet.Options.

Note: Internal compression functions are available in CodeGear Delphi 2007 for Win32, Borland Developer Studio 2006, Borland Delphi 2005, and Borland Delphi 7. To use BLOB compression under Borland Delphi 6 and Borland C++ Builder you should use your own compression functions. To use them set the CompressProc and UncompressProc variables declared in the MemUtils unit.

Example

```
type
    TCompressProc = function(dest: IntPtr; destLen: IntPtr; const source: In
    TUncompressProc = function(dest: IntPtr; destlen: IntPtr; source: IntPtr
var
    CompressProc: TCompressProc;
    UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

TSharedObject

  TBlob

    TCompressedBlob

See Also

- TBlob

- TMemDataSet.GetBlob

- TCustomDADataSet.Options

Request Support   DAC Forum   Provide Feedback

5.17.1.3.1 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

5.17.1.3.2 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

5.17.1.3.2.1 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

5.17.1.3.2.2 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

5.17.1.4 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](#)

5.17.1.4.1 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](#)

5.17.1.5 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

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

5.17.1.5.1 Members

TMemData class overview.

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

5.17.1.6 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

5.17.1.6.1 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

5.17.1.6.2 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

Request Support   DAC Forum   Provide Feedback

5.17.1.6.2.1 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.

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

5.17.1.6.2.2   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](#)

5.17.1.6.2.3   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

5.17.1.6.2.4 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

5.17.1.6.3 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

5.17.1.6.3.1 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

Request Support   DAC Forum   Provide Feedback

5.17.1.7 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

5.17.1.7.1 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**

5.17.1.7.2 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**

5.17.1.7.2.1 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

5.17.1.7.3 Methods

Methods of the **TSharedObject** class.

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

Public

Name	Description
AddRef	Increments the reference count for the number of references dependent on the TSharedObject object.
Release	Decrements the reference count.

See Also

- TSharedObject Class

- TSharedObject Class Members

Request Support   DAC Forum   Provide Feedback

5.17.1.7.3.1 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

5.17.1.7.3.2 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**

5.17.2 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**

5.17.2.1 TLocateExOptions Set

Represents the set of TLocateExOption.

Unit

MemData

Syntax

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

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

5.17.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

MemData

Syntax

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

5.17.3   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.

5.17.3.1   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

5.17.3.2 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

5.17.3.3 TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

Unit

MemData

Syntax

```
TDANumericType = (ntFloat, ntBCD, ntFmtBCD);
```

Values

Value	Meaning
ntBCD	Data is stored on the client side as currency and represented as TBCDField. This format allows storing data with precision up to 0,0001.

ntFloat	Data stored on the client side is in double format and represented as TFloatField. The default value.
ntFmtBCD	Data is represented as TFMTBCDField. TFMTBCDField gives greater precision and accuracy than TBCDField, but it is slower.

Request Support   DAC Forum   Provide Feedback

5.17.3.4 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

5.17.3.5 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.

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

5.17.3.6 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.

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

Reserved.

5.18 MemDS

This unit contains implementation of the TMemDataSet class.

Classes

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

Variables

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

Request Support   DAC Forum   Provide Feedback

5.18.1 Classes

Classes in the **MemDS** unit.

Classes

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

Request Support   DAC Forum   Provide Feedback

5.18.1.1 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

5.18.1.1.1 Members

TMemDataSet class overview.

Properties

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive	Specifies the upper and lower boundaries for a range.
LocalConstraints	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

[LocalUpdate](#)	Used to prevent implicit update of rows on database server.
[Prepared](#)	Determines whether a query is prepared for execution or not.
[Ranged](#)	Indicates whether a range is applied to a dataset.
[UpdateRecordTypes](#)	Used to indicate the update status for the current record when cached updates are enabled.
[UpdatesPending](#)	Used to check the status of the cached updates buffer.

Methods

Name	Description
[ApplyRange](#)	Applies a range to the dataset.
[ApplyUpdates](#)	Overloaded. Writes dataset's pending cached updates to a database.
[CancelRange](#)	Removes any ranges currently in effect for a dataset.
[CancelUpdates](#)	Clears all pending cached updates from cache and restores dataset in its prior state.
[CommitUpdates](#)	Clears the cached updates buffer.
[DeferredPost](#)	Makes permanent changes to the database server.
[EditRangeEnd](#)	Enables changing the ending value for an existing range.
[EditRangeStart](#)	Enables changing the starting value for an existing range.
[GetBlob](#)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is

	known.
[Locate](#)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
[LocateEx](#)	Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.
[Prepare](#)	Allocates resources and creates field components for a dataset.
[RestoreUpdates](#)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#)	Cancels changes made to the current record when cached updates are enabled.
[SaveToXML](#)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetRange](#)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
[UnPrepare](#)	Frees the resources allocated for a previously prepared query on the server and client sides.
[UpdateResult](#)	Reads the status of the latest call to the ApplyUpdates method while

	cached updates are enabled.
[UpdateStatus](#)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
[OnUpdateError](#)	Occurs when an exception is generated while cached updates are applied to a database.
[OnUpdateRecord](#)	Occurs when a single update component can not handle the updates.

© 1997-2024
Devart. All Rights
Reserved.

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

5.18.1.1.2 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

5.18.1.1.2.1 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

5.18.1.1.2.2 IndexFieldNames Property

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

Class

TMemDataSet

Syntax

```
property IndexFieldNames: string;
```

Remarks

Use the IndexFieldNames property to get or set the list of fields on which the recordset is sorted. Specify the name of each column in IndexFieldNames to use as an index for a table. Column names order is significant. Separate names with semicolons. The specified columns don't need to be indexed. Set IndexFieldNames to an empty string to reset the recordset to the sort order originally used when the recordset's data was first retrieved.

Each field may optionally be followed by the keyword ASC / DESC or CIS / CS / BIN.

Use ASC, DESC keywords to specify a sort order for the field. If one of these keywords is not used, the default sort order for the field is ascending.

Use CIS, CS or BIN keywords to specify the sort type for string fields:

CIS - compare without case sensitivity;

CS - compare with case sensitivity;

BIN - compare by character ordinal values (this comparison is also case sensitive).

If a dataset uses a TCustomDAConnection component, the default value of the sort type depends on the TCustomDAConnection.Options option of the connection. If a dataset does not use a connection (TVirtualTable dataset), the default is CS.

Read IndexFieldNames to determine the field or fields on which the recordset is sorted.

Sorting is performed locally.

Note:

You cannot sort by BLOB fields.

IndexFieldNames cannot be set to True when TCustomDADataSet.UniDirectional=True.

Example

The following procedure illustrates how to set IndexFieldNames in response to a button click:

```
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

Request Support   DAC Forum   Provide Feedback

5.18.1.1.2.3 KeyExclusive Property

Specifies the upper and lower boundaries for a range.

Class

TMemDataSet

Syntax

```
property KeyExclusive: Boolean;
```

Remarks

Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values.

By default, KeyExclusive is False, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to True.

See Also

- SetRange

- SetRangeEnd

- SetRangeStart

Request Support   DAC Forum   Provide Feedback

5.18.1.1.2.4 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.

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

5.18.1.1.2.5 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.

© 1997-2024   Request Support   DAC Forum   Provide Feedback

5.18.1.1.2.6 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.

See Also

- Prepare

Request Support   DAC Forum   Provide Feedback

5.18.1.1.2.7 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

5.18.1.1.2.8 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

5.18.1.1.2.9 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

5.18.1.1.3 Methods

Methods of the **TMemDataSet** class.

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

Public

Name	Description
ApplyRange	Applies a range to the dataset.
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange	Removes any ranges currently in effect for a dataset.
CancelUpdates	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates	Clears the cached updates buffer.
DeferredPost	Makes permanent changes to the database server.

[EditRangeEnd](#)	Enables changing the ending value for an existing range.
[EditRangeStart](#)	Enables changing the starting value for an existing range.
[GetBlob](#)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
[Locate](#)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
[LocateEx](#)	Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.
[Prepare](#)	Allocates resources and creates field components for a dataset.
[RestoreUpdates](#)	Marks all records in the cache of updates as unapplied.
[RevertRecord](#)	Cancels changes made to the current record when cached updates are enabled.
[SaveToXML](#)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
[SetRange](#)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#)	Indicates that subsequent assignments to field values

	specify the start of the range of rows to include in the dataset.
[UnPrepare](#)	Frees the resources allocated for a previously prepared query on the server and client sides.
[UpdateResult](#)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
[UpdateStatus](#)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TMemDataSet Class](#)

- [TMemDataSet Class Members](#)

© 1997-2024 Devart. All Rights Reserved.   [Request Support](#)   [DAC Forum](#)   [Provide Feedback](#)

5.18.1.1.3.1 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](#)

5.18.1.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

Class

[TMemDataSet](#)

Overload List

Name	Description
[ApplyUpdates](#)	Writes dataset's pending cached updates to a database.
[ApplyUpdates(**const** UpdateRecKinds: TUpdateRecKinds)](#)	Writes dataset's pending cached updates of specified records to a database.

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

Writes dataset's pending cached updates to a database.

Class

[TMemDataSet](#)

Syntax

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

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

Note: The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

Example

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

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

See Also

- TMemDataSet.CachedUpdates

- TMemDataSet.CancelUpdates

- TMemDataSet.CommitUpdates

- TMemDataSet.UpdateStatus

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

5.18.1.1.3.3 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

5.18.1.1.3.4 CancelUpdates Method

Clears all pending cached updates from cache and restores dataset in its prior state.

Class

[TMemDataSet](#)

Syntax

```
procedure CancelUpdates;
```

Remarks

Call the CancelUpdates method to clear all pending cached updates from cache and restore dataset in its prior state.

It restores the dataset to the state it was in when the table was opened, cached updates were last enabled, or updates were last successfully applied to the database.

When a dataset is closed, or the CachedUpdates property is set to False, CancelUpdates is called automatically.

See Also

- [CachedUpdates](#)

- [TMemDataSet.ApplyUpdates](#)

- [UpdateStatus](#)

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

5.18.1.1.3.5 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

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

5.18.1.1.3.6 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**

5.18.1.1.3.7 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**

5.18.1.1.3.8 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

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

5.18.1.1.3.9 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

```
LiteQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

See Also

- TBlob

Request Support   DAC Forum   Provide Feedback

5.18.1.1.3.10 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

5.18.1.1.3.11 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.

    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 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

5.18.1.1.3.12 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 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

5.18.1.1.3.13 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](#)

5.18.1.1.3.14 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**

5.18.1.1.3.15 SaveToXML Method

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

Class

TMemDataSet

Overload List

Name	Description
SaveToXML(Destination: TStream)	Saves the current dataset data to a stream in the XML format compatible with ADO format.
SaveToXML(const FileName: string)	Saves the current dataset data to a file in the XML format compatible with ADO format.

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

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

Class

TMemDataSet

Syntax

```
procedure SaveToXML(Destination: TStream); overload;
```

Parameters

Destination
   Holds a TStream object.

Remarks

Call the SaveToXML method to save the current dataset data to a file or a stream in the XML format compatible with ADO format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToXML until the dataset is closed. This file can be read by other applications while it is opened, but they cannot write to the file.

When saving data to a stream, a TStream object must be created and its position must be set in a preferable value.

See Also

- TVirtualTable.LoadFromFile

- TVirtualTable.LoadFromStream

    Request Support   DAC Forum   Provide Feedback

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

Class

TMemDataSet

Syntax

```
procedure SaveToXML(const FileName: string); overload;
```

Parameters

FileName
   Holds the name of a destination file.

See Also

- TVirtualTable.LoadFromFile

- TVirtualTable.LoadFromStream

    Request Support   DAC Forum   Provide Feedback

5.18.1.1.3.16 SetRange Method

Sets the starting and ending values of a range, and applies it.

Class

TMemDataSet

Syntax

```
procedure SetRange(const StartValues: array of System.TVarRec;
const EndValues: array of System.TVarRec; StartExlusive: Boolean
= False; EndExclusive: Boolean = False);
```

Parameters

StartValues
  Indicates the field values that designate the first record in the range. In C++,
  StartValues_Size is the index of the last value in the StartValues array.

EndValues
  Indicates the field values that designate the last record in the range. In C++,
  EndValues_Size is the index of the last value in the EndValues array.

StartExlusive
  Iindicates the upper and lower boundaries of the start range.

EndExclusive
  Indicates the upper and lower boundaries of the end range.

Remarks

Call SetRange to specify a range and apply it to the dataset. The new range replaces the
currently specified range, if any.

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a
single procedure call. SetRange performs the following functions:

- 1. Puts the dataset into dsSetKey state.

- 2. Erases any previously specified starting range values and ending range values.

- 3. Sets the start and end range values.

- 4. Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are ignored when performing a search.

See Also
- ApplyRange

- CancelRange

- EditRangeEnd

- EditRangeStart

- IndexFieldNames

- KeyExclusive

- SetRangeEnd

- SetRangeStart

Request Support   DAC Forum   Provide Feedback

5.18.1.1.3.17 SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

```
procedure SetRangeEnd;
```

Remarks

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL.

Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call ApplyRange to activate the modified range.

See Also

- ApplyRange

- CancelRange

- EditRangeStart

- IndexFieldNames

- SetRange

- SetRangeStart

Request Support   DAC Forum   Provide Feedback

5.18.1.1.3.18 SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

```
procedure SetRangeStart;
```

Remarks

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL.

Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call ApplyRange to activate the modified range.

See Also

- ApplyRange

- CancelRange

- EditRangeStart

- IndexFieldNames

- SetRange

- SetRangeEnd

　　Request Support　　DAC Forum　　Provide Feedback

5.18.1.1.3.19 UnPrepare Method

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

Class

TMemDataSet

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free the resources allocated for a previously prepared query on the server and client sides.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- Prepare

　　Request Support　　DAC Forum　　Provide Feedback

5.18.1.1.3.20 UpdateResult Method

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

Class

[TMemDataSet](#)

Syntax

```
function UpdateResult: TUpdateAction;
```

Return Value

a value of the TUpdateAction enumeration.

Remarks

Call the UpdateResult method to read the status of the latest call to the ApplyUpdates method while cached updates are enabled. UpdateResult reflects updates made on the records that have been edited, inserted, or deleted.

UpdateResult works on the record by record basis and is applicable to the current record only.

See Also

* [CachedUpdates](#)

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

5.18.1.1.3.21 UpdateStatus Method

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

Class

[TMemDataSet](#)

Syntax

```
function UpdateStatus: TUpdateStatus; override;
```

Return Value

a value of the TUpdateStatus enumeration.

Remarks

Call the UpdateStatus method to determine the current update status for the dataset when

cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of the dataset.

See Also

- CachedUpdates

Request Support   DAC Forum   Provide Feedback

5.18.1.1.4 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

5.18.1.1.4.1 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](#)

5.18.1.1.4.2 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

5.18.2 Variables

Variables in the **MemDS** unit.

Variables

Name	Description
DoNotRaiseExcetionOnUaFail	An exception will be raised if the value of the UpdateAction parameter is uaFail.

	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.
[SendDataSetChangeEventAfterOpen](#)	

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

5.18.2.1 DoNotRaiseExcetionOnUaFail Variable

An exception will be raised if the value of the UpdateAction parameter is uaFail.

Unit

[MemDS](#)

Syntax

```
DoNotRaiseExcetionOnUaFail: boolean = False;
```

Remarks

Starting with LiteDAC , 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](#)

5.18.2.2 SendDataSetChangeEventAfterOpen Variable

The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

Unit

[MemDS](#)

Syntax

```
SendDataSetChangeEventAfterOpen: boolean = True;
```

Remarks

Starting with LiteDAC , 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

5.19   VirtualDataSet

This unit contains implementation of the TVirtualDataSet component.

Classes

Name	Description
TCustomVirtualDataSet	A base class for representation of arbitrary data in tabular form.
TVirtualDataSet	Dataset that processes arbitrary non-tabular data.

Types

Name	Description
TOnDeleteRecordEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.
TOnGetFieldValueEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.
TOnGetRecordCountEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.
TOnModifyRecordEvent	This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataS

et.OnModifyRecord events.

Request Support   DAC Forum   Provide Feedback

5.19.1 Classes

Classes in the **VirtualDataSet** unit.

Classes

Name	Description
TCustomVirtualDataSet	A base class for representation of arbitrary data in tabular form.
TVirtualDataSet	Dataset that processes arbitrary non-tabular data.

Request Support   DAC Forum   Provide Feedback

5.19.1.1 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

5.19.1.1.1 Members

TCustomVirtualDataSet class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML

	format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support   DAC Forum   Provide Feedback

5.19.1.2 TVirtualDataSet Class

Dataset that processes arbitrary non-tabular data.

For a list of all members of this type, see TVirtualDataSet members.

Unit

VirtualDataSet

Syntax

```
TVirtualDataSet = class(TCustomVirtualDataSet);
```

Inheritance Hierarchy

TMemDataSet

  TCustomVirtualDataSet

   TVirtualDataSet

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.19.1.2.1 Members

TVirtualDataSet class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is

	known.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while

	cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.19.2 Types

Types in the **VirtualDataSet** unit.

Types

Name	Description
TOnDeleteRecordEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.
TOnGetFieldValueEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.
TOnGetRecordCountEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.
TOnModifyRecordEvent	This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

5.19.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.

Unit

[VirtualDataSet](#)

Syntax

```
TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo:
Integer) of object;
```

Parameters

Sender
  An object that raised the event.

RecNo
  Number of the record being deleted.

5.19.2.2 TOnGetFieldValueEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.

Unit

[VirtualDataSet](#)

Syntax

```
TOnGetFieldValueEvent = procedure (Sender: TObject; Field: TField;
RecNo: Integer; out Value: Variant) of object;
```

Parameters

Sender
  An object that raised the event.

Field
  The field, which data has to be returned.

RecNo
  The number of the record, which data has to be returned.

Value
  Requested field value.

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

5.19.2.3 TOnGetRecordCountEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.

Unit

[VirtualDataSet](#)

Syntax

```
TOnGetRecordCountEvent = procedure (Sender: TObject; out Count:
Integer) of object;
```

Parameters

Sender
  An object that raised the event.

Count
  The number of records that the virtual dataset will contain.

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

5.19.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and
E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

Unit

[VirtualDataSet](#)

Syntax

```
TOnModifyRecordEvent = procedure (Sender: TObject; var RecNo:
Integer) of object;
```

Parameters

Sender

  An object that raised the event.

RecNo

  Number of the record being inserted or modified.

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

5.20 VirtualTable

This unit contains implementation of the TVirtualTable component.

Classes

Name	Description
TVirtualTable	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

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

5.20.1 Classes

Classes in the **VirtualTable** unit.

Classes

Name	Description
TVirtualTable	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

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

5.20.1.1 TVirtualTable Class

Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

For a list of all members of this type, see TVirtualTable members.

Unit

VirtualTable

Syntax

```
TVirtualTable = class(TMemDataSet);
```

Inheritance Hierarchy

TMemDataSet

  TVirtualTable

    Request Support   DAC Forum   Provide Feedback

5.20.1.1.1 Members

TVirtualTable class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
DefaultSortType	Used to determine the default type of local sorting for string fields.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a

	TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
Assign	Copies fields and data from another TDataSet component.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.

EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
LoadFromFile	Loads data from a file into a TVirtualTable component.
LoadFromStream	Copies data from a stream into a TVirtualTable component.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

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

5.20.1.1.2 Properties

Properties of the **TVirtualTable** class.

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

Public

Name	Description

CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Published

Name	Description
DefaultSortType	Used to determine the default type of local sorting for string fields.

See Also

- TVirtualTable Class

- TVirtualTable Class Members

© 1997-2024
Devart. All Rights
Reserved.

Request Support   DAC Forum   Provide Feedback

5.20.1.1.2.1 DefaultSortType Property

Used to determine the default type of local sorting for string fields.

Class

[TVirtualTable](#)

Syntax

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

Remarks

The DefaultSortType property is used when a sort type is not specified explicitly after the field name in the [TMemDataSet.IndexFieldNames](#) property of a dataset.

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

5.20.1.1.3 Methods

Methods of the **TVirtualTable** class.

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

Public

Name	Description
[ApplyRange](#) (inherited from **TMemDataSet**)	Applies a range to the dataset.
[ApplyUpdates](#) (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
[Assign](#)	Copies fields and data from another TDataSet component.
[CancelRange](#) (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
[CancelUpdates](#) (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior

	state.
[CommitUpdates](inherited) (inherited from **TMemDataSet**)	Clears the cached updates buffer.
[DeferredPost](inherited) (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
[EditRangeEnd](inherited) (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
[EditRangeStart](inherited) (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
[GetBlob](inherited) (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
[LoadFromFile](#)	Loads data from a file into a TVirtualTable component.
[LoadFromStream](#)	Copies data from a stream into a TVirtualTable component.
[Locate](inherited) (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
[LocateEx](inherited) (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.
[Prepare](inherited) (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
[RestoreUpdates](inherited) (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
[RevertRecord](inherited) (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
[SaveToXML](inherited) (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO

	format.
[SetRange](#) (inherited from **[TMemDataSet](#)**)	Sets the starting and ending values of a range, and applies it.
[SetRangeEnd](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
[SetRangeStart](#) (inherited from **[TMemDataSet](#)**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
[UnPrepare](#) (inherited from **[TMemDataSet](#)**)	Frees the resources allocated for a previously prepared query on the server and client sides.
[UpdateResult](#) (inherited from **[TMemDataSet](#)**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
[UpdateStatus](#) (inherited from **[TMemDataSet](#)**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- [TVirtualTable Class](#)

- [TVirtualTable Class Members](#)

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

5.20.1.1.3.1 Assign Method

Copies fields and data from another TDataSet component.

Class

`TVirtualTable`

Syntax

```
procedure Assign(Source: TPersistent); override;
```

Parameters

Source
   Holds the TDataSet component to copy fields and data from.

Remarks

Call the Assign method to copy fields and data from another TDataSet component.

Note: Unsupported field types are skipped (i.e. destination dataset will contain less fields than the source one). This may happen when Source is not a TVirtualTable component but some server-oriented dataset.

Example

```
Query1.SQL.Text := 'SELECT * FROM DEPT';
Query1.Active := True;
VirtualTable1.Assign(Query1);
VirtualTable1.Active := True;
```

See Also

* TVirtualTable

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

5.20.1.1.3.2 LoadFromFile Method

Loads data from a file into a TVirtualTable component.

Class

TVirtualTable

Syntax

```
procedure LoadFromFile(const FileName: string; LoadFields:
boolean = True; DecodeHTMLEntities: boolean = True);
```

Parameters

FileName
   Holds the name of the file to load data from.

LoadFields
  Indicates whether to load fields from the file.
DecodeHTMLEntities
  Indicates whether to decode HTML entities from the file.

Remarks

Call the LoadFromFile method to load data from a file into a TVirtualTable component. Specify the name of the file to load into the field as the value of the FileName parameter.This file may be an XML document in ADO-compatible format or in virtual table data format. The file format is detected automatically.

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

5.20.1.1.3.3 LoadFromStream Method

Copies data from a stream into a TVirtualTable component.

Class

`TVirtualTable`

Syntax

```
procedure LoadFromStream(Stream: TStream; LoadFields: boolean =
True; DecodeHTMLEntities: boolean = True);
```

Parameters

Stream
  Holds the stream from which the field's value is copied.
LoadFields
  Indicates whether to load fields from the stream.
DecodeHTMLEntities
  Indicates whether to decode HTML entities from the stream.

Remarks

Call the LoadFromStream method to copy data from a stream into a TVirtualTable component. Specify the stream from which the field's value is copied as the value of the Stream parameter. Data in the stream may be in ADO-compatible format or in virtual table data format. The data format is detected automatically.

Request Support   DAC Forum   Provide Feedback