Table of Contents

1 What's New

New Features in EntityDAC 3.5:

- Added support for RAD Studio 13 Florence

New Features in EntityDAC 3.4:

- Added support for RAD Studio 12 Athens Release 3
- Added support for the RAD Studio 64-bit IDE

New Features in EntityDAC 3.3:

- Added support for RAD Studio 12 Release 2
- Added support for macOS Sonoma
- Added support for iOS 17
- Added support for Android 13
- Fixed bug with executing LINQ queries which contain quantifiers using the dbExpress provider

New Features in EntityDac 3.2

- Added support for RAD Studio 11 Alexandria Release 3

New Features in EntityDac 3.1

- Added support for RAD Studio 11 Alexandria Release 2
- Added support for iOS Simulator ARM 64-bit target platform
- Added support for iOS 15
- Added support for Android 12

New Features in EntityDac 3.0

- RAD Studio 11 Alexandria Release 1 is supported
- macOS ARM is supported

New Features in EntityDac 2.4

- RAD Studio 10.4 Sydney is supported

New Features in EntityDac 2.3

- Android 64-bit is supported
- Now Trial editions for macOS and Linux are fully functional
- The Bytes attribute for the Guid generator is supported

New Features in EntityDac 2.2

- 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
- The Bytes attribute for the Guid generator is supported

New Features in EntityDac 2.1

- RAD Studio 10.3 Rio is supported

New Features in EntityDac 2.0

- iOS is supported
- Android is supported
- Linux is supported
- NexusDB data provider is added
- Now contexts are thread-safe
- Support for using TGUID as an entity key or as an expression argument is added
- Support for enum types is added
- Now the Contains method in LINQ expressions can accept a subquery or a set of expressions as an argument
- Now Entity Developer includes Firebird .NET Provider 5.12.1.0
- Published access level for an entity property is added

New Features in EntityDac 1.7

- Support for using TGUID as an entity key or as an expression argument is added
- Support for enum types is added
- Published access level for an entity property is added
- Now the Contains method in LINQ expressions can accept a subquery or a set of expressions as an argument
- Now EntityDeveloper includes Firebird .NET Provider 5.12.1.0

New Features in EntityDac 1.6

- RAD Studio 10.2 Tokyo is supported

New Features in EntityDac 1.5

- RAD Studio 10.1 Berlin is supported
- The ServerDataType property in the TMetaAttribute class is added
- Handling of attribute default values is improved
- The Options.CollectionOptions property is added for the Context component
- Cache disabling feature is improved

New Features in EntityDac 1.4

- RAD Studio 10 Seattle is supported
- Capability to disable entity caching with TDataContext.Cache.Enabled and TMetaType.AllowCaching properties is added

New Features in EntityDac 1.3

- Support for RAD Studio XE8 is added
- Now InterBase and Firebird are different SQL dialects
- Auto-Sync for entity property is supported
- Ability to reload Entity that was changed in database is added
- IdGenerator=Increment for entity property is added in EntityDeveloper

New Features in EntityDac 1.2

- Support for RAD Studio XE7 is added
- Firebird .NET Provider in EntityDeveloper is added
- Units are renamed:
  - EntityDAC.Common.CLRClasses.pas -> EntityDAC.Common.DacClasses.pas
  - EntityDAC.Common.MemData.pas -> EntityDAC.Common.DacMemData.pas
  - EntityDAC.Common.MemUtils.pas -> EntityDAC.Common.DacMemUtils.pas
  - EntityDAC.Common.Win32Timer.pas -> EntityDAC.Common.DacWin32Timer.pas
  - EntityDAC.Common.MemDS.pas -> EntityDAC.Common.DataSet.pas

New Features in EntityDac 1.1

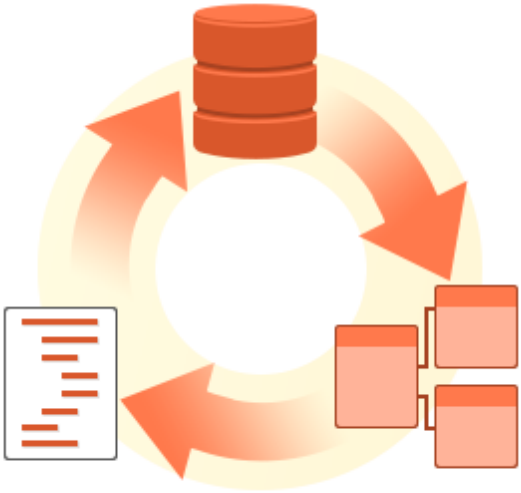
- First release of EntiDAC

Request Support DAC Forum Provide Feedback

2 General Information

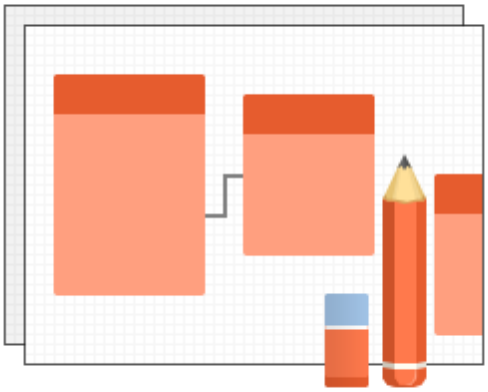
2.1 Overview

EntityDAC is an ORM for Delphi with LINQ support. It provides a powerful framework that allows to perform object-relational mapping of database tables to Delphi classes with full support for encapsulation, inheritance, polymorphism and other OOP features. To retrieve data from a database, LINQ is used as a database independent query engine. In addition, there is a feature-rich ORM modeling tool available - Entity Developer, that allows to create and edit your ORM models visually, and generate Delphi entity classes by this model automatically.



Database-First, Model-First and Code-First development

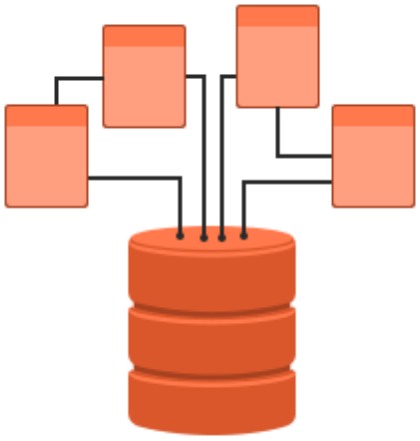
There are several approaches to database application development. Database-First - a database is developed, then a data model and application classes are generated on the basis of the existing database. Model-First - a data model is developed first in a model developer tool, then a database and

	application classes are generated by this model. Code-First - in this case, application classes are the original source, on the basis of which a database can be generated, as well as a model, if necessary. When developing applications using EntityDAC, you will be able to use all the three ways.
	Visual ORM Model Designer with Code generation Entity Developer

	allows you to create and edit ORM models visually, without typing a line of XML code or manual describing class attributes in Delphi code. It supports creation of all kinds of mapping, such as table splitting, mapping entity to several tables, complex types, inheritance hierarchies, etc. Code generation is very flexible due to using T4-like templates, allowing virtually anything you may want for code generation, and you may even create your own templates for other programming languages.

LINQ queries

Using ORM when developing your applications must not only accelerate development of the application itself, but also unify the application code and make applications independent of the specifics and syntax of the SQL database for which it is developed, that will allow to support multiple databases in your application with no effort. Therefore, Language Integrated Query (LINQ) is used in EntityDAC as a query language. Using LINQ also significantly simplifies writing and further support of queries,

	since, at this, the Code completion Delphi engine is used when typing LINQ keywords, class names, their attributes, etc. And in the same way, LINQ queries syntax check is performed at the stage of application compilation.
	## Class mapping Object-relational mapping of database tables to Delphi classes can be performed not only to classes inherited from a basic TEntity class, but to custom classes inherited form TObject as well. Such an approach

	allows using EntityDAC for development of new applications, as well as simply introduce ORM to already existing projects. In addition, different mapping methods are supported: code mapping, attribute mapping and XML mapping.
	## Entity & Query Cache To increase application performance, EntityDAC allows to cache metadata, all entities loaded from the database, LINQ queries, and much more. Such caching allows to avoid

	multiple loading of the same data and vastly improve performance in comparison to using the standard Data Access Components.

Request Support DAC Forum Provide Feedback

2.2 Features

Table of Contents

General usability:

- Powerful visual designer EntityDeveloper with code generation support.
- LINQ is used as database independent query language. Also LINQ queries can be written in the Delphi code using either the Code completion Delphi engine.

- Cross-database development - once developed application can connect to different databases without any code modifications.
- As a base class for mapped objects can be used TEntity or any descendants from TObject.
- Meta type inheritance support.
- Cascade data saving is supported.
- Submit and rollback all changes in context are supported.
- TDataSet compatible components to manipulate data with standard and third-party data-aware controls binding support.
- Live-binding support for any components.
- Ability to bind controls and get data in design-time.
- Write database independent queries directly in the code using either the native Delphi syntax or a C# LINQ syntax.
- IEnumerable-based easy-to-manipulate entity collections.
- Powerful design-time capabilities.
- A set of TDataSet compatible components to manipulate data.
- Ability to bind controls and get data in design-time

Various target platforms support:

- Windows 32-bit and 64-bit
- macOS 64-bit and ARM (Apple Silicon M1)
- iOS Simulator ARM 64-bit
- Linux 64-bit
- iOS 64-bit
- Android 32-bit and 64-bit

Various development platforms support:

- FMX
- VCL

Multiply database support:

- Oracle
- SQL Server
- MySQL
- Firebird
- PostgreSQL
- SQLite
- etc. (see full list here - Compatibility)

Several approaches to database application development:

- Database-First
- Model-First
- Code-First development

Data types support:

- Ordinal
- Nullable
- Complex

Associations support:

- One-To-Many
- One-To-One
- Many-To-Many

For object mapping can be used:

- Code mapping
- Attribute mapping
- XML mapping

Devart data access components support:

- UniDAC
- ODAC
- SDAC
- MyDAC
- IBDAC
- PgDAC
- LiteDAC

Standard data access components support:

- ADO
- IBX
- dbExpress
- FireDAC

3rd-party data access components support:

- BDE
- DOA
- FIbPlus
- NexusDB
- Zeos

Request Support DAC Forum Provide Feedback

2.3 Requirements

- Before installing a new version of EntityDAC, uninstall any previous version of EntityDAC you may have. If you run into problems or have any compatibility questions, please email support@devart.com

Note: You can avoid performing EntityDAC uninstallation manually when upgrading to a new

version by directing the EntityDAC 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 EntityDACXX.exe /force, specifying the full path to the appropriate version of the installation program).

- When installing EntityDAC from the sources to Windows Vista or Windows 7, it is necessary to have full access to the EntityDAC folder.
- EntityDAC is installed for and can be used only in compatible IDEs.
- EntityDAC Standard and Professional editions are supplied with Entity Developer, that, in turn, requires .NET Framework 3.5 Service Pack 1 or higher.

Request Support DAC Forum Provide Feedback

2.4 Compatibility

IDE Compatibility

EntityDAC is compatible with the following IDEs:

- Embarcadero RAD Studio 13 Florence
  - Embarcadero Delphi 13 Florence for Windows
  - Embarcadero Delphi 13 Florence for macOS
  - Embarcadero Delphi 13 Florence for Linux
  - Embarcadero Delphi 13 Florence for iOS
  - Embarcadero Delphi 13 Florence for Android
- Embarcadero RAD Studio 12 Athens (Requires Release 1, Release 2 or Release 3)
  - Embarcadero Delphi 12 Athens for Windows
  - Embarcadero Delphi 12 Athens for macOS
  - Embarcadero Delphi 12 Athens for Linux
  - Embarcadero Delphi 12 Athens for iOS
  - Embarcadero Delphi 12 Athens for Android

- Embarcadero RAD Studio 11 (up to 11.3) Alexandria
  - Embarcadero Delphi 11 Alexandria for Windows
  - Embarcadero Delphi 11 Alexandria for macOS
  - Embarcadero Delphi 11 Alexandria for Linux
  - Embarcadero Delphi 11 Alexandria for iOS
  - Embarcadero Delphi 11 Alexandria for Android
- Embarcadero RAD Studio 10.4 Sydney
  - 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 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 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 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 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 RAD Studio XE8
  - Embarcadero Delphi XE8 for Windows
  - Embarcadero Delphi XE8 for macOS
  - Embarcadero Delphi XE8 for iOS
  - Embarcadero Delphi 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 RAD Studio XE6
  - Embarcadero Delphi XE6 for Windows
  - Embarcadero Delphi XE6 for macOS
  - Embarcadero Delphi XE6 for iOS
  - Embarcadero Delphi 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 RAD Studio XE4
  - Embarcadero Delphi XE4 for Windows
  - Embarcadero Delphi XE4 for macOS
  - Embarcadero Delphi XE4 for iOS
- Embarcadero RAD Studio XE3 (Requires Update 2)
  - Embarcadero Delphi XE3 for Windows
  - Embarcadero Delphi XE3 for macOS
- Embarcadero RAD Studio XE2 (Requires Update 4 Hotfix 1)
  - Embarcadero Delphi XE2 for Windows
  - Embarcadero Delphi XE2 for macOS
- Embarcadero RAD Studio XE
  - Embarcadero Delphi XE
- Embarcadero RAD Studio 2010
  - Embarcadero Delphi 2010
- CodeGear RAD Studio 2009 (Requires Update 3)
  - CodeGear Delphi 2009
- CodeGear RAD Studio 2007
  - CodeGear Delphi 2007 for Windows 32-bit

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

Supported Target Platforms

- Windows 32-bit and 64-bit
- macOS 64-bit and ARM (Apple Silicon M1)
- iOS Simulator ARM 64-bit
- Linux 64-bit
- iOS 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)

Devart Data Access Components Compatibility

- UniDAC
- ODAC
- SDAC
- MyDAC
- IBDAC
- PgDAC
- LiteDAC

Standard Data Access Components Compatibility

EntityDAC supports the following standard data access components:

- ADO
- IBX
- dbExpress
- FireDAC

Third-Party Data Access Components Compatibility

EntityDAC also supports various 3rd-party data access components:

- BDE
- DOA
- FIbPlus
- NexusDB
- Zeos

Entity Developer Database Compatibility

- Oracle 9 and higher
- SQL Server 2000 and higher
- MySQL 4.1 and higher
- Firebird 2 and higher
- PostgreSQL 8 and higher
- SQLite 3 and higher
- DB2 9.5 and higher

Entity Developer Providers Compatibility

Entity Developer supports the following ADO.NET providers:

- **Oracle:**
  - dotConnect for Oracle
  - OracleClient
  - Oracle Data Provider for .NET
- **SQL Server:**
  - SqlClient
- **SQL Server Compact:**
  - Microsoft data provider for SQL Server CE
- **MySQL:**
  - dotConnect for MySQL
  - MySQL Connector/Net

- **Firebird:**
  - Firebird ADO.NET Data Provider
- **PostgreSQL:**
  - dotConnect for PostgreSQL
  - Npgsql
- **SQLite:**
  - dotConnect for SQLite
  - System.Data.SQLite
- **Salesforce:**
  - dotConnect for Salesforce
- **DB2:**
  - dotConnect for DB2
  - IBM DB2 .NET Data Provider

Request Support DAC Forum Provide Feedback

2.5 Component List

EntityDAC includes the following components:

Standard EntityDAC Components

	TEntityConnection	Lets you set up and control connections to different servers. Also used for transaction control over sessions and for performing SQL queries to a database.
	TEntityXMLModel	Represents the meta-model in design-time. Used to set-up EntityDAC dataset components, such as TEntityTable and TEntityQuery.
	TEntityContext	Manages the entities. Used for creating, updating and deleting entities, retrieving and storing entities from/to the database, storing used entities in the cache for future use, destroying of unused entities.

	TEntityDataSet	Keeps data from an arbitrary source. Can contain either a single entity, or entity list. Can be used in run-time only.
	TEntityDataSource	Provides an interface for connecting data-aware controls on a form and EntityDAC dataset components.

Professional EntityDAC Components

	TEntityTable	Lets you retrieve and update entities of the single meta-type without writing LINQ statements.
	TEntityQuery	Uses LINQ statements to retrieve entities from database tables and pass it to one or more data-aware components through a TDataSource object. This component provides a mechanism for updating data.

Devart Data Providers

	TUniDACDataProvider	Links the data provider for Devart Universal Data Access Components to an application.
	TODACDataProvider	Links the data provider for Devart Oracle Data Access Components to an application.
	TSDACDataProvider	Links the data provider for Devart SQL Server Data Access Components to an application.
	TMyDACDataProvider	Links the data provider for Devart MySQL Data Access Components to an application.
	TIBDACDataProvider	Links the data provider for Devart InterBase Data Access Components to an application.
	TPgDACDataProvider	Links the data provider for Devart PostgreSQL Data Access Components to an application.
	TLiteDACDataProvider	Links the data provider for Devart SQLite Data Access Components to an application.

Standard Data Providers

	TADODataProvider	Links the data provider for ADO components to an application.

	TDBXDataProvider	Links the data provider for dbExpress components to an application.
	TIBXDataProvider	Links the data provider for InterBase Express components to an application.
	TFireDACDataProvider	Links the data provider for FireDAC to an application.

Request Support DAC Forum Provide Feedback

2.6 Hierarchy Chart

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

- TObject
  - TPersistent
    - TComponent
      - TEntityConnection
      - TCustomContext
        - TDataContext
          - TCustomEntityContext
            - TEntityContext
          - TCustomObjectContext
            - TObjectContext
      - TDataSet
        - TMemDataSet
          - TCustomVirtualDataSet

- TCustomEntityDataSet
  - TCustomEntityTable
    - TEntityTable
  - TEntityDataSet
  - TEntityQuery
- TCustomEntityModel
  - TEntityXMLModel
- TDataProvider
  - TUniDACDataProvider
  - TODACDataProvider
  - TSDACDataProvider
  - TMyDACDataProvider
  - TPgDACDataProvider
  - TIBDACDataProvider
  - TLiteDACDataProvider
  - TADODataProvider
  - TIBXDataProvider
  - TDBXDataProvider
  - TFireDACDataProvider
- TEDValue
- TEDValues
- TEntity
  - TMappedEntity
  - TUnmappedEntity
- TEntityAttribute
- TEntityAttributes
- TEntityReference

- TMappedReference
- TEntityReferences
  - TMappedReferences
- TMappedCollections
- TMetaData
  - TMetaModel
  - TMetaTable
    - TUnmappedMetaTable
  - TMetaType
    - TMappedMetaType
- T:Devart.EntityDAC.TMetaList
  - TMetaColumnList
    - TMetaColumns
  - TMetaTableList

Request Support DAC Forum Provide Feedback

2.7 Editions

EntityDAC comes in three editions: **Express**, **Standard** and **Professional**.

Express Edition is a free version of EntityDAC including Standard and Devart Data Providers, and some of the EntityDAC features for evaluation.

Standard Edition is a cost-effective solution for developers looking for a high-performance and feature-rich ORM for Delphi.

Professional Edition extends Standard Edition with several important design-time features and data-aware components.

You can get **Source Access** of all the component classes in EntityDAC by purchasing the special EntityDAC Professional Edition with Source Code.

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

The matrix below compares features of EntityDAC editions.

Features	Express	Standard	Professional
Entity Developer			
Visual ORM Model Designer	✗	✓	✓
Mapping			
Code-mapped Entities	✓	✓	✓
Attribute-mapped Entities	✓	✓	✓
XML-mapped Entities	✓	✓	✓
Attribute-mapped Objects	✓	✓	✓
Mapping Customization	✗	✓	✓
Metadata			
Ordinal Meta Types	✓	✓	✓
Complex Meta Types	✗	✓	✓
Meta Type Inheritance	✗	✓	✓
Generators	✓	✓	✓
One-To-Many	✓	✓	✓
One-To-One	✗	✓	✓
Many-To-Many	✗	✓	✓
Data Context			
Get Single Entity	✓	✓	✓

Get Entity List	✓	✓	✓
Save Entity	✓	✓	✓
Submit & Reject All Changes	✓	✓	✓
Direct SQL Execution	✓	✓	✓
Deferred Relation Load	✓	✓	✓
Create & Drop Database	✗	✗	✓
LINQ			
LINQ queries in Run-Time	✓	✓	✓
LINQ queries in Design-Time	✗	✓	✓
LINQ Code Insight in IDE	✗	✓	✓
Binding to Controls			
Run-Time Live-Binding	✓	✓	✓
Design-Time Live-Binding	✗	✓	✓
Binding to Data-Aware Controls			
Run-Time Binding	✗	✓	✓
Design-Time Binding	✗	✗	✓
Design-Time Components			
TEntityConnection	✗	✓	✓
TEntityModel	✗	✓	✓
TEntityContext	✗	✓	✓
Data-Aware Components			

TEntityDataSource	✗	✓	✓
TEntityDataSet	✗	✓	✓
TEntityTable	✗	✗	✓
TEntityQuery	✗	✗	✓
Devart Data Providers			
UniDAC Data Provider	✓	✓	✓
ODAC Data Provider	✓	✓	✓
SDAC Data Provider	✓	✓	✓
MyDAC Data Provider	✓	✓	✓
IBDAC Data Provider	✓	✓	✓
PgDAC Data Provider	✓	✓	✓
LiteDAC Data Provider	✓	✓	✓
Standard Data Providers			
ADO Data Provider	✓	✓	✓
IBX Data Provider	✓	✓	✓
dbExpress Data Provider	✓	✓	✓
FireDAC Data Provider	✓	✓	✓
Third-Party Data Providers			
BDE Data Provider	✗	✓	✓
DOA Data Provider	✗	✓	✓
FIbPlus Data Provider	✗	✓	✓
NexusDB Data Provider	✗	✓	✓

Zeos Data Provider	✗	✓	✓

Request Support DAC Forum Provide Feedback

2.8 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 EntityDAC 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 EntityDAC Express Edition is freeware. You are entitled to install and use the Software on any number of computers for any number of developers.

1.2 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").

1.3 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.

1.4 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 Trial edition that can be distributed for free as original Devart's EntityDAC 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 EntityDAC. You can distribute EntityDAC only in compiled Executable Programs or Dynamic-Link Libraries with required run-time libraries and packages.

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

4. TRANSFER

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

5. TERMINATION

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

6. EVALUATION

Devart may provide evaluation ("Trial") versions of the Software. You may transfer or distribute Trial versions of the Software as an original installation package only. If the Software you have obtained is marked as a "Trial" version, you may install and use the Software for a

period of up to 60 calendar days from the date of installation (the "Trial Period"), subject to the additional restriction that it is used solely for evaluation of the Software and not in conjunction with the development or deployment of any application in production. You may not use applications developed using Trial versions of the Software for any commercial purposes. Upon expiration of the Trial Period, the Software must be uninstalled, all its copies and all accompanying written materials must be destroyed.

7. WARRANTY

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

8. SUBSCRIPTION AND SUPPORT

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

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

9. COPYRIGHT

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

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

Request Support DAC Forum Provide Feedback

2.9 Trial Limitations

EntityDAC Trial edition is a fully functional Professional edition just limited by 60-day trial period and the following:

- only 5 metatypes can be created on the iOS and Android platforms.

After the trial period expires, you must either order a registered version or uninstall EntityDAC.

Request Support DAC Forum Provide Feedback

2.10 Getting Support

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

Support Options

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

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

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

EntityDAC Priority Support

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

To get help through the EntityDAC 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 you are using.
- Your EntityDAC Registration number.
- Full EntityDAC edition name and version number. You can find both of these in the About sheet of TEntityConnection Editor or from the EntityDAC | About menu.
- A detailed problem description.
- If possible, a small test project that reproduces the problem. It is recommended to use Scott or SYS schema objects only. Please include definitions for all and avoid using third-party components.

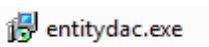
Request Support DAC Forum Provide Feedback

3 Getting Started

3.1 Installation

This part of the tutorial will describe the EntityDAC installation process.

To start the installation, run the downloaded installer, for example:

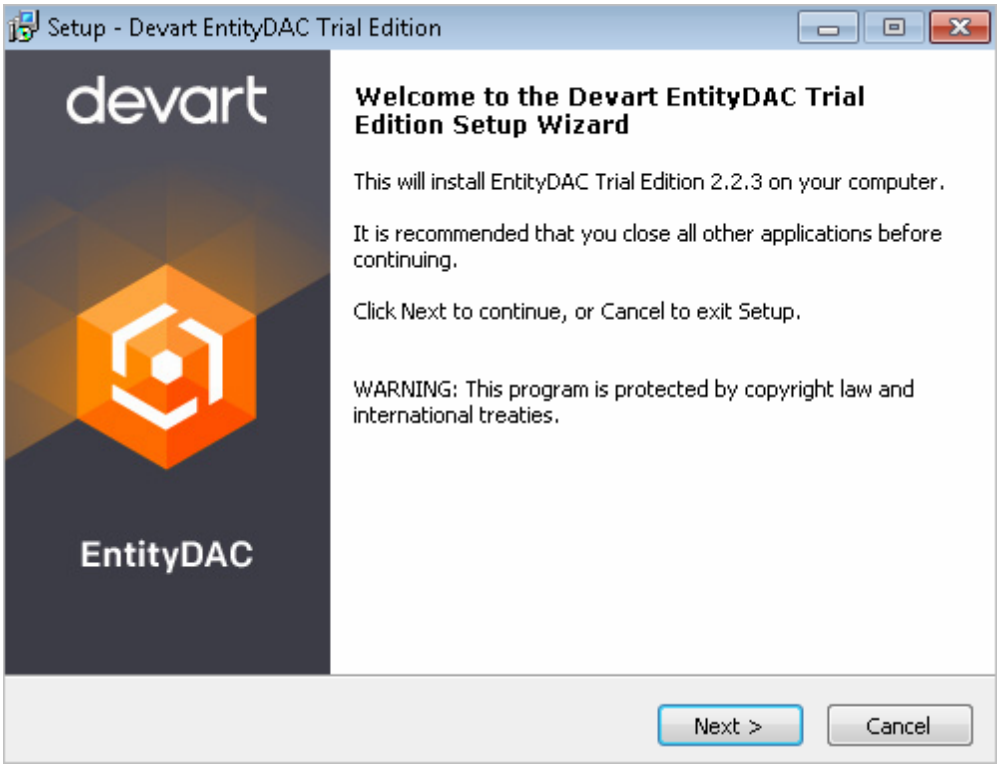


The installer file name may differ for different editions:

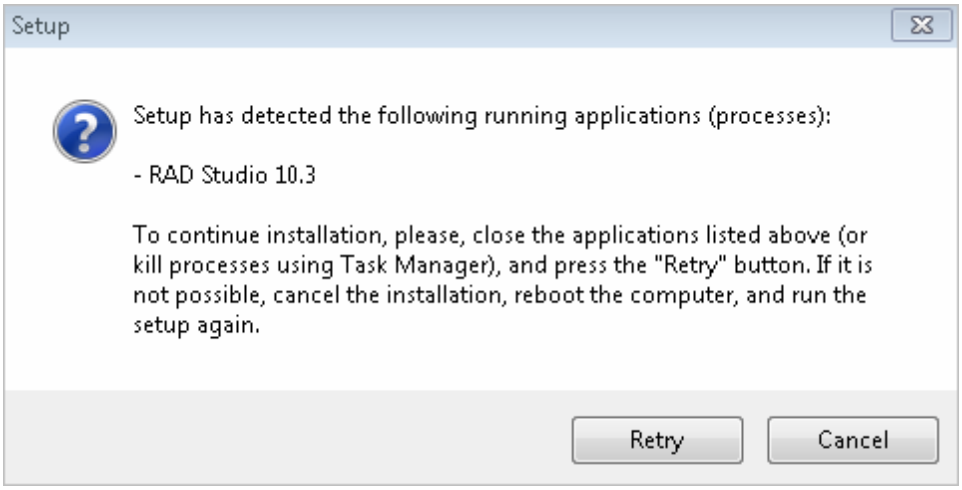
- EntityDAC Professional Edition - *entitydacXXpro.exe*
- EntityDAC Standard Edition - *entitydacXXstd.exe*
- EntityDAC Express Edition - *entitydacexpress.exe*
- EntityDAC Trial Edition - *entitydac.exe*

where XX - the current version of EntityDAC.

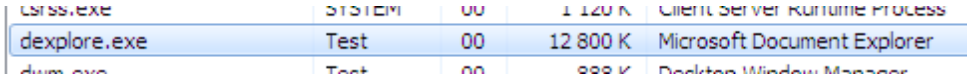
First wizard page informs us about the edition and version of the product being installed.



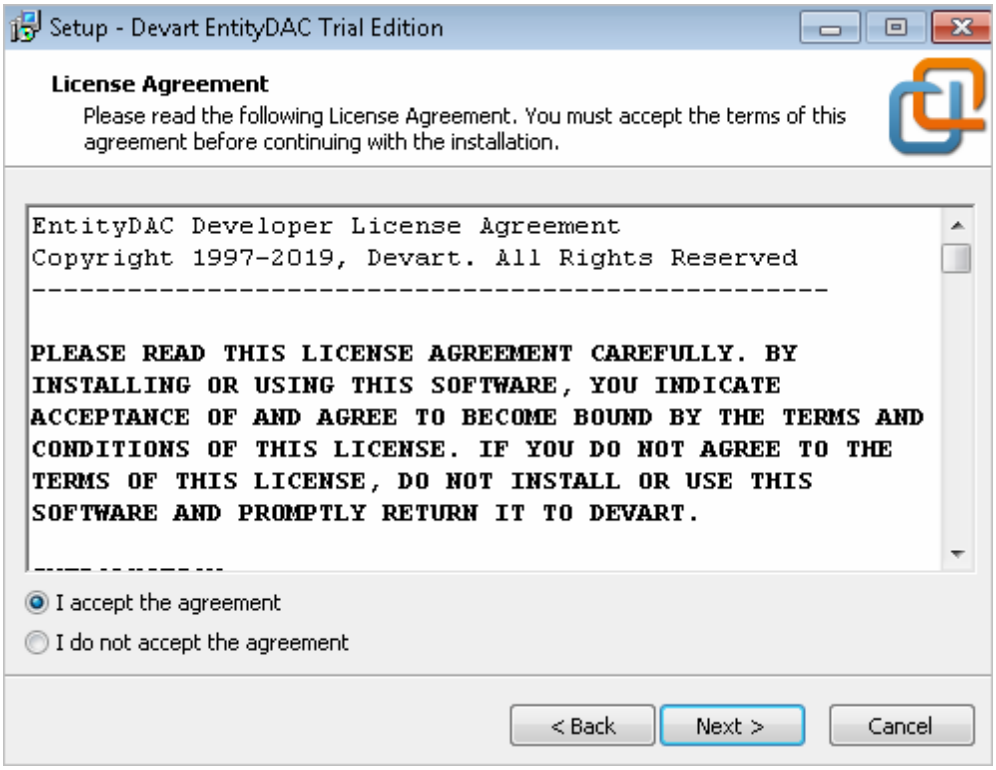
Note, that any using IDE has to be closed before starting the installation. Otherwise, the following error message will be displayed. In this case, you should close all running IDE and click the "Retry" button.



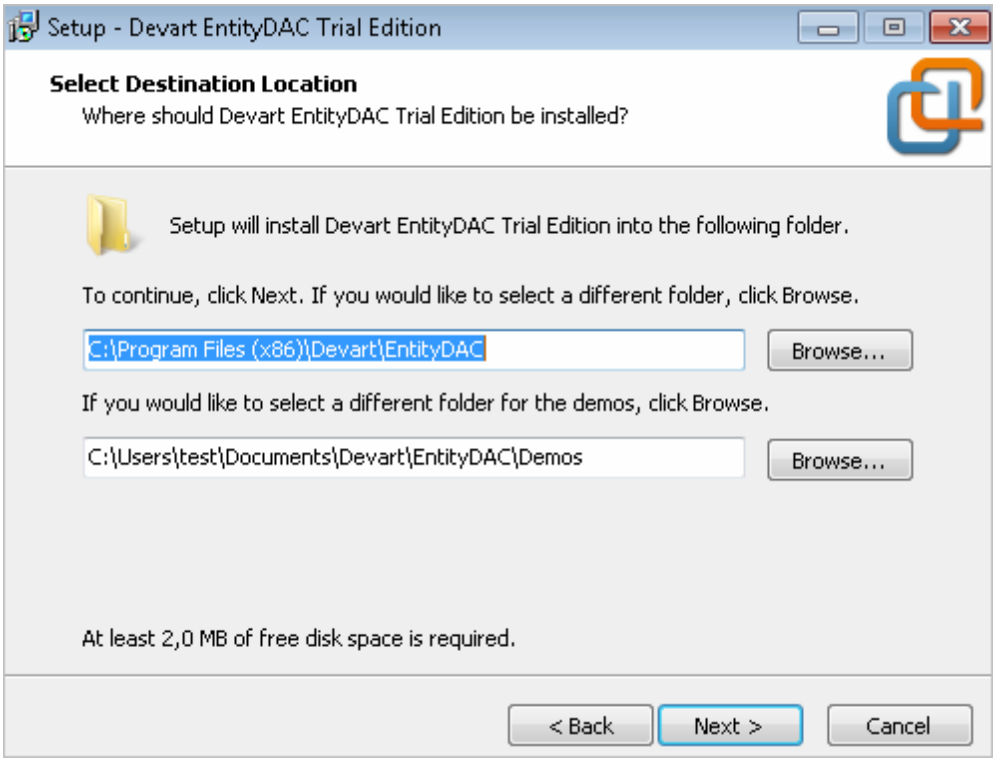
In some cases, after using RAD Studio help, the Microsoft Document Explorer process remains running even after IDE is closed. In this case, you should kill the "dexplore.exe" process manually using Task Manager. After doing this, click the "Retry" button in the setup dialog.



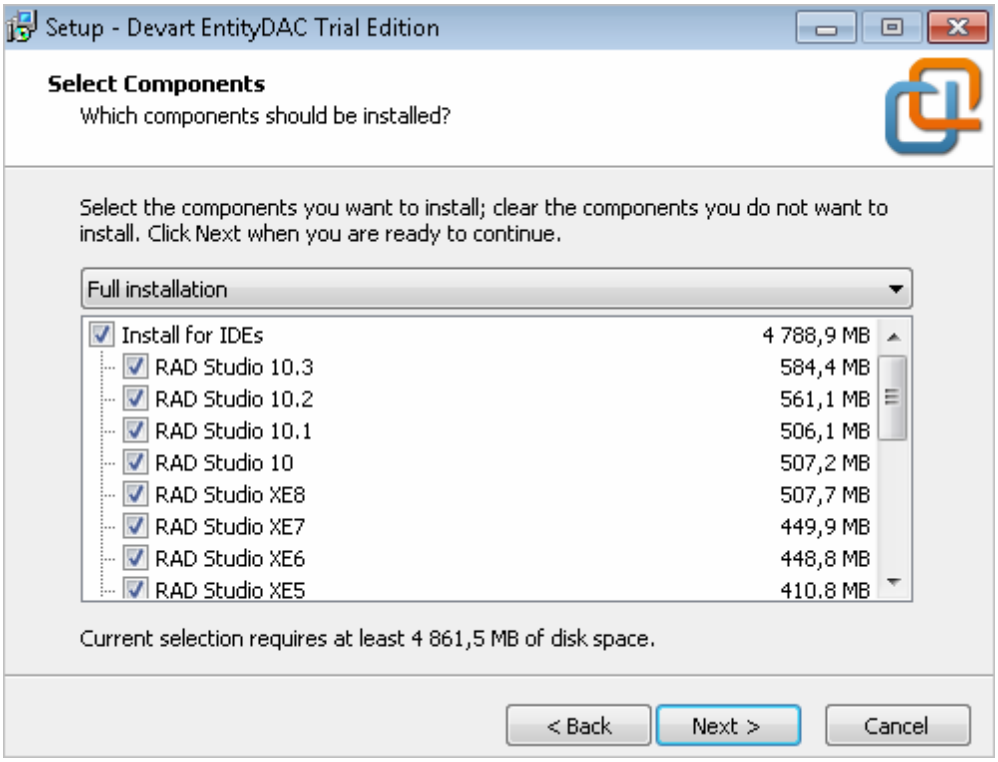
In the next wizard page, you should read and accept the license agreement.



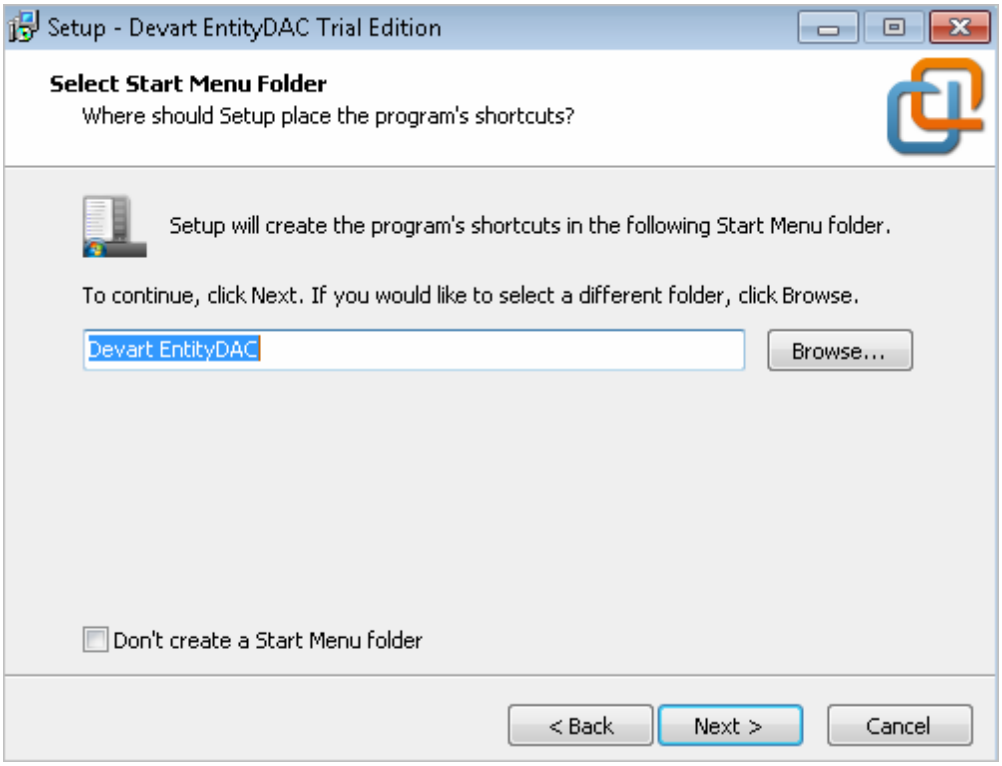
Next, you should specify the folder in which the product will be installed. By default, the folder is "Program Files\Devart\EntityDAC". Also, you can specify a folder for the EntityDAC demos.



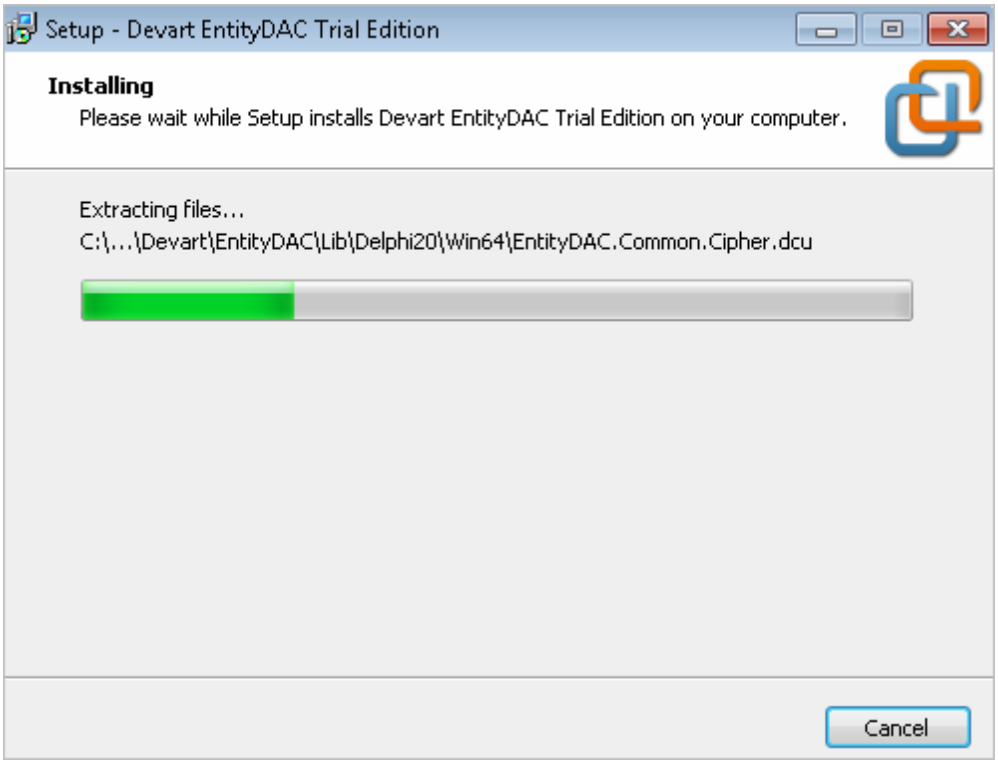
The next step is the component selection. You can select for which RAD Studio versions the product will be installed. Also, you can specify, if there is need to install Entity Developer for EntityDAC, demo projects and help files. By default, all these products will be installed automatically.



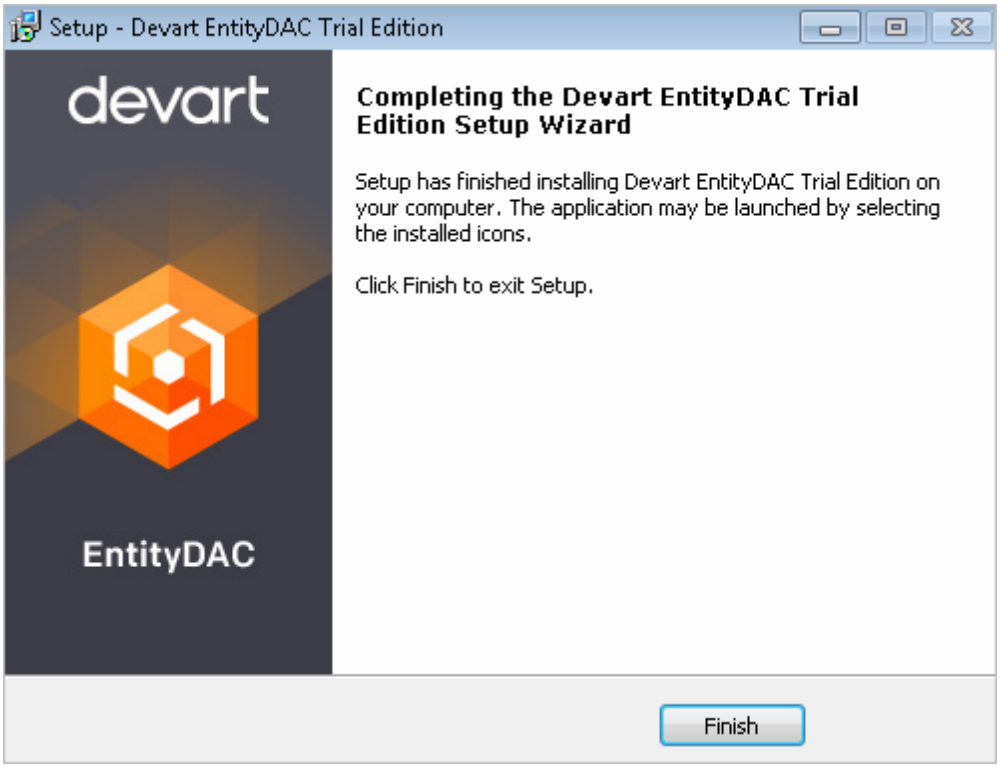
Then, you will be prompted to choose the Start Menu folder name for the installed product. By default, the name is "Devart EntityDAC".



After the confirmation page, the installation process will be started.



And finally, click the "Finish" button to end the installation.



Request Support DAC Forum Provide Feedback

3.2 Data Providers Installation

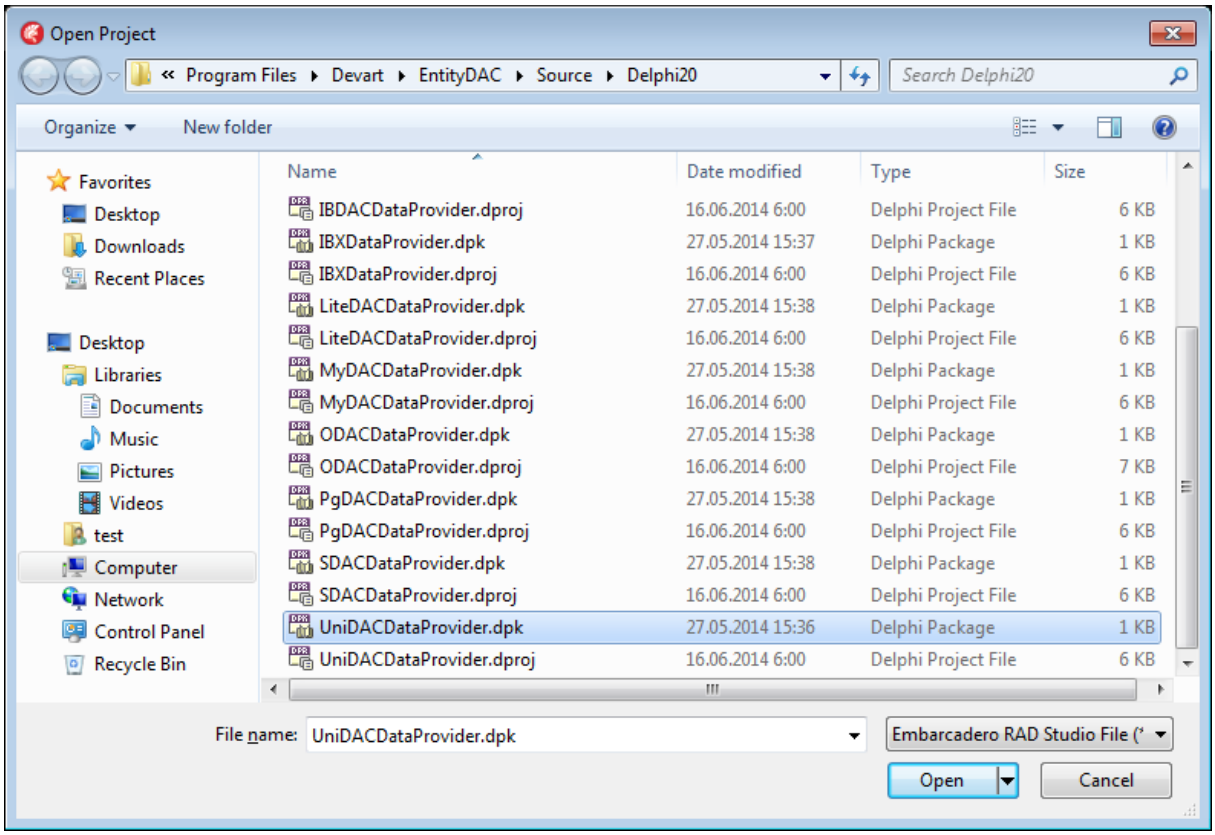
EntityDAC data providers are installed by default in Standard and Professional editions. However, there are several cases, when the data providers need to be installed manually:

- in case, if they were lost after broken installation or due to some other problems;
- in case of using EntityDAC edition with source code;
- in case you want to install 3rd-party data providers that are located in the Demo folder;
- in case of using Delphi Starter or Trial Edition (since the Starter and Trial Editions have no command line compiler).

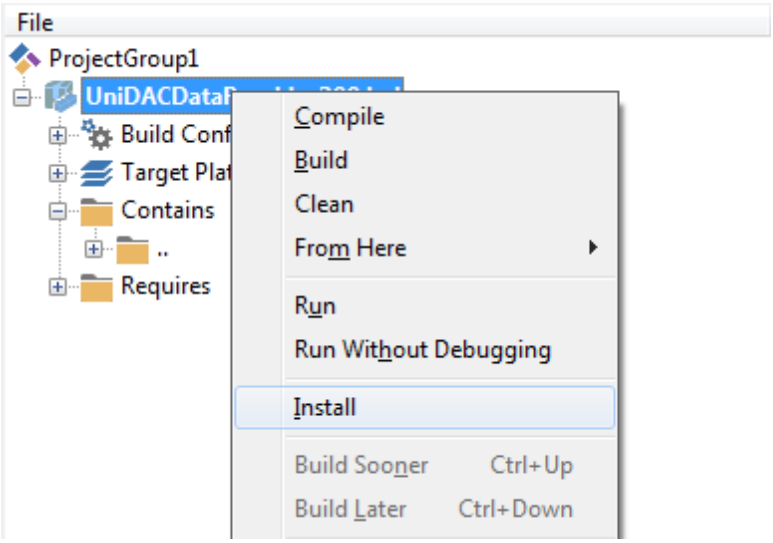
To install data providers manually:

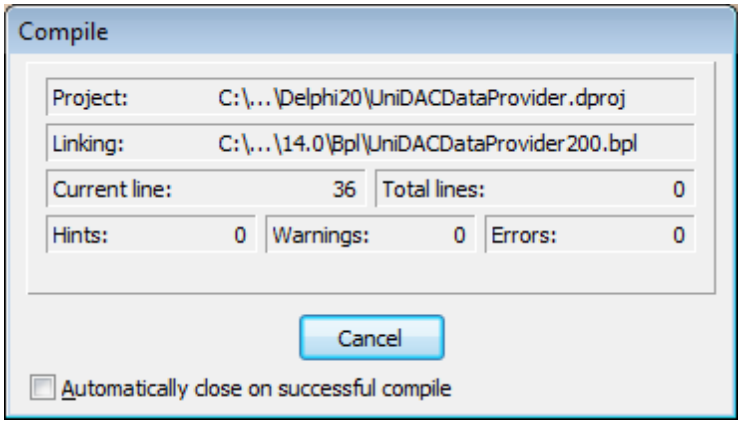
1. in RAD Studio click the "Open Project" (Ctrl+F11) button.
2. open the EntityDAC installation folder (C:\Program Files\Devart\EntityDAC by default) and

select your IDE version in the Source folder (e.g., Delphi20 - for XE6), and select the needed data provider project.

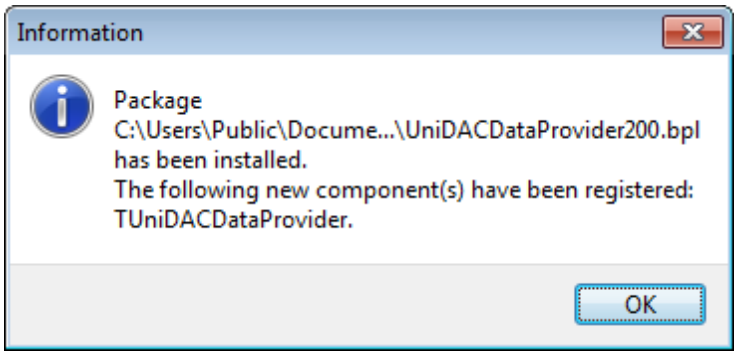


3. Right click on the project in the Project Manager, click "Install".

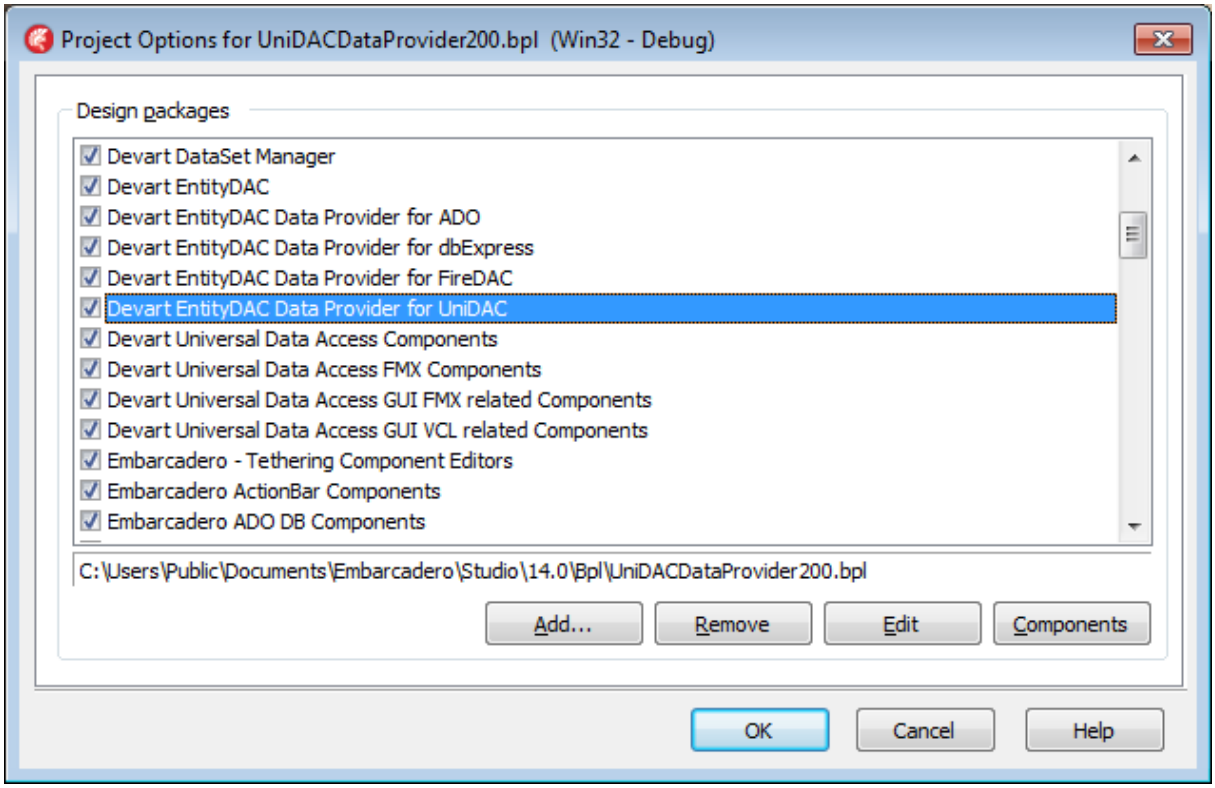




4. the data provider is installed and ready for use.



You can find it in the "Component->Install Packages" dialog:



 Request Support DAC Forum Provide Feedback

3.3 Creating My First Application

This tutorial aims to demonstrate how to develop an application using EntityDAC Standard edition.

If you have the Professional edition, you can read the Using Professional Edition article to learn how to use the advantages of the Professional edition in the application you will create in this tutorial.

If you are using EntityDAC Express edition, see the Creating My First Application With Express Edition tutorial.

 Request Support DAC Forum Provide Feedback

3.3.1 Generating Model and Entity Classes from Database

In this part of the tutorial we will create a new EntityDAC model from an existing SQLite database.

Prepare

To follow the tutorial you can use the demo SQLite database provided with EntityDAC. By default, the database is placed in the "Documents\Devart\EntityDAC\Demos\EntityDemo\DB\SQLite\" folder.

Or, you can create a new SQLite database named "demo.db3" manually using any SQLite management tool, and then create demo tables in it using the "create.sql" script provided with the EntityDAC demo (the script is placed in the "Documents\Devart\EntityDAC\Demos\EntityDemo\Scripts\SQLite\" folder by default).

As it is described in the "Database Connection" article, EntityDAC itself does not contain a database connectivity mechanisms. Therefore, you have to install either Devart Universal Data Access Components or Devart SQLite Data Access Components to be able to provide interaction between EntityDAC and SQLite database.

Run Entity Developer from the **EntityDAC** menu in RAD Studio.

To create a new model in Entity Developer, select menu "File -> New Model".

Step 1

The first step in the model creating wizard is to choose the type of the model. There are four predefined model types in EntityDAC:

1. Code-mapped entities.

In this case, for each database entity there will be generated an "entity" class, which is TEntity descendant fully managed by a data context. The model's metadata is generated in a separate unit as a set of special "metadata" classes which are code-linked to corresponding entity classes.

2. Attribute-mapped entities.

For this model type, entity classes will be generated as well, but the model's metadata classes are not generated. Instead, entity classes are marked with the special attributes,

and metadata will be generated automatically at run-time using these attributes.

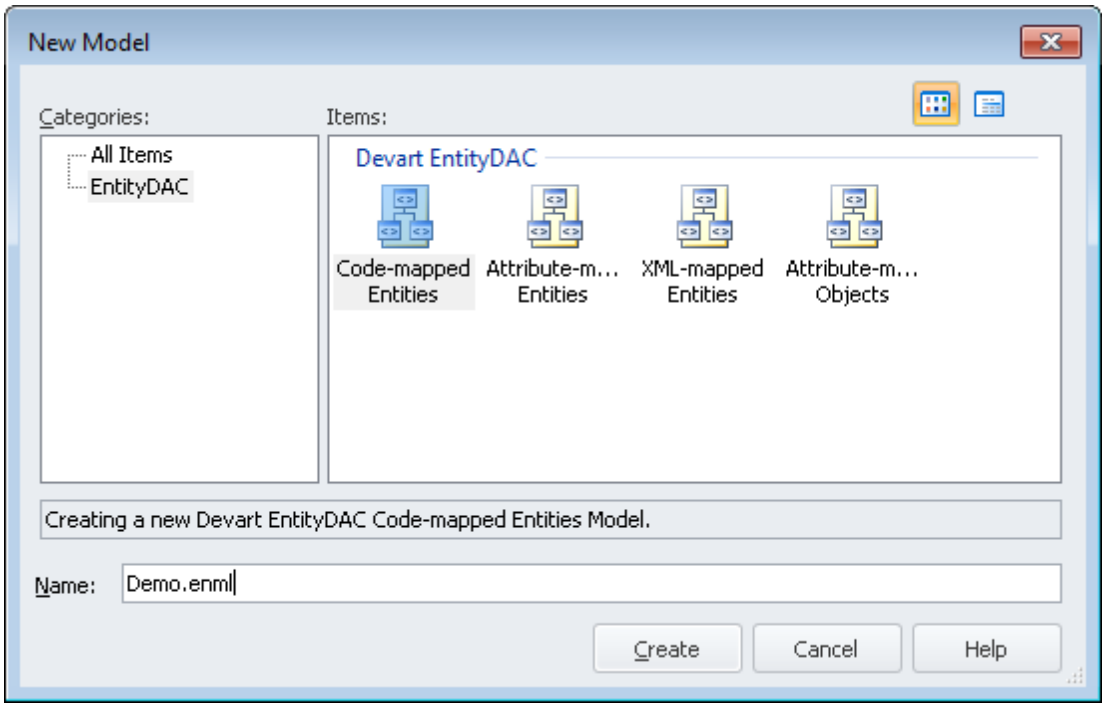
3. XML-mapped entities.

Only entity classes will be generated. The model metadata has to be specified as an external XML-file which can be generated using Entity Developer or created manually. The XML-file format is described in the "A XML-mapped entities" article.

4. Attribute-mapped objects.

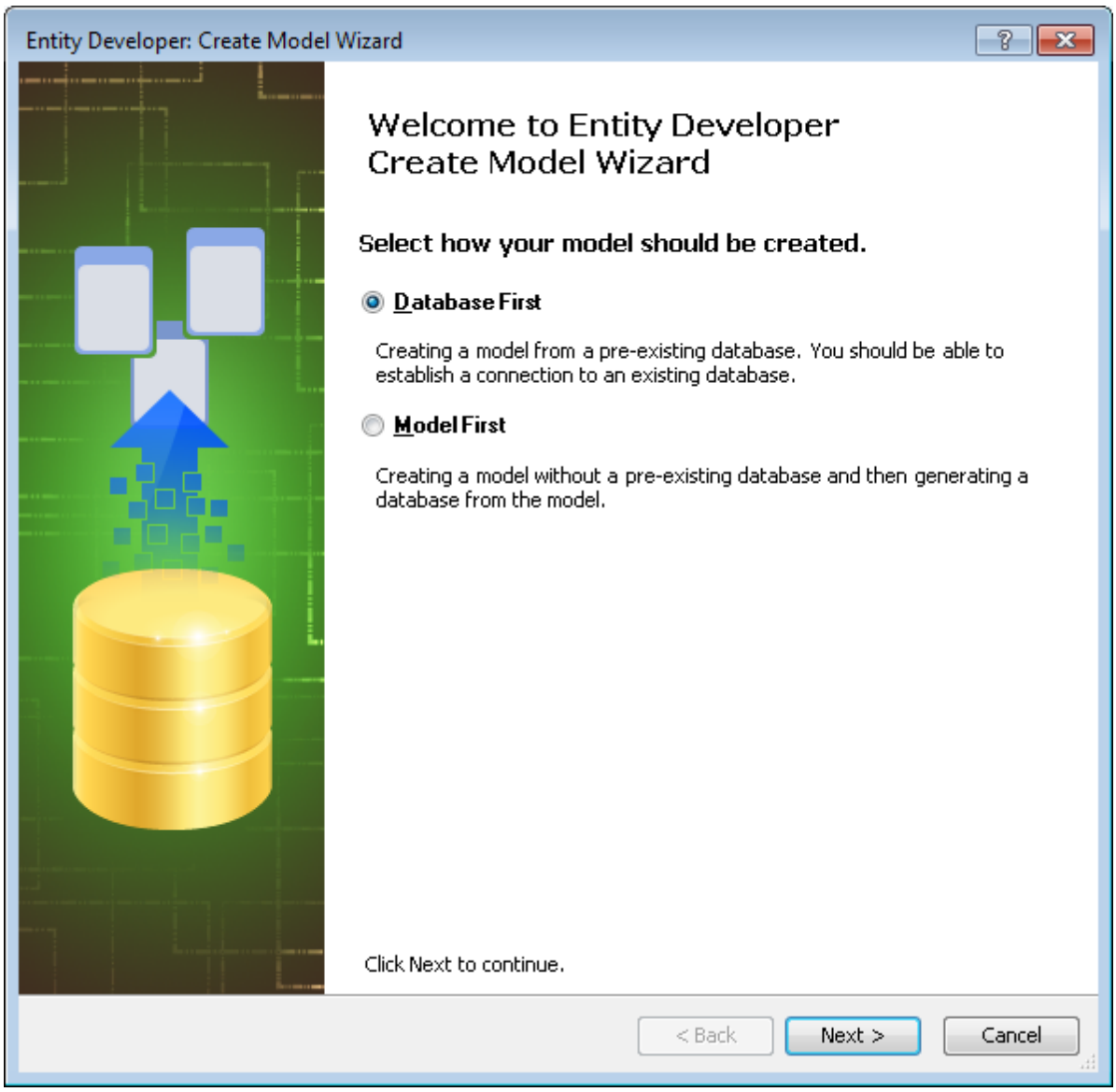
This type of model is similar to the "Attribute-mapped entities" model, but the generated classes are not TEntity, but TObject descendants. So, their life cycle management is a little different.

The Code-mapped entities model type is the most intuitive and easy to learn, so let's choose the "Code-mapped entities" model, set the project name to "Demo.odml" and press the "Create" button.



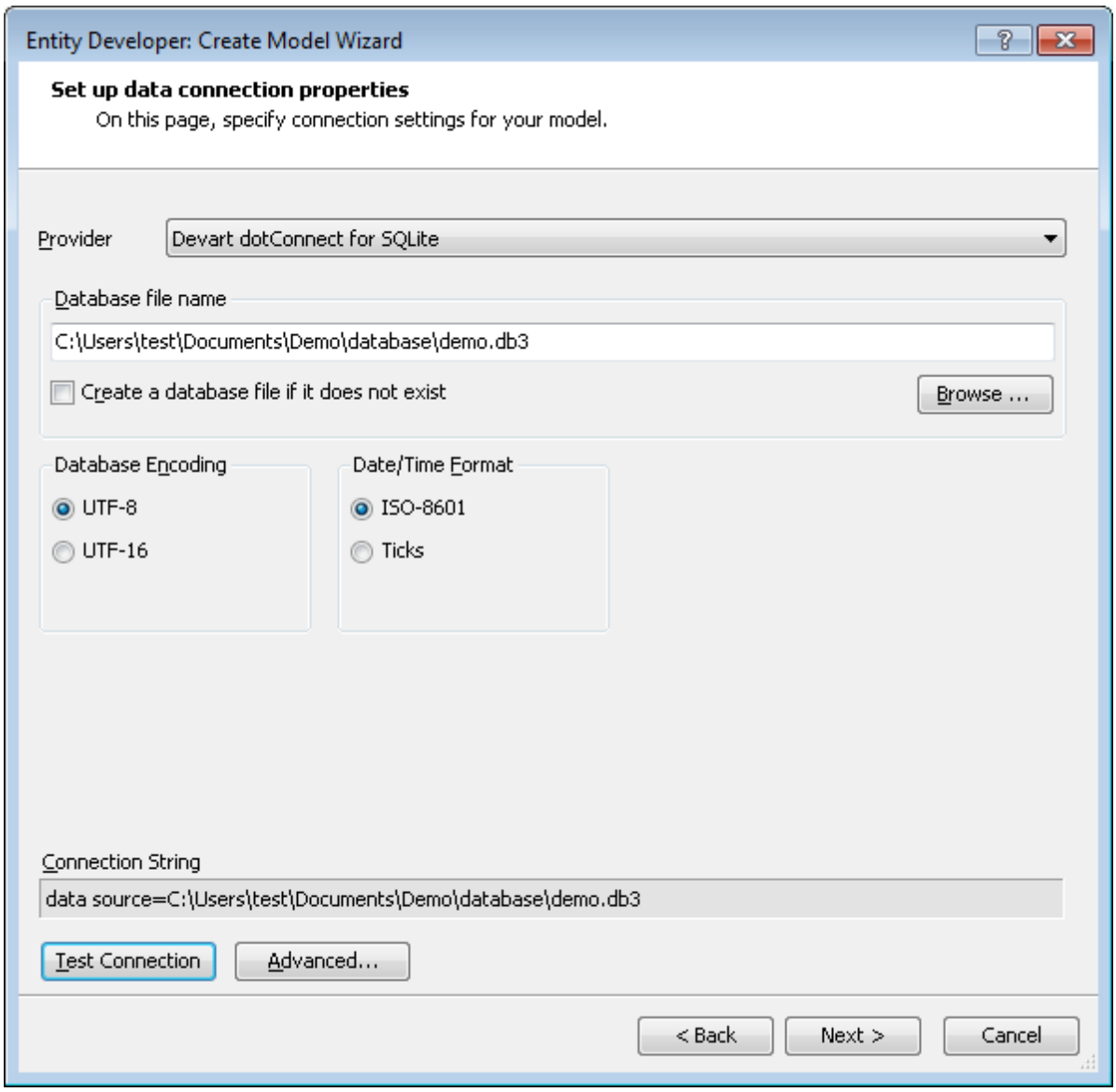
Step 2

Next, choose the "Database First" creating method.



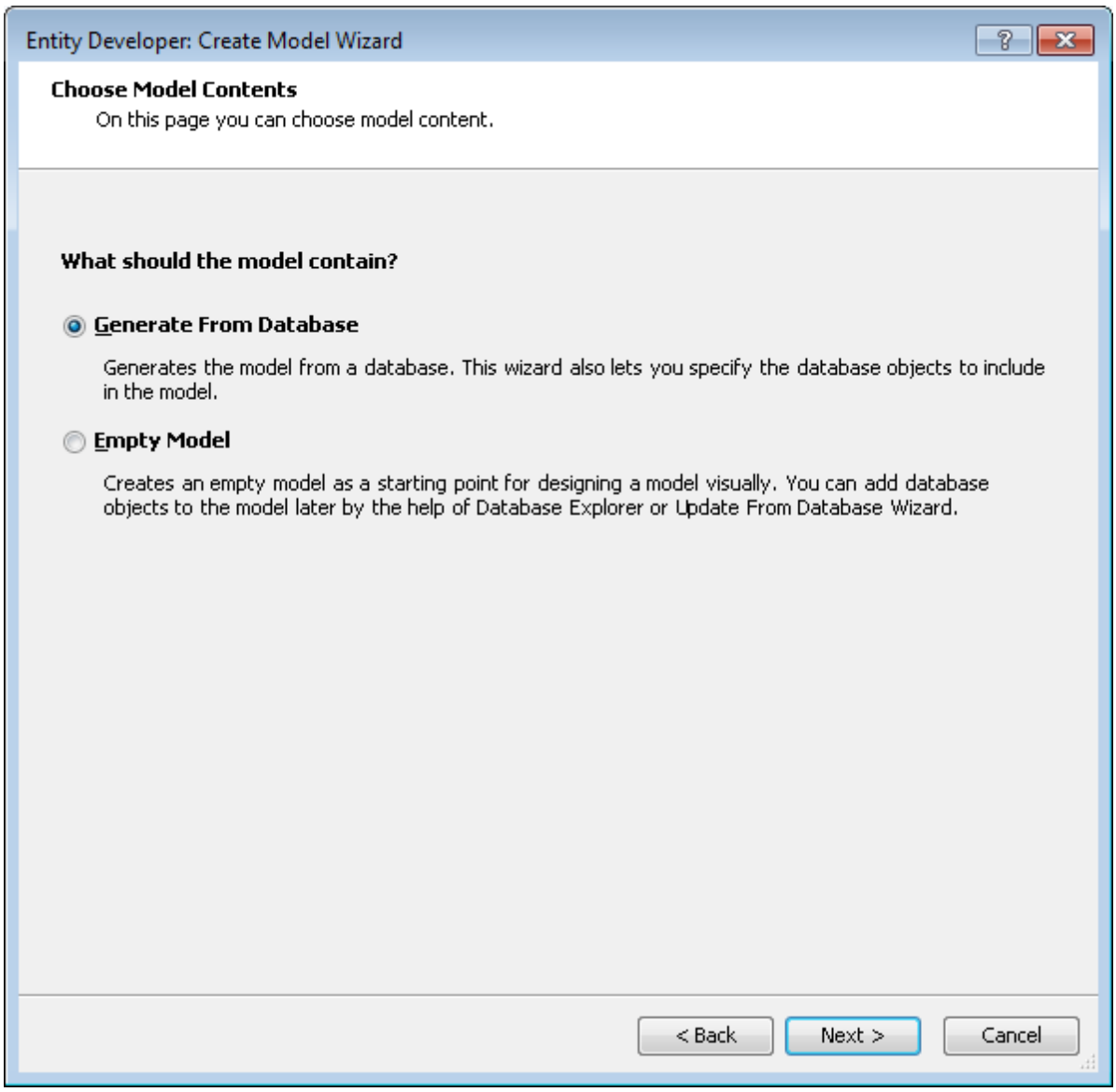
Step 3

Now, we have to set up the database connection. Select the "Devart dotConnect for SQLIte" provider and specify the path to the "demo.db3" file created on the "Prepare" stage.



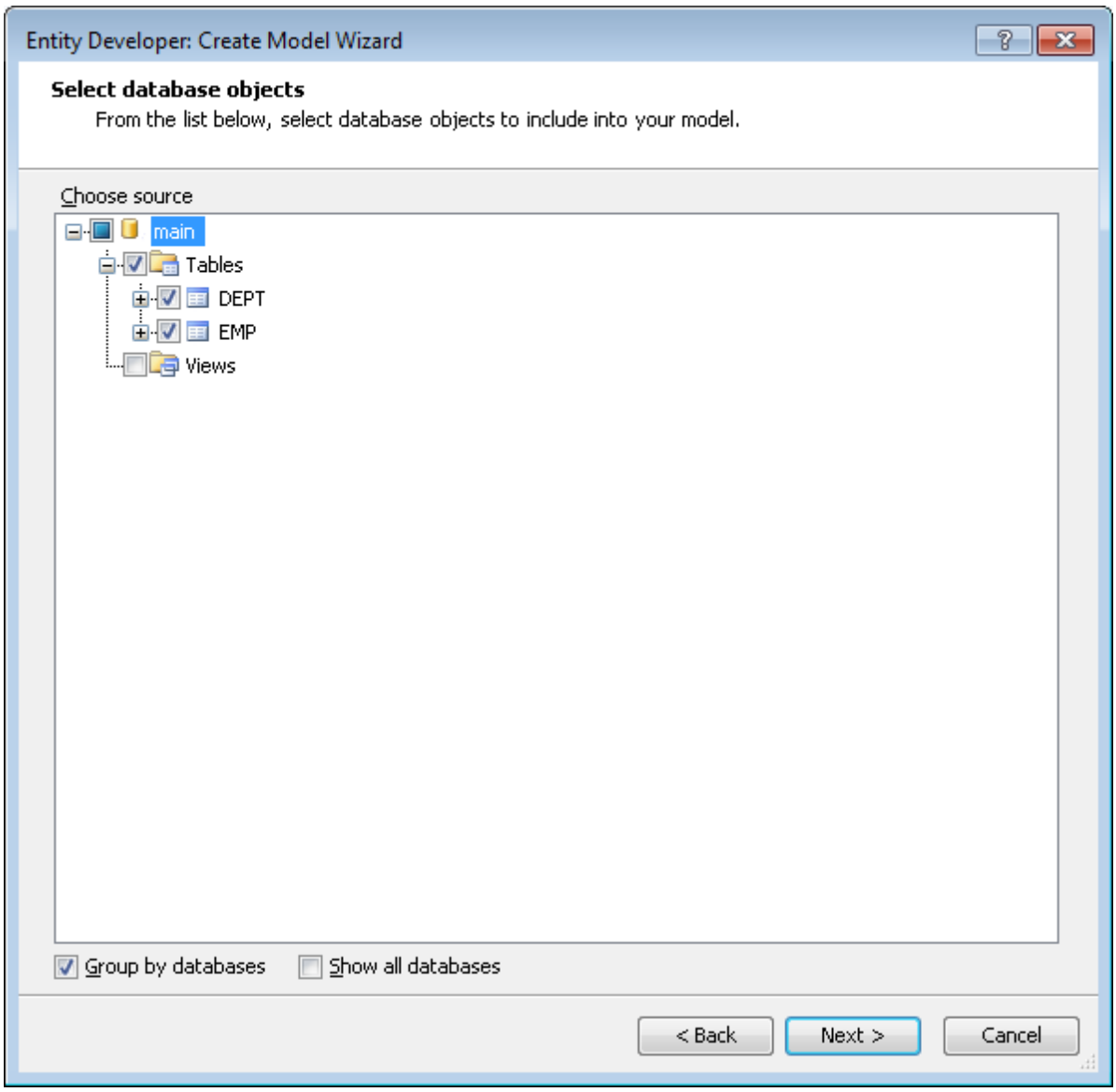
Step 4

In the next step, select the "Generate From Database" option, so we can add database objects to the model automatically instead of creating them manually in the model diagram view.



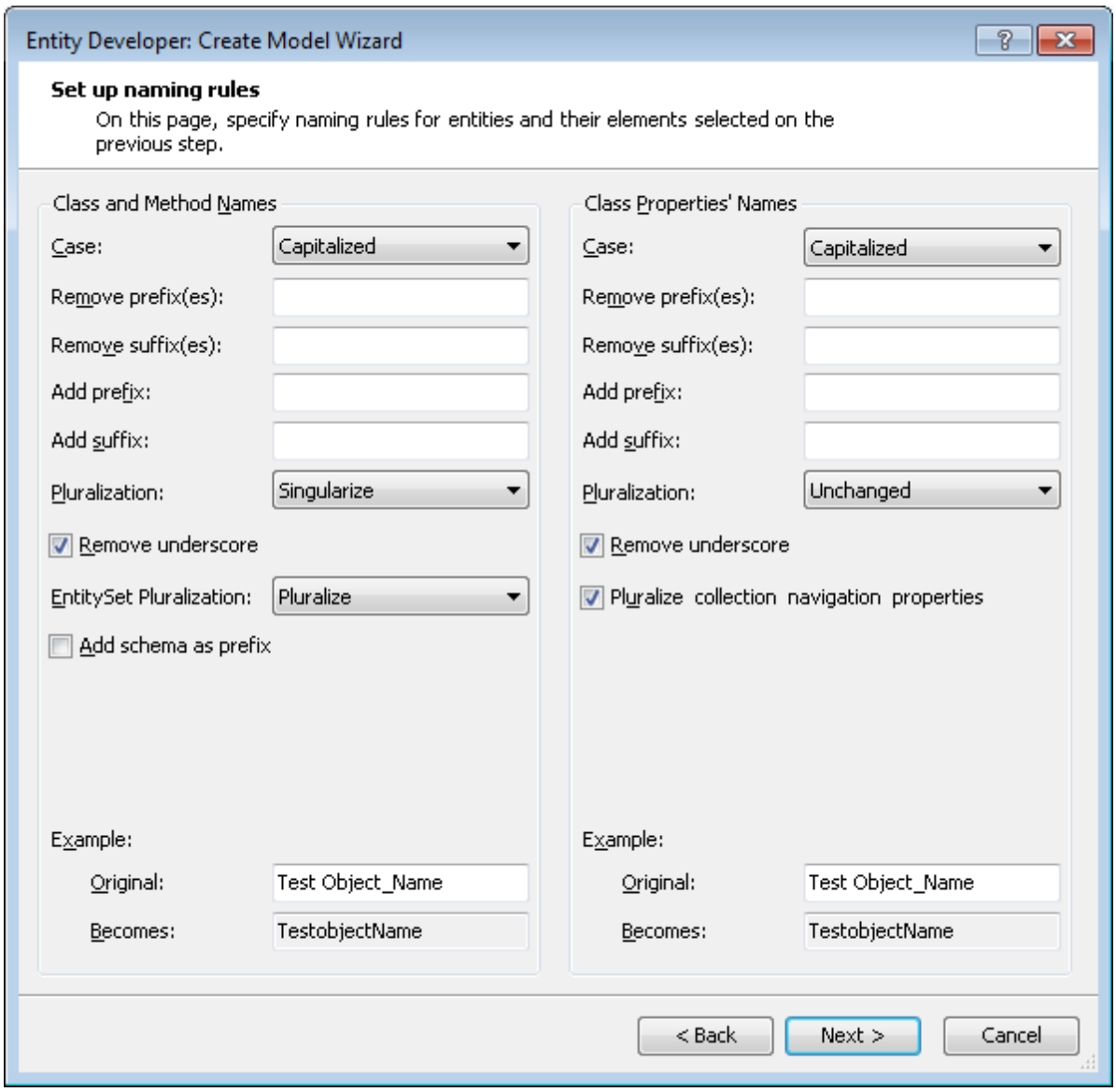
Step 5

After selecting this option, a list of database objects will be shown, from which we can choose the ones that we want to include to the model.



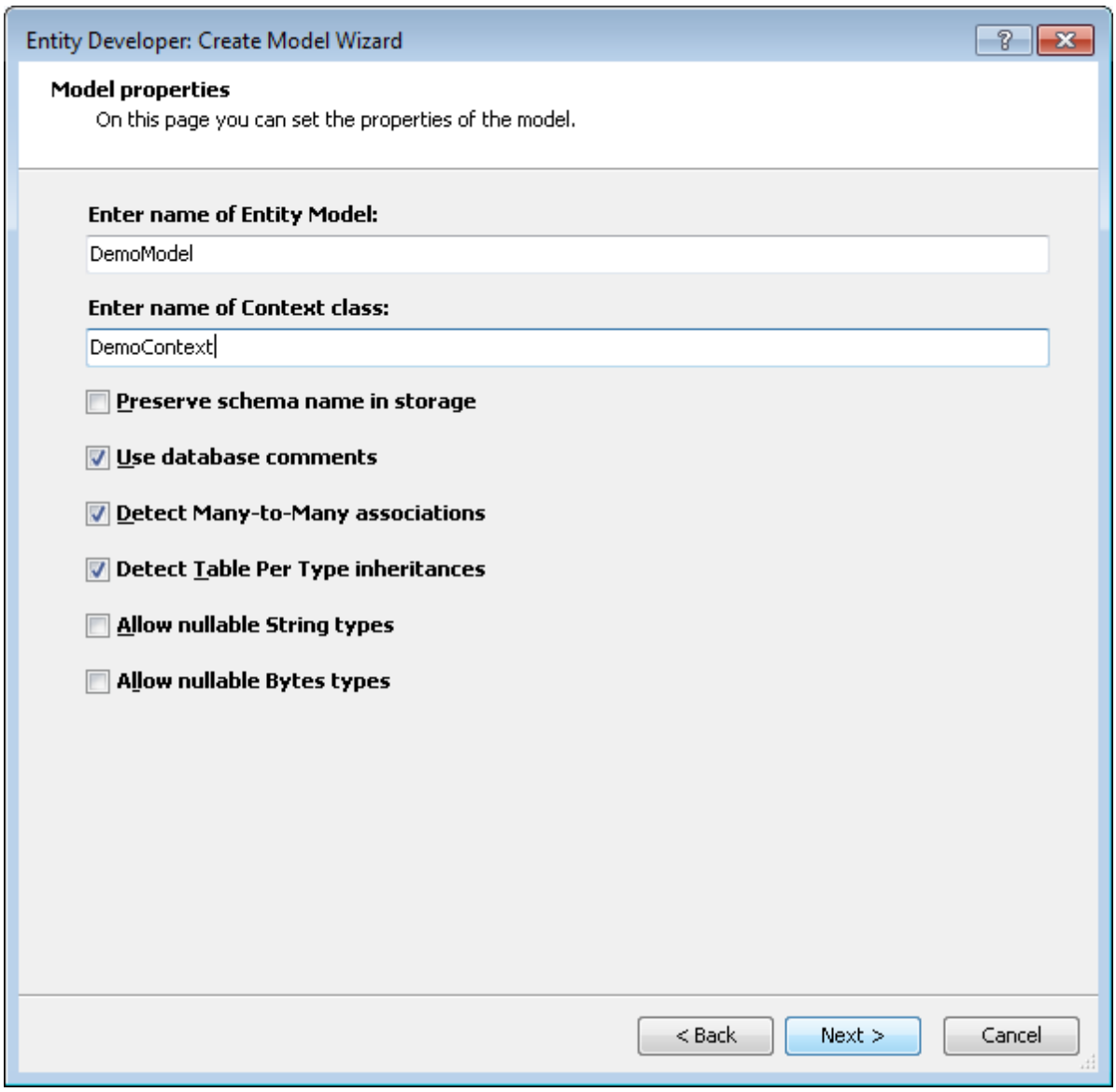
Step 6

When generating classes, Entity Developer automatically sets names for the classes and their properties depending on the database object names. The next wizard page allows to set up name generating rules. For now, leave the default option values unchanged.



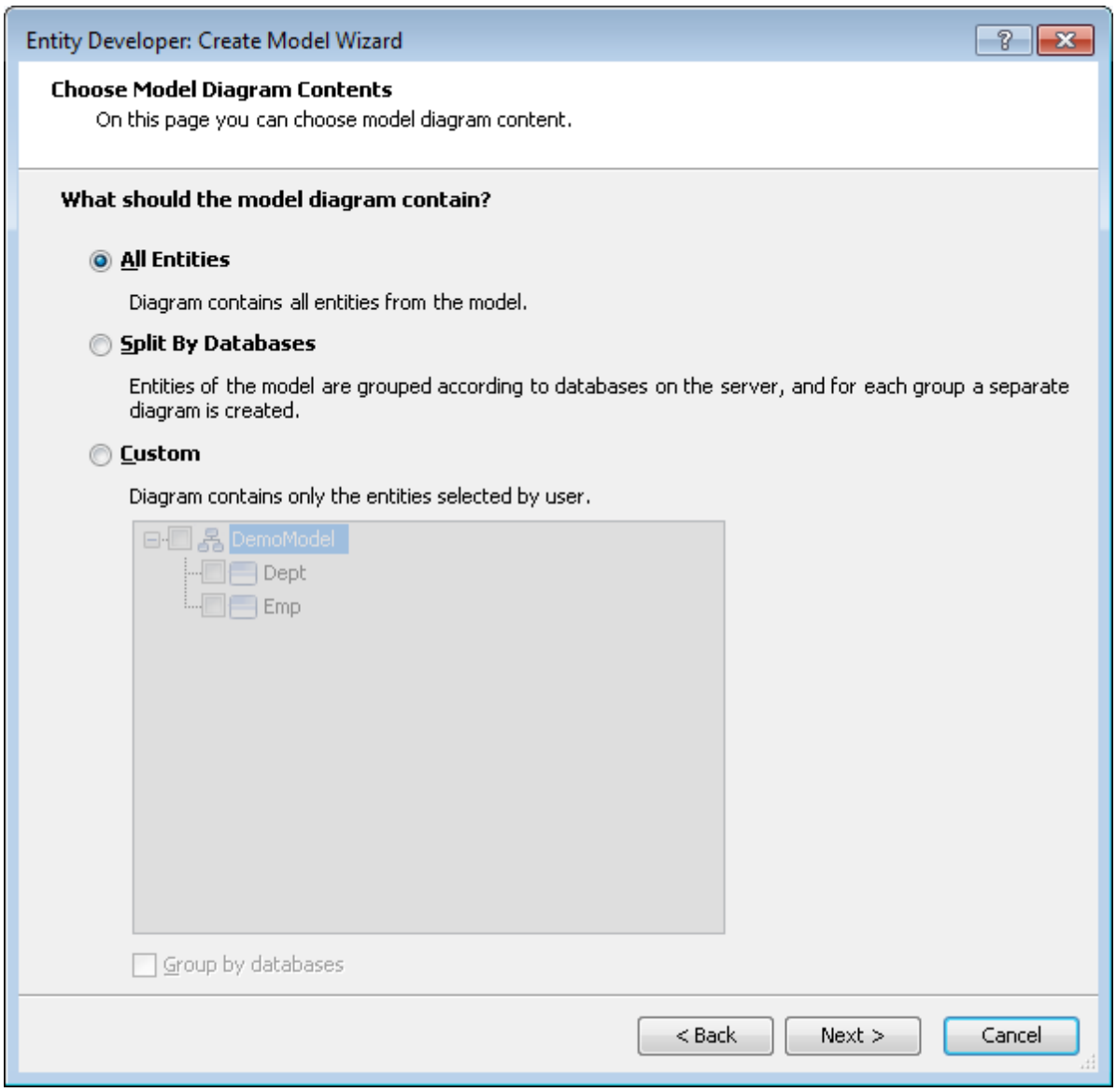
Step 7

Let's set names for our model and data context. At the next wizard page, set the model name to "DemoModel" and data context name to "DemoContext".



Step 8

Next page allows to choose the database objects that will be added to the model's diagram view. Leave the "All Entities" option selected.



Step 9

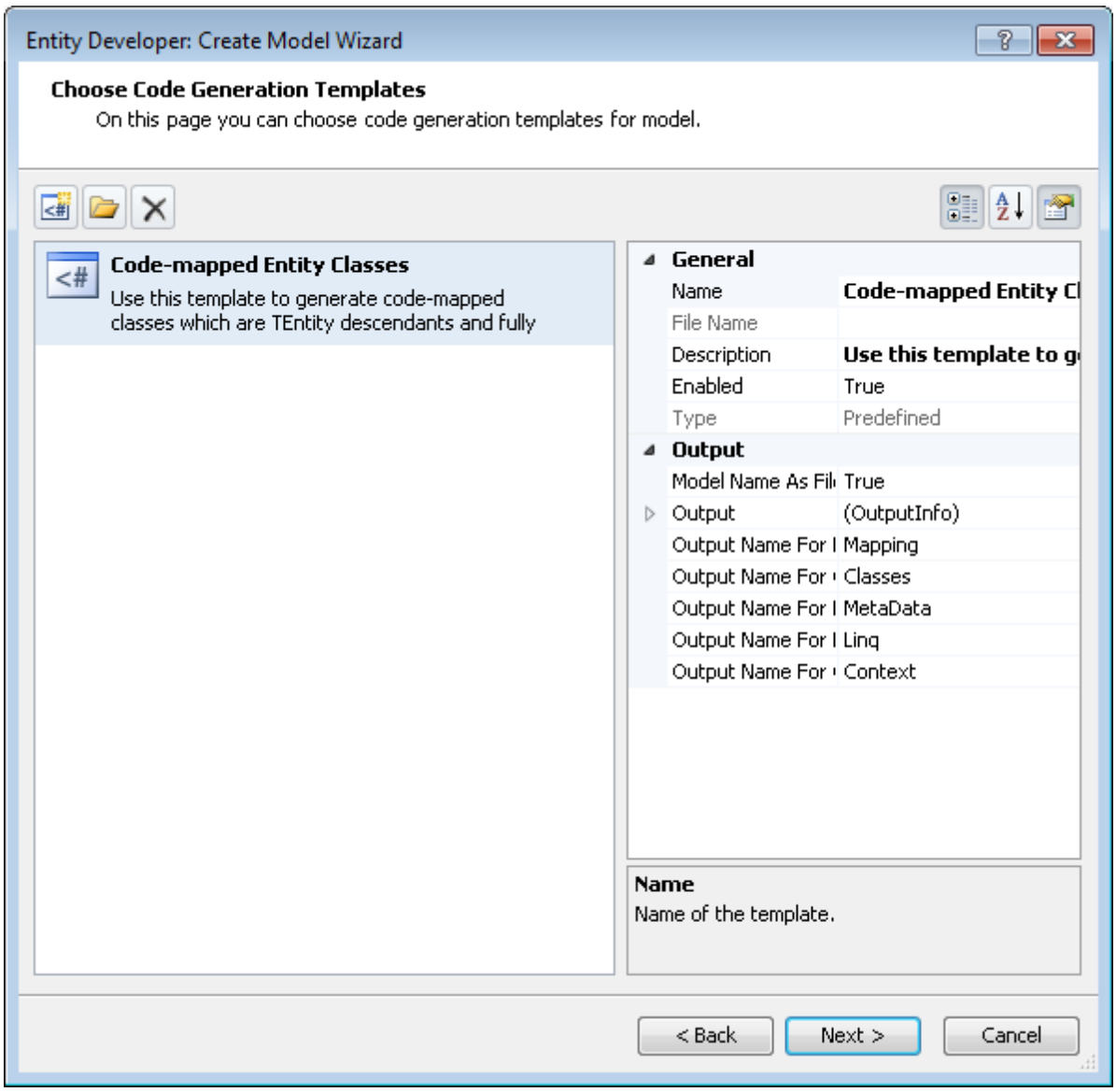
On the "Code Generation Templates" page we can set up the names of the generated files.

For the model type that we have chosen, several different units will be generated:

- a classes unit, which contains entity classes declarations;
- a metadata unit, which contains metadata classes declarations;
- a LINQ unit, which contains declarations of special utility classes, which will allow to write LINQ-style queries in the Delphi code;

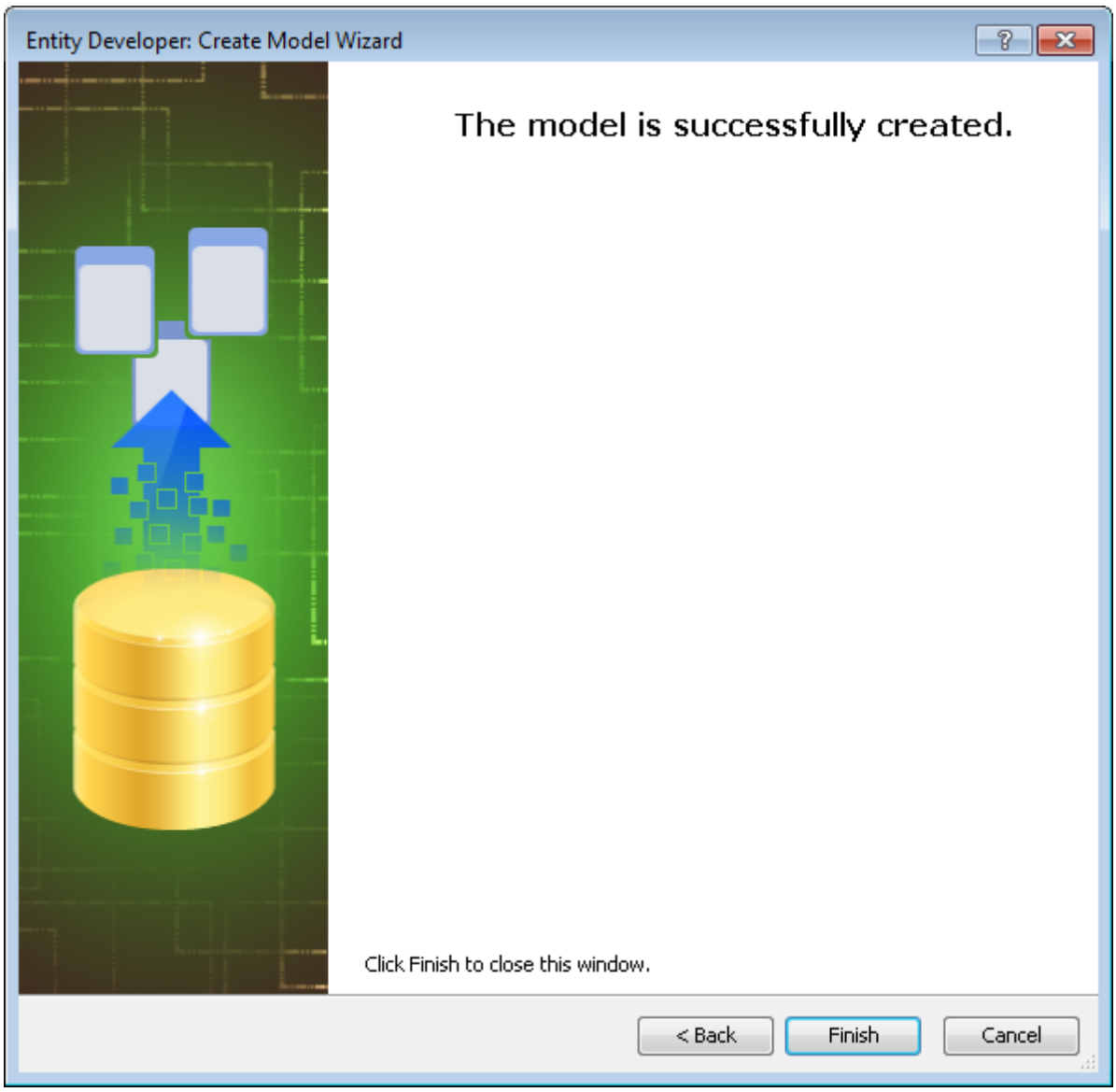
- a mapping file of the XML format which can be used to set up EntityDAC components in design-time using the TEntityXMLModel component;
- a data context unit which contains a declaration of the preconfigured data context component for the model being generated.

The name of each unit can be specified on the wizard page. By default, the names are: "Classes", "MetaData", "Linq", "Mapping" and "Context" respectively. Also, when the "Model Name As File Prefix" property is set to True, the model name, which we have set on the Step 7, will be added as a prefix to each file name. So, with the default settings, full file names will be: "DemoModel.Classes.pas", "DemoModel.MetaData.pas", "DemoModel.Linq.pas", "DemoModel.Mapping.pas" and "DemoModel.Context.pas".



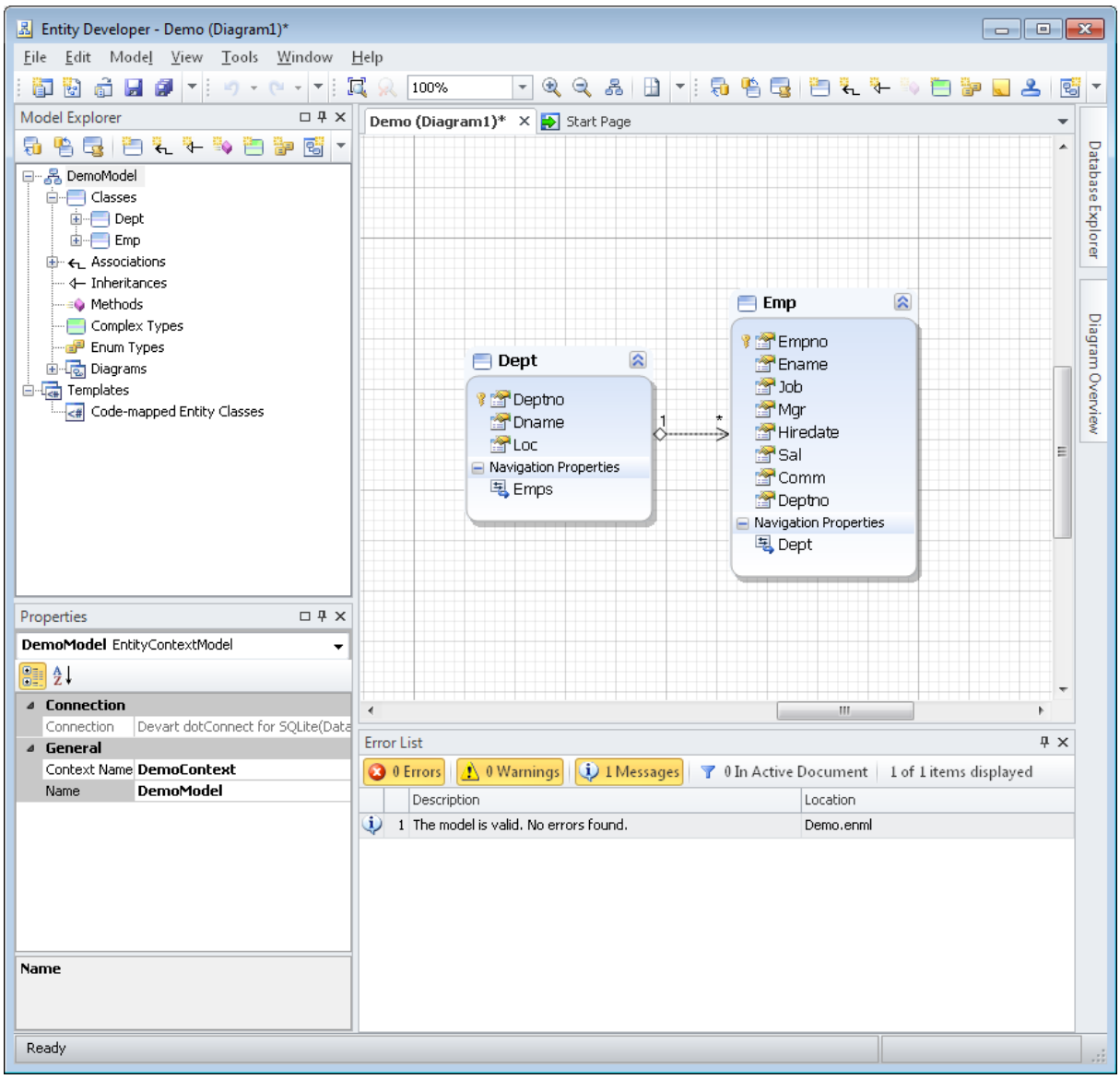
Step 10

Creating of our model is finished.

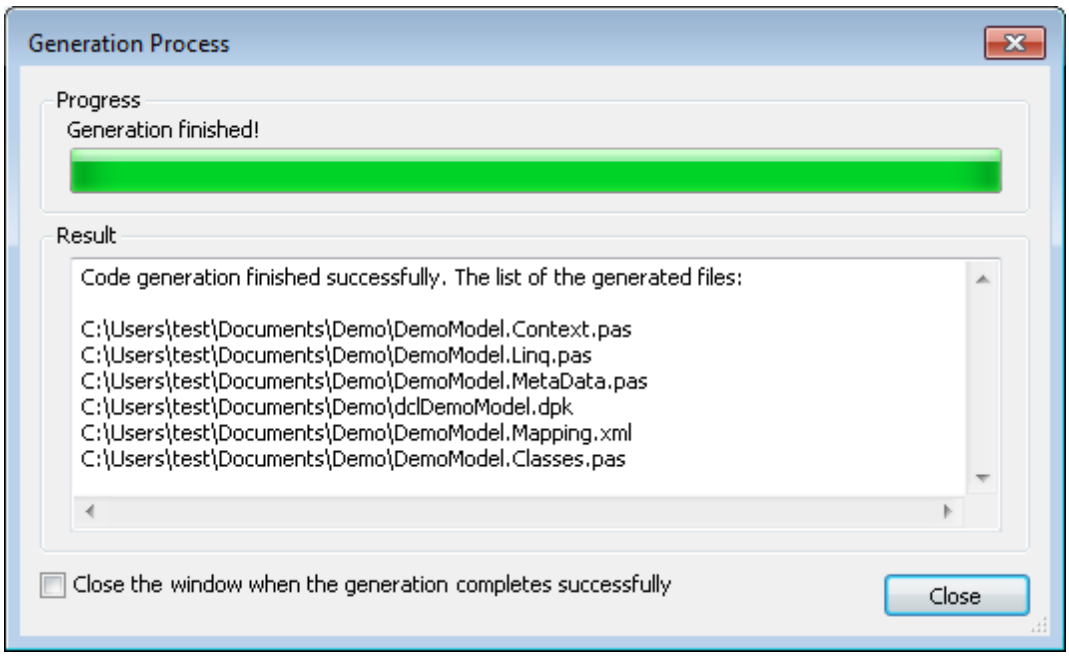


After finishing the wizard, the model will be created, and the model's diagram will be shown. About work with the model you can learn in the

Entity Developer documentation.



The final step in our tutorial is to generate files. To do this, press the "Generate Code" button (or the F7 key).



The generated files will be placed to the folder, in which the model project is saved. Now we are ready to move to the next stage - the creation of our test application.

Request Support DAC Forum Provide Feedback

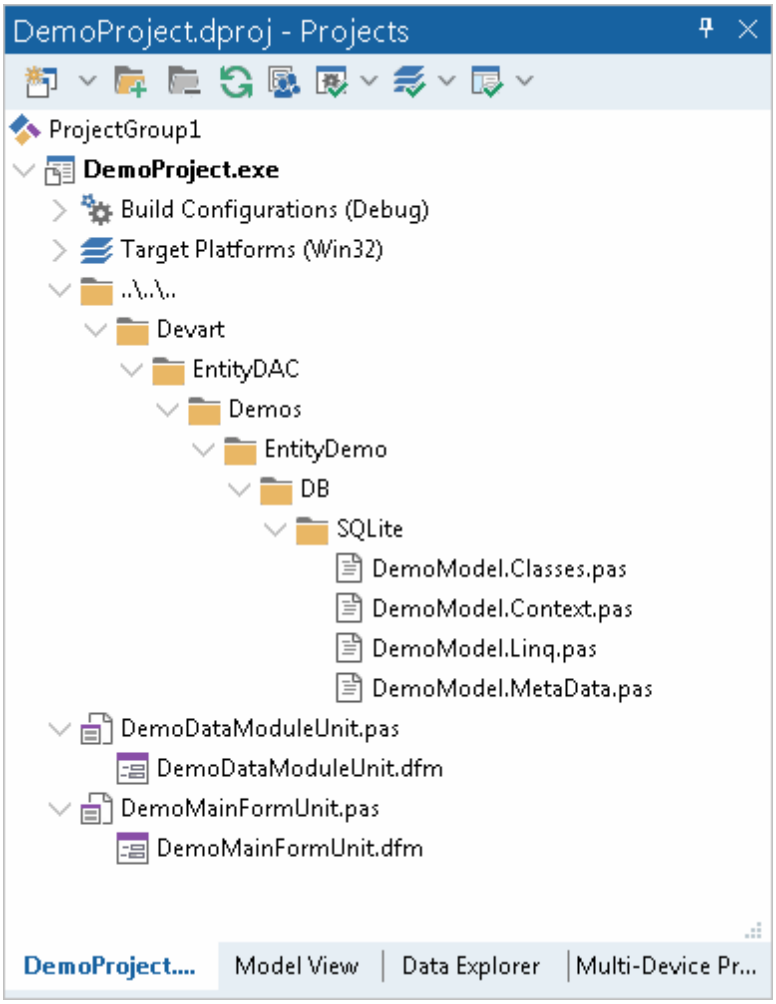
3.3.2 Creating New Application That Shows Entity List

In this part of the tutorial, we start to create our first application using EntityDAC.

Step 1

In RAD Studio, let's create a new VCL Forms Application project, and set its main form name to "DemoMainForm". Add a new data module to the application and name it "DemoDataModule". Then add to the project previously generated model units (DemoModel.Context.pas, DemoModel.Linq.pas, DemoModel.MetaData.pas, DemoModel.Classes.pas).

Save the project as "DemoProject", naming the main form and data module units as "DemoMainFormUnit" and "DemoDataModuleUnit" respectively.



Open the project source (select the "View Source" item in the project context menu; or select the project in the Project Manager and press Ctrl+V). In the project source, change the order of the project's form creation:

move the line

```
Application.CreateForm(TDemoDataModule, DemoDataModule);
```

above the line

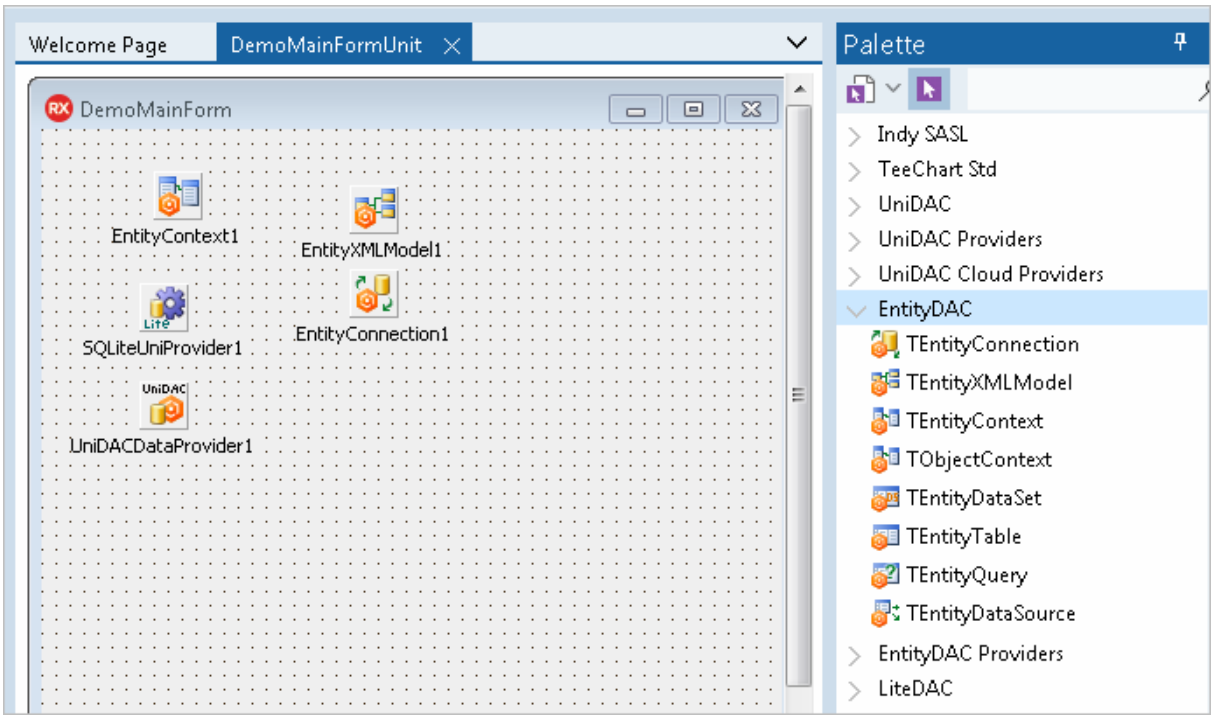
```
Application.CreateForm(TDemoMainForm, DemoMainForm);
```

Then, save the project.

Step 2

Open the data module in the Form Designer and place TEntityConnection, TEntityXMLModel

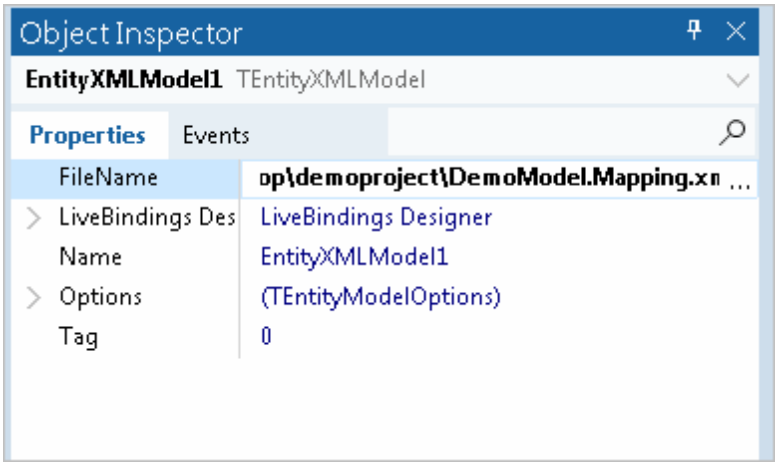
and TEntityContext components onto it from the "EntityDAC" component palette.



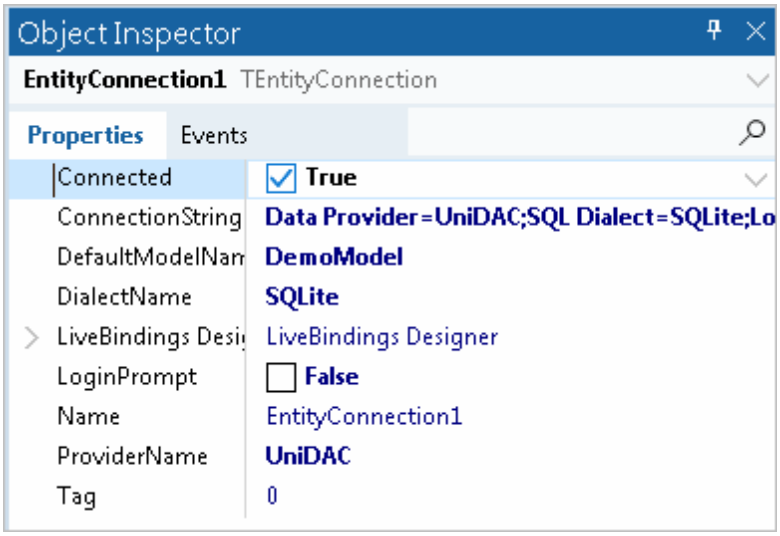
In addition, if you are using UniDAC, place TUniDACDataProvider from the "EntityDAC Providers" component palette and TSQLiteUniProvider from the "UniDAC Providers" component palette. Or, if you are using LiteDAC - place the TLitedacDataProvider component from the "EntityDAC Providers" component palette.

Step 3

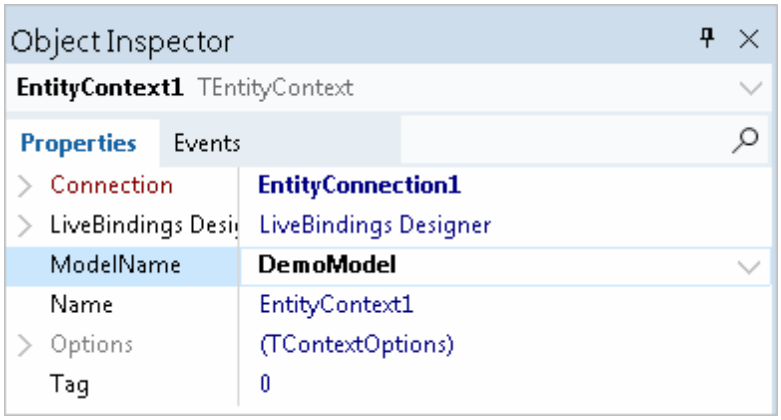
In order to have access to model's metadata in design-time, set the FileName property of the TEntityXMLModel component to the *.Mapping.xml file.



Then select the EntityConnection component, specify the path to the database in the ConnectionString property and set other properties as follows:



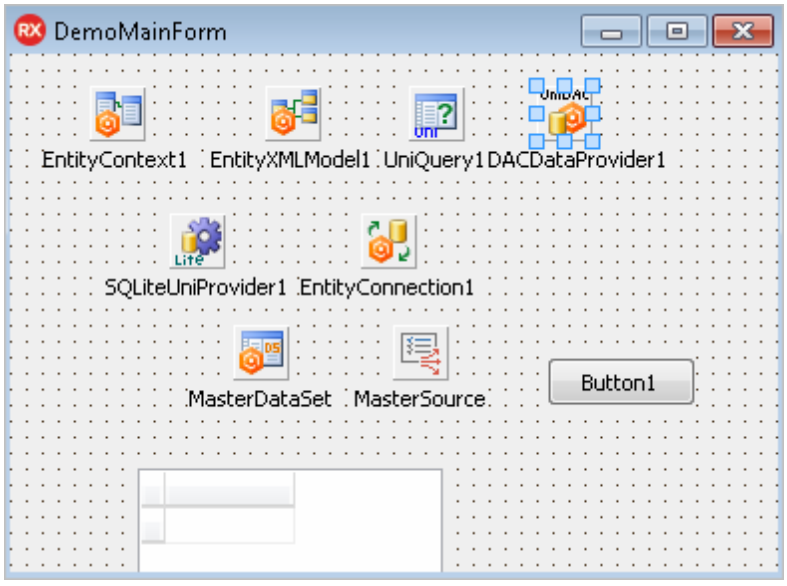
Finally, set up the data context properties.



Step 4

Let's set up the main form in order to display a list of database entities.

Add the DemoDataModuleUnit into the USES clause of the main form unit. Then open the main form in the form designer and place TEntityDataSet, TDataSource and TDBGrid components onto it. Set the component names as it is shown on the picture. Set the DataSource property of the TDBGrid component to "MasterSource", and the DataSet property of the TDataSource component - to "MasterDataSet". Also, place TButton on the form and set its name to "Button1".



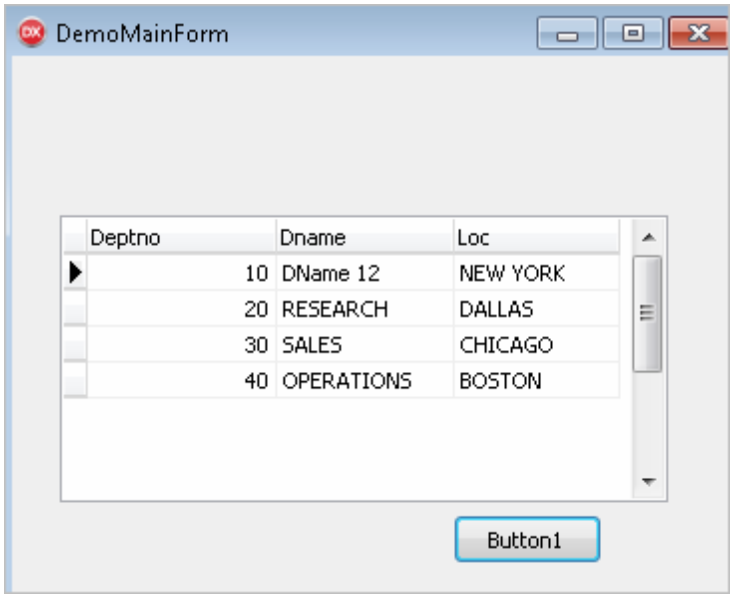
TEntityDataSet is designed to store an arbitrary list of entities, thus, it can be set up in run-time only. Write the code shown below in the Button1.OnClick event handler.

```
implementation
{$R *.dfm}
uses
  EntityDAC.Enumerable,
  EntityDAC.Linq;
procedure TDemoMainForm.Button1Click(Sender: TObject);
var
  Depts: ILinqQueryable;
begin
  Depts := Linq.From(DemoDataModule.EntityContext1['Dept']).Select;
  MasterDataSet.SourceCollection := DemoDataModule.EntityContext1.GetEntitie
  MasterDataSet.Open;
end;
```

In order to declare the Depts variable of type ILinqQueryable, add the EntityDAC.Enumerable and EntityDAC.Linq units into the USES clause.

In the code we initialize the Depts variable with the result of the LINQ query that returns the list of all entities of metatype 'Dept', then set Depts as the source collection for MasterDataSet.

Now we can make sure that everything is done correctly. Compile and run the project, and press the "Open" button.



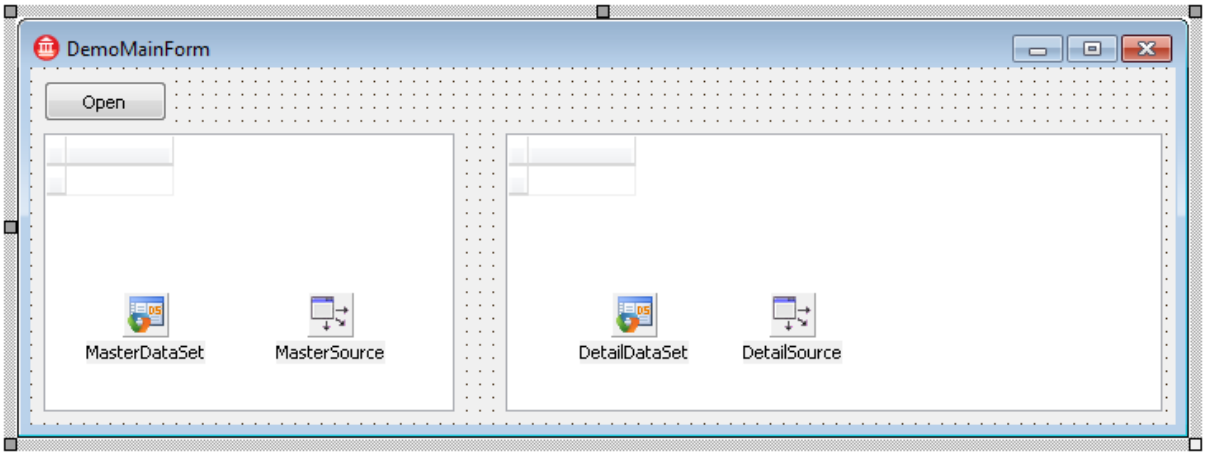
Request Support DAC Forum Provide Feedback

3.3.3 Extend the Application to Show Master-Detail Datasets

In this part of the tutorial we continue to develop our application.

Step 1

First, place another TDBGrid on the main form. Then, place TEntityDataSet component from the "EntityDAC" component palette, and the TDataSource component. Name the components as shown below. Set the DataSource property of TDBGrid to "DetailSource", and the DetailSource.DataSet property to "DetailDataSet".



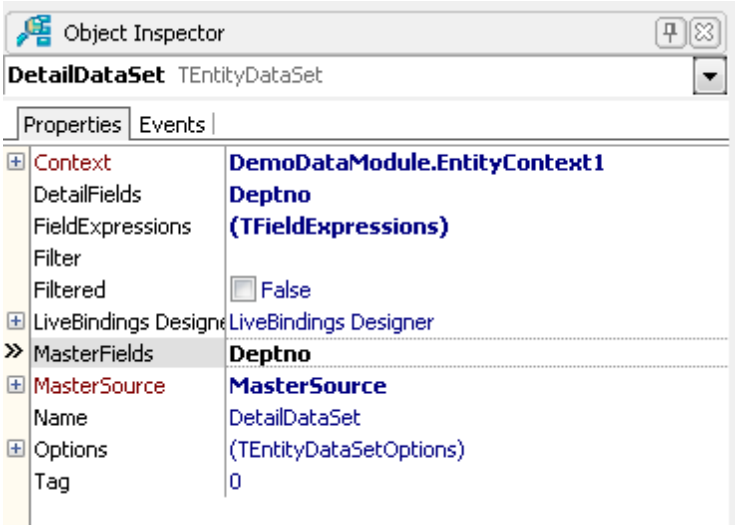
Step 2

Extend the Button1.OnClick event handler as shown below:

```
procedure TDemoMainForm.Button1Click(Sender: TObject);
var
  Depts: ILinqQueryable;
  Emps: ILinqQueryable;
begin
  Depts := Linq.From(DemoDataModule.EntityContext1['Dept']).Select;
  MasterDataSet.SourceCollection := DemoDataModule.EntityContext1.GetEntitie
  Emps := Linq.From(DemoDataModule.EntityContext1['Emp']).Select;
  DetailDataSet.SourceCollection := DemoDataModule.EntityContext1.GetEntitie
  DetailDataSet.Open;
  MasterDataSet.Open;
end;
```

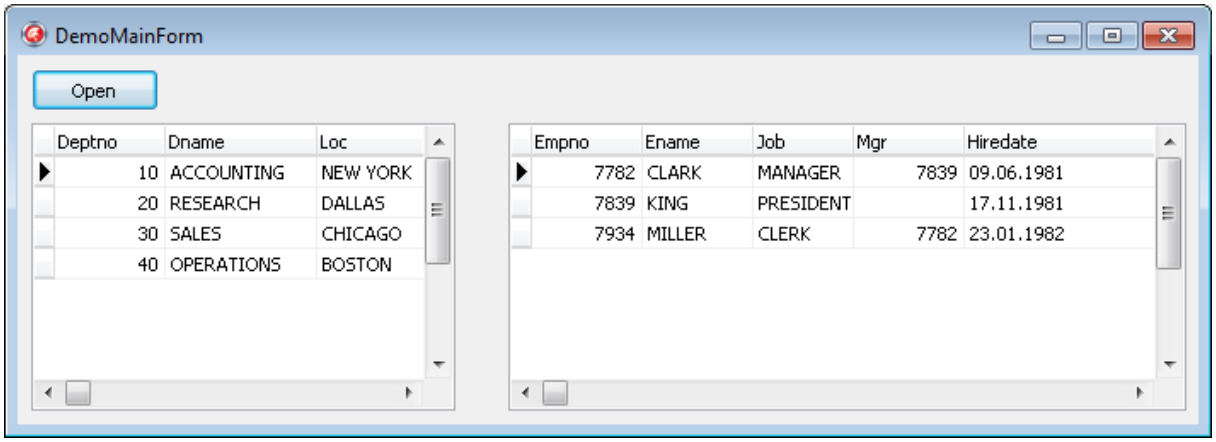
Step 3

Finally, set up the master-detail relationship between MasterDataSet and DetailDataSet.



Step 4

Now we are ready to test our application. Compile and run the project, and press the "Open" button.



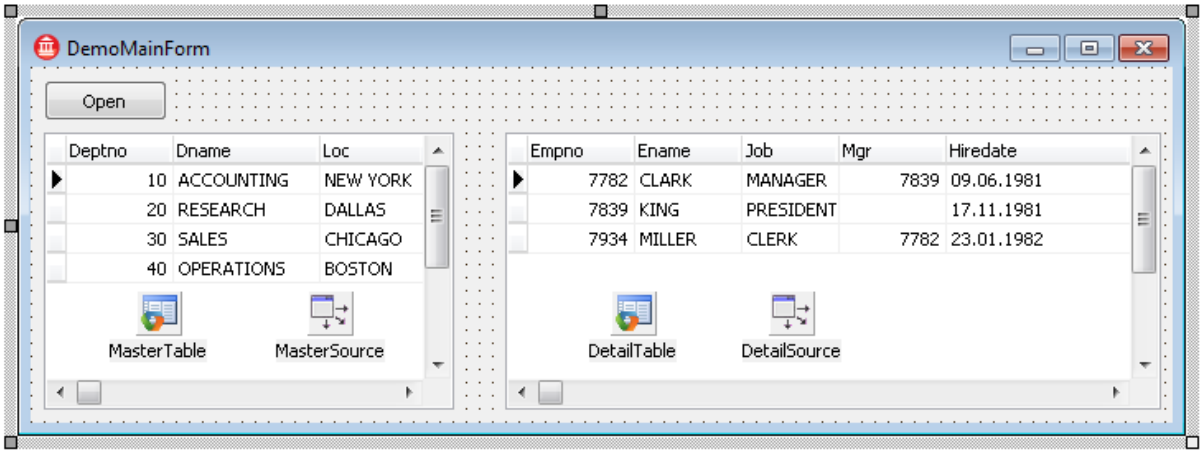
Request Support DAC Forum Provide Feedback

3.3.4 Using Professional Edition

EntityDAC Professional edition includes the most complete set of components that allows to solve the considered task in even more simple way.

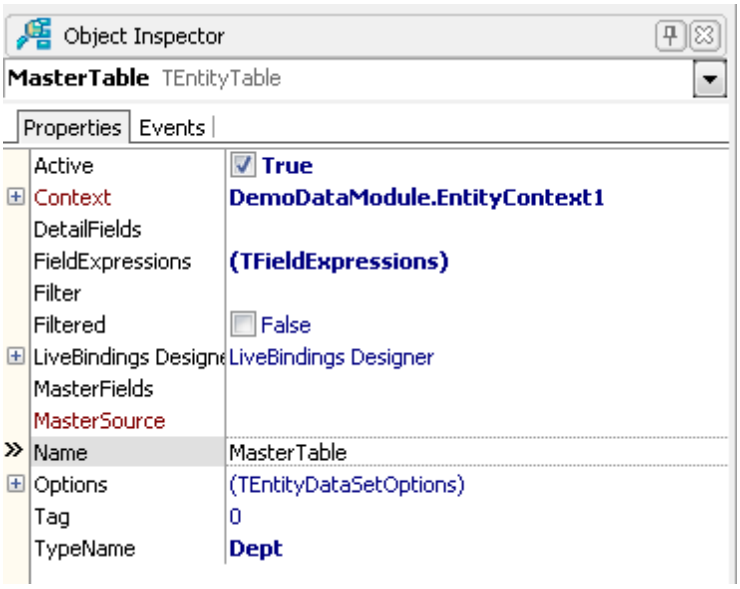
Step 1

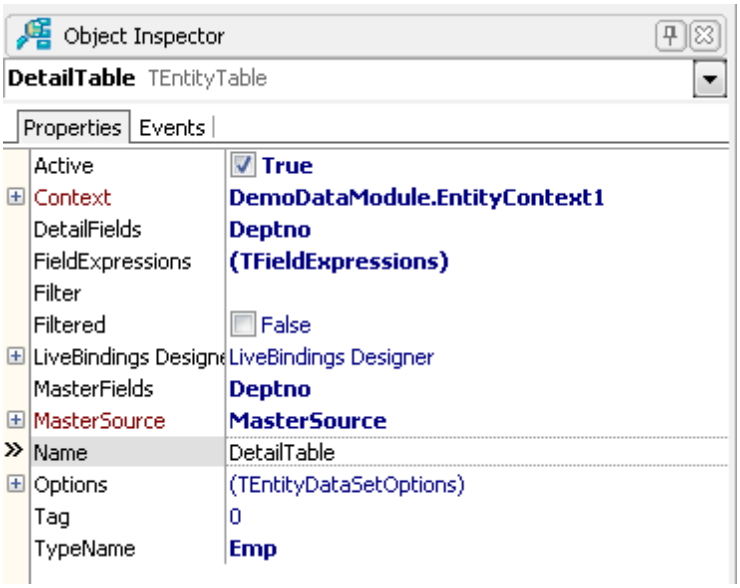
Open the main form in the form designer and delete MasterDataSet and DetailDataSet. Instead, put two TEntityTable components on the form, naming them MasterTable and DetailTable correspondingly.



Step 2

Link MasterDataSource to MasterTable and DetailDataSource to DetailTable. Then set the MasterTable.TypeName to Dept and DetailTable.TypeName to Emp. Also, set up the master-detail relationship between MasterTable and DetailTable.



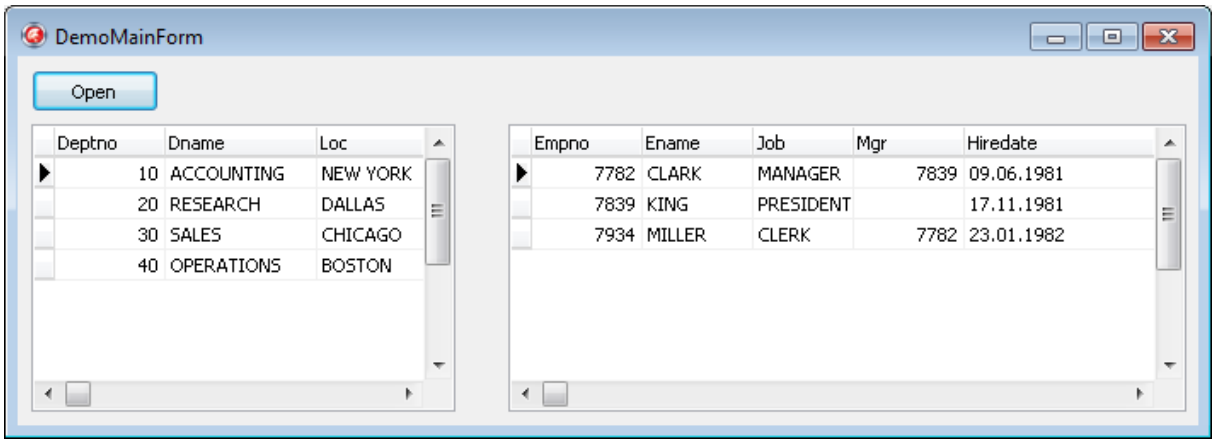


Step 3

Modify the Button1.OnClick event handler. Remove all inner code except opening datasets.

```
procedure TDemoMainForm.Button1Click(Sender: TObject);
begin
  DetailTable.Open;
  MasterTable.Open;
end;
```

Compile and run the project, and press the “Open" button.



Request Support DAC Forum Provide Feedback

3.4 Creating My First Application With Express Edition

In this tutorial we will create our first application using EntityDAC Express Edition.

Objective

Create an application with two lists. The first one displays a list of an enterprise departments. The second one displays a list of the selected department employees. The application data is stored in a SQLite database, and Devart Universal Data Access Components is used as a data-access layer.

Step 1. Create a database

First we have to create a database for our application. Lets create a new SQLite database named "demo.db3" using any of the SQLite management tools.

Then we have to create tables necessary for the application. We need two tables – DEPT and EMP – with the following structure:

```
CREATE TABLE DEPT (
  DEPTNO INTEGER PRIMARY KEY,
  DNAME VARCHAR(14) NOT NULL,
  LOC VARCHAR(13) NOT NULL
);
CREATE TABLE EMP (
  EMPNO INTEGER PRIMARY KEY,
  ENAME VARCHAR(10) NOT NULL,
  JOB VARCHAR(9) NOT NULL,
  MGR INTEGER,
  HIREDATE TIMESTAMP NOT NULL,
  SAL REAL,
  COMM REAL,
  DEPTNO INT REFERENCES DEPT
);
```

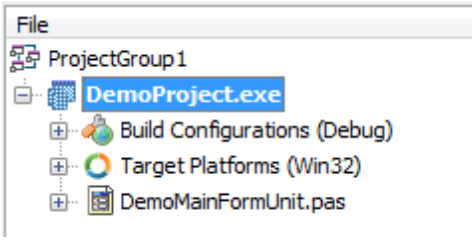
Finally, let's fill test tables with data:

```
INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT VALUES (30,'SALES','CHICAGO');
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');
INSERT INTO EMP VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20);
INSERT INTO EMP VALUES (7499,'ALLEN','SALESMAN',7698,'1981-2-20',1600,300,30
INSERT INTO EMP VALUES (7521,'WARD','SALESMAN',7698,'1981-2-22',1250,500,30)
INSERT INTO EMP VALUES (7566,'JONES','MANAGER',7839,'1981-4-2',2975,NULL,20)
INSERT INTO EMP VALUES (7654,'MARTIN','SALESMAN',7698,'1981-9-28',1250,1400,
INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER',7839,'1981-5-1',2850,NULL,30)
INSERT INTO EMP VALUES (7782,'CLARK','MANAGER',7839,'1981-6-9',2450,NULL,10)
INSERT INTO EMP VALUES (7788,'SCOTT','ANALYST',7566,'1987-7-13',3000,NULL,20
INSERT INTO EMP VALUES (7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,
```

```
INSERT INTO EMP VALUES (7844,'TURNER','SALESMAN',7698,'1981-9-8',1500,0,30);
INSERT INTO EMP VALUES (7876,'ADAMS','CLERK',7788,'1987-7-13',1100,NULL,20);
INSERT INTO EMP VALUES (7900,'JAMES','CLERK',7698,'1981-12-3',950,NULL,30);
INSERT INTO EMP VALUES (7902,'FORD','ANALYST',7566,'1981-12-3',3000,NULL,20)
INSERT INTO EMP VALUES (7934,'MILLER','CLERK',7782,'1982-1-23',1300,NULL,10)
```

Step 2. Create the application

In RAD Studio, create a new VCL Forms Application project, and set its main form name to *DemoMainForm*. Save the project with the name *"DemoProject.dproj"*, naming the main form unit as *"DemoMainFormUnit.pas"*.

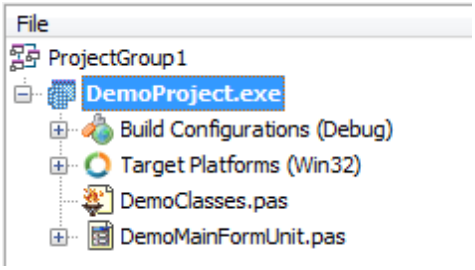


Step 3. Declare entity classes

EntityDAC Express Edition does not include Entity Developer, therefore we have to manually declare entity classes needed for the application. Since our database has two tables, we need to declare two corresponding entity classes.

Now, let's choose the mapping type we will use. EntityDAC provides four different mapping types which are described in the "Model Mapping" article. Every mapping type has its benefits and disadvantages. Our application will use the mapping type known as "attribute-mapped entities", because this mapping type is more easy to implement than "code-mapped entities", but at the same time it provides all advantages of using *TEntity* as a base class for entities.

Let's add a new unit to our project and save it as *"DemoClasses.pas"*.



In the unit, declare TDept and TEmp classes corresponding to the tables structure.

```
  [Table('DEPT')]
  [Model('Demo')]
  [Key('FDeptno')]
  TDept = class(TMappedEntity)
  private
    [Column('DEPTNO', [ReadOnly])]
    FDeptno: Integer;
    [Column('DNAME', 14)]
    FDname: String;
    [Column('LOC', 13)]
    FLoc: String;
    [Column]
    [Collection('TEmp', 'FDept', 'FDeptno', 'FDeptno', srNone, drNone)]
    FEmps: TDeptEmps;
    function GetDeptno: Integer;
    function GetDname: String;
    procedure SetDname(const Value: String);
    function GetLoc: String;
    procedure SetLoc(const Value: String);
  protected
    constructor Create(AMetaType: TMetaType); overload; override;
  public
    constructor Create; overload; override;
    property Deptno: Integer read GetDeptno;
    property Dname: String read GetDname write SetDname;
    property Loc: String read GetLoc write SetLoc;
    property Emps: TDeptEmps read FEmps;
  end;
  [Table('EMP')]
  [Model('Demo')]
  [Key('FEmpno')]
  TEmp = class(TMappedEntity)
  private
    [Column('EMPNO', [ReadOnly])]
    FEmpno: Integer;
    [Column('ENAME', 10)]
    FEname: String;
    [Column('JOB', 9)]
    FJob: String;
    [Column('MGR', [CanBeNull])]
    FMgr: IntegerNullable;
    [Column('HIREDATE')]
    FHiredate: DateTime;
    [Column('SAL', [CanBeNull])]
    FSal: DoubleNullable;
    [Column('COMM', [CanBeNull])]
    FComm: DoubleNullable;
    [Column('DEPTNO', [CanBeNull])]
    FDeptno: IntegerNullable;
    [Column]
    [Reference('TDept', 'FEmps', 'FDeptno', 'FDeptno', srNone, drNone)]
    FDept: TMappedReference;
    function GetEmpno: Integer;
    function GetEname: String;
    procedure SetEname(const Value: String);
    function GetJob: String;
```

```
    procedure SetJob(const Value: String);
    function GetMgr: IntegerNullable;
    procedure SetMgr(const Value: IntegerNullable);
    function GetHiredate: TDateTime;
    procedure SetHiredate(const Value: TDateTime);
    function GetSal: DoubleNullable;
    procedure SetSal(const Value: DoubleNullable);
    function GetComm: DoubleNullable;
    procedure SetComm(const Value: DoubleNullable);
    function GetDeptno: IntegerNullable;
    procedure SetDeptno(const Value: IntegerNullable);
    function GetDept: TDept;
    procedure SetDept(const Value: TDept);
  protected
    constructor Create(AMetaType: TMetaType); overload; override;
  public
    constructor Create; overload; override;
    property Empno: Integer read GetEmpno;
    property Ename: String read GetEname write SetEname;
    property Job: String read GetJob write SetJob;
    property Mgr: IntegerNullable read GetMgr write SetMgr;
    property Hiredate: TDateTime read GetHiredate write SetHiredate;
    property Sal: DoubleNullable read GetSal write SetSal;
    property Comm: DoubleNullable read GetComm write SetComm;
    property Deptno: IntegerNullable read GetDeptno write SetDeptno;
    property Dept: TDept read GetDept write SetDept;
  end;
```

The complete unit source is available in the "DemoClasses.pas" appendix. Entity class mapping is defined using special mapping attributes. Detailed information about mapping attributes you can found in the "Attribute-mapped entities" article.

Step 4. Connect to the database

Now we are ready to establish the database connection in our project. As it specified in the objective, we use Devart UniDAC as a data-access layer, so make sure that you have UniDAC installed.

Since EntityDAC Express Edition does not provide any design-time components, we have to implement all needed functionality in the code. First, add several units to the main form USES clause:

```
uses
  EntityDAC.EntityConnection,
  EntityDAC.DataProvider.UniDAC,
  SQLiteUniProvider;
```

We need the *EntityDAC.EntityConnection* unit in order to use the *TEntityConnection* class. The *EntityDAC.DataProvider.UniDAC* unit provides intermediate layer between EntityDAC and UniDAC. The *SQLiteUniProvider* unit is a part of UniDAC and implements access to a

SQLite database.

In the main form declaration, declare fields and methods needed for establishing a connection:

```
type
  TDemoMainForm = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    FConnection: TEntityConnection;
  public
    procedure Connect;
  end;
```

And write the methods implementation:

```
procedure TDemoMainForm.FormCreate(Sender: TObject);
begin
  FConnection := TEntityConnection.Create;
end;
procedure TDemoMainForm.FormDestroy(Sender: TObject);
begin
  FConnection.Free;
end;
procedure TDemoMainForm.FormShow(Sender: TObject);
begin
  Connect;
end;
procedure TDemoMainForm.Connect;
begin
  FConnection.ProviderName := 'UniDAC';
  FConnection.DialectName := 'SQLite';
  FConnection.ConnectionString := 'ProviderName=SQLite;Direct=True;DataBase=
  FConnection.Connect;
end;
```

As you can see from the Connect method implementation, the project database file - *"demo.db3"* - has to be previously placed in the project output folder (by default, <Project path>\Debug\Win32).

Now, we are ready to test the database connection. Compile and run the application. If no error messages appeared and the main application form is shown, then the connection is established successfully.

Note: if you are using not a UniDAC data provider or another database server, then replace the following code lines with yours:

```
uses
 ...
  EntityDAC.DataProvider.xxxDAC, //used provider name
```

```
 ...
```

```
...
  FConnection.ProviderName := '...'; // used provider name
  FConnection.DialectName := '...';  // used SQL dialect
  FConnection.ConnectionString := '...'; // connection string for used data
  FConnection.Connect;
...
```

Step 5. Prepare to obtain data from the database

In order to obtain any data from the database, we have to use the *TEntityContext* class that is designed for data manipulation in EntityDAC. Include the *EntityDAC.EntityContext* unit to the form USES clause, add the following field to the form's private declaration section:

```
    FContext: TentityContext;
```

and then expand the form OnCreate and OnDestroy methods to implement the entity context creation and destruction:

```
procedure TDemoMainForm.FormCreate(Sender: TObject);
begin
  FConnection := TEntityConnection.Create;
  FContext := TEntityContext.Create;
  FContext.Connection := FConnection;
  FContext.ModelName := 'Demo';
end;
procedure TDemoMainForm.FormDestroy(Sender: TObject);
begin
  FConnection.Free;
  FContext.Free;
end;
```

In the *OnCreate* method we create the entity context instance, associate it with the connection and then set the meta model name to the name defined in the entity classes declaration made on Step 3.

Step 6. Populate the departments list

EntityDAC Express Edition does not provide any dataset component, therefore we will obtain the departments list using a special *IEntityEnumerable* interface and then visualize the list using the default *TListBox* component.

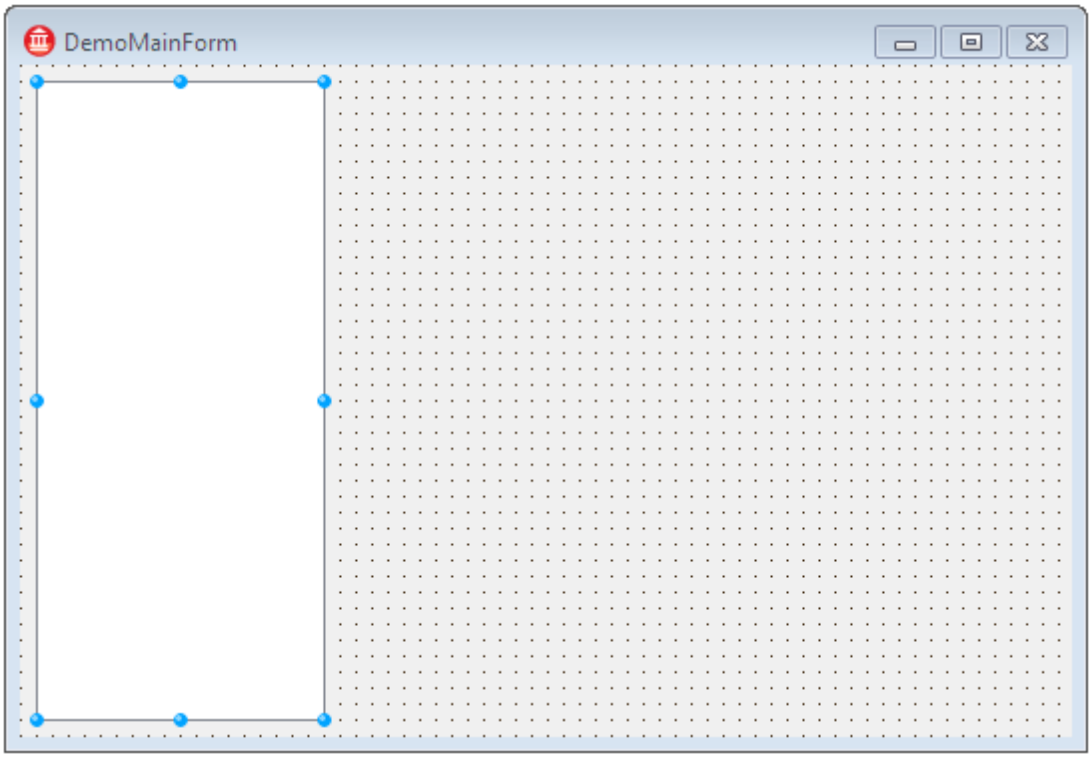
Declare a form field for storing the departments collection in the form's *private* declaration section:

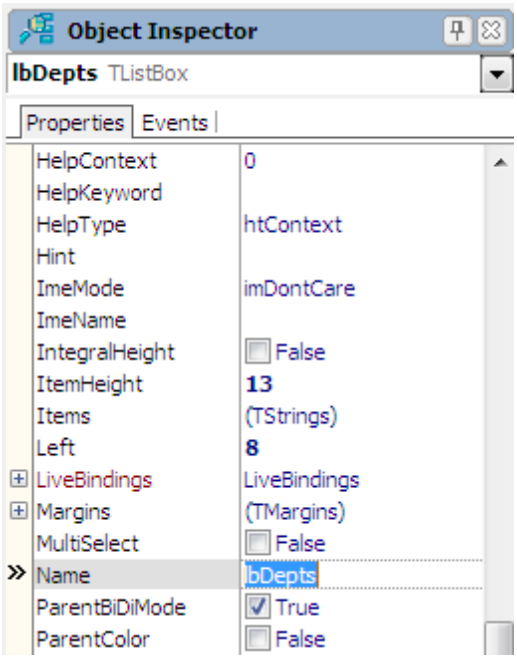
```
    FDepts: IentityEnumerable<TDept>;
```

Since the *FDepts* collection is of interface type, we dont need to care about its destruction. In

order to use the *IEntityEnumerable* interface, add the *EntityDAC.Entity* unit to the form USES clause. Also, add the *DemoClasses* unit to obtain access to entity classes.

In the form designer, drop the *TListBox* component on the form and name it *lbDepts*:



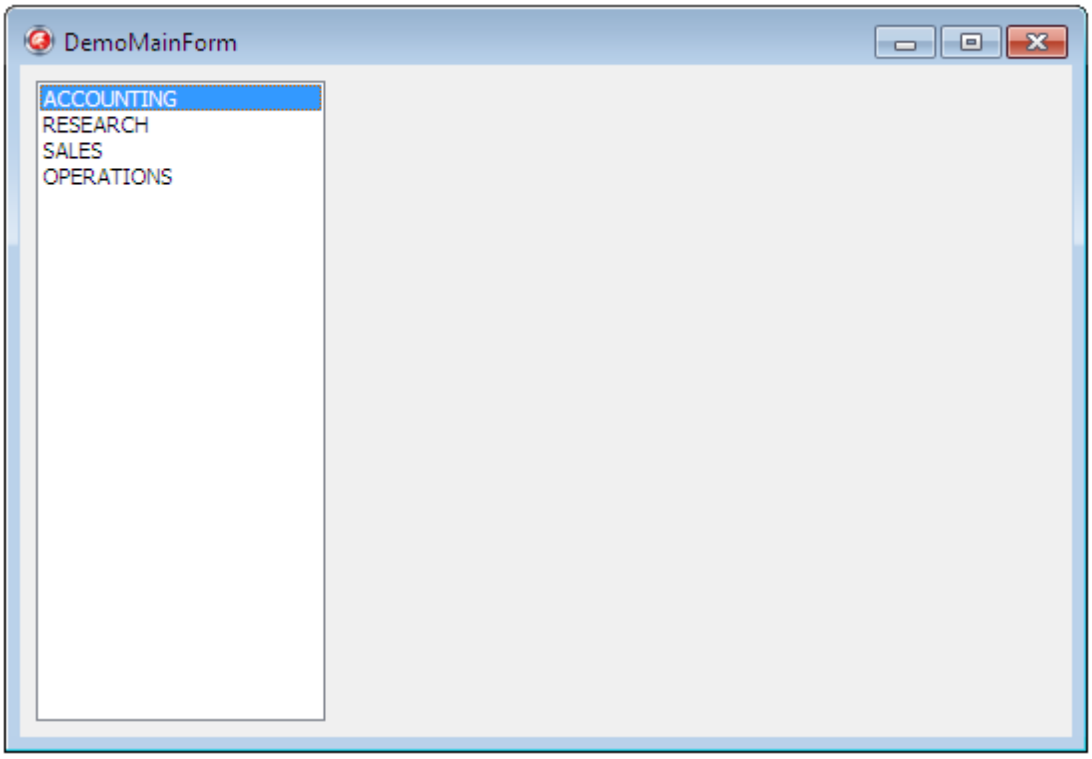


Then, declare the *PopulateDepts* form method, write its implementation like the following and add the method call to the form *OnShow* event handler:

```
procedure TDemoMainForm.FormShow(Sender: TObject);
begin
  Connect;
  PopulateDepts;
end;
procedure TDemoMainForm.PopulateDepts;
var
  Dept: TDept;
begin
  FDepts := FContext.GetEntities<TDept>;
  for Dept in FDepts do
    lbDepts.Items.Add(Dept.DName);
end;
```

In the *PopulateDepts* method we first obtain a complete collection of *TDept* instances (a complete list of departments) and then populate the *lbDepts* with department names.

Try to compile and run the application, and you will see the form with the department list:

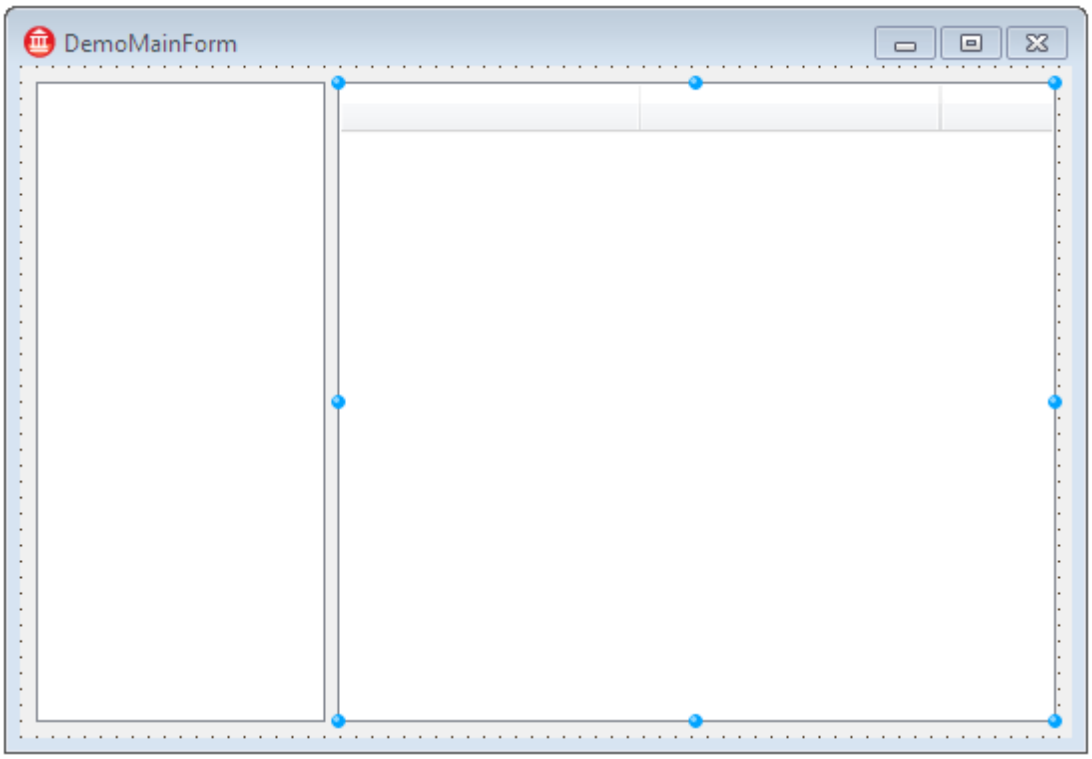


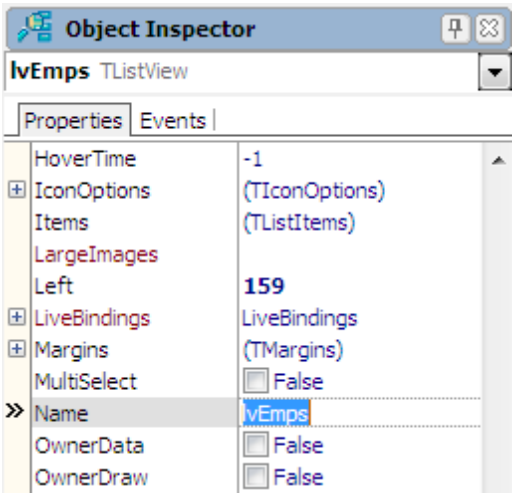
Step 7. Populate the employees list

Next task of the objective is to display a list of the selected department employees. For

example, let's display names and job names of employees. Since the employee list will contain more than one data column, we will use the *TListView* component to implement it.

In the form designer, drop the *TListView* component on the form and name it *lvEmps*. Then set the component *ViewStyle* property to *vsReport* and create two columns in the *Columns* property editor:





Declare the *PopulateEmps* form method, write its implementation like the following and add

the method call to the *OnClick* event handler of the *lbDepts* component:

```
procedure TDemoMainForm.lbDeptsClick(Sender: TObject);
begin
  PopulateEmps;
end;
procedure TDemoMainForm.PopulateEmps;
var
  Emps: IEntityEnumerable<TEmp>;
  Emp: TEmp;
  Item: TListItem;
begin
  lvEmps.Items.Clear;
  if lbDepts.ItemIndex < 0 then
    Exit;
  Emps := FDepts[lbDepts.ItemIndex].Emps;
  for Emp in Emps do begin
    Item := lvEmps.Items.Add;
    Item.Caption := Emp.Ename;
    Item.SubItems.Add(Emp.Job);
  end;
end;
```

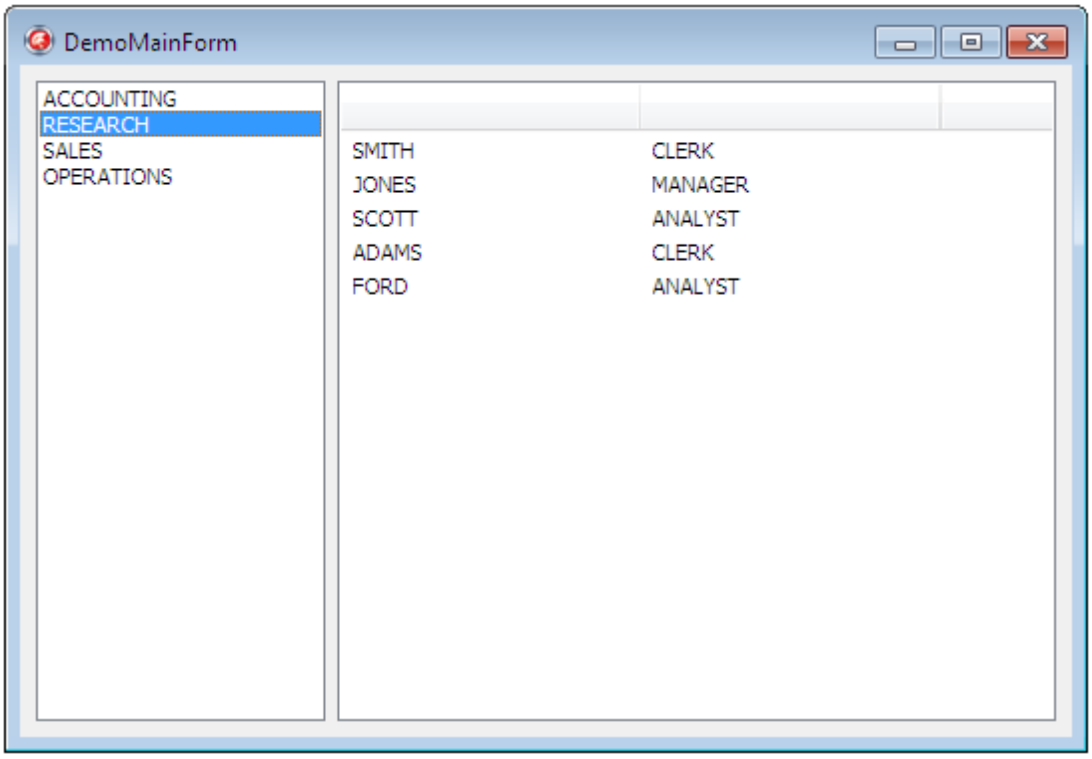
In the *PopulateEmps* method we take the department entity instance from the *FDepts* collection by the index corresponding to the selected item index in the *lbDepts* component, and then access its employees collection directly through the *TDept.Emps* property. This is the easiest way to access entity's referenced collection.

Another way is to obtain employees list using the *TEntityContext.GetEntities* method with a condition. Here is an example of such approach:

```
procedure TDemoMainForm.PopulateEmps;
var
  Dept: TDept;
  Emps: IEntityEnumerable<TEmp>;
  Emp: TEmp;
  Item: TListItem;
begin
  lvEmps.Items.Clear;
  if lbDepts.ItemIndex < 0 then
    Exit;
  Dept := FDepts[lbDepts.ItemIndex];
  Emps := FContext.GetEntities<TEmp>('deptno = ' + IntToStr(Dept.Deptno));
  for Emp in Emps do begin
    Item := lvEmps.Items.Add;
    Item.Caption := Emp.Ename;
    Item.SubItems.Add(Emp.Job);
  end;
end;
```

Compile and run your application. We have a department list on the application main form. When clicking a department name, the employees list is filled with the department employees

names.



You can find the project units sources in the article appendix.

Sources:

- DemoClasses.pas
- DemoProject.dpr
- DemoMainFormUnit.pas
- DemoMainFormUnit.dfm

Request Support DAC Forum Provide Feedback

3.4.1 DemoProject.dpr

```
program DemoProject;
uses
  Vcl.Forms,
  DemoMainFormUnit in 'DemoMainFormUnit.pas' {DemoMainForm},
  DemoClasses in 'DemoClasses.pas';
{$R *.res}
```

```
begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TDemoMainForm, DemoMainForm);
  Application.Run;
end.
```


Request Support DAC Forum Provide Feedback

3.4.2 DemoMainFormUnit.pas

```
unit DemoMainFormUnit;
interface
uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
  System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs,
  Vcl.StdCtrls, Vcl.Grids, Vcl.ComCtrls,
  EntityDAC.Entity,
  EntityDAC.EntityConnection,
  EntityDAC.EntityContext,
  EntityDAC.DataProvider.UniDAC,
  SQLiteUniProvider,
  DemoClasses;
type
  TDemoMainForm = class(TForm)
    lbDepts: TListBox;
    lvEmps: TListView;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure lbDeptsClick(Sender: TObject);
  private
    FConnection: TEntityConnection;
    FContext: TEntityContext;
    FDepts: IEntityEnumerable<TDept>;
  public
    procedure Connect;
    procedure PopulateDepts;
    procedure PopulateEmps;
  end;
var
  DemoMainForm: TDemoMainForm;
implementation
{$R *.dfm}
{ TDemoMainForm }
procedure TDemoMainForm.FormCreate(Sender: TObject);
begin
  FConnection := TEntityConnection.Create;
  FContext := TEntityContext.Create;
  FContext.Connection := FConnection;
  FContext.ModelName := 'Demo';
end;
procedure TDemoMainForm.FormDestroy(Sender: TObject);
begin
```

```
  FConnection.Free;
  FContext.Free;
end;
procedure TDemoMainForm.FormShow(Sender: TObject);
begin
  Connect;
  PopulateDepts;
end;
procedure TDemoMainForm.lbDeptsClick(Sender: TObject);
begin
  PopulateEmps;
end;
procedure TDemoMainForm.Connect;
begin
  FConnection.ProviderName := 'UniDAC';
  FConnection.DialectName := 'SQLite';
  FConnection.ConnectionString := 'ProviderName=SQLite;Direct=True;DataBase=
  FConnection.Connect;
end;
procedure TDemoMainForm.PopulateDepts;
var
  Dept: TDept;
begin
  FDepts := FContext.GetEntities<TDept>;
  for Dept in FDepts do
    lbDepts.Items.Add(Dept.Dname);
end;
procedure TDemoMainForm.PopulateEmps;
var
  Emps: IEntityEnumerable<TEmp>;
  Emp: TEmp;
  Item: TListItem;
begin
  lvEmps.Items.Clear;
  if lbDepts.ItemIndex < 0 then
    Exit;
  Emps := FDepts[lbDepts.ItemIndex].Emps;
  for Emp in Emps do begin
    Item := lvEmps.Items.Add;
    Item.Caption := Emp.Ename;
    Item.SubItems.Add(Emp.Job);
  end;
end;
end.
```


3.4.3 DemoMainFormUnit.dfm

```
object DemoMainForm: TDemoMainForm
  Left = 0
  Top = 0
  Caption = 'DemoMainForm'
  ClientHeight = 337
```

```
  ClientWidth = 527
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  OldCreateOrder = False
  OnCreate = FormCreate
  OnDestroy = FormDestroy
  OnShow = FormShow
  PixelsPerInch = 96
  TextHeight = 13
  object lbDepts: TListBox
    Left = 8
    Top = 8
    Width = 145
    Height = 321
    ItemHeight = 13
    TabOrder = 0
    OnClick = lbDeptsClick
  end
  object lvEmps: TListView
    Left = 159
    Top = 8
    Width = 360
    Height = 321
    Columns = <
      item
        Width = 150
      end
      item
        Width = 150
      end>
    TabOrder = 1
    ViewStyle = vsReport
  end
end
```


Request Support DAC Forum Provide Feedback

3.4.4 DemoClasses.pas

```
unit DemoClasses;
interface
uses
  SysUtils, Classes,
  EntityDAC.EntityAttributes,
  EntityDAC.EntityContext,
  EntityDAC.MetaData,
  EntityDAC.MetaEntity,
  EntityDAC.NullableTypes,
  EntityDAC.Types;
type
```

```
  TDept = class;
  TEmp = class;
  TDeptEmps = class;
  [Table('DEPT')]
  [Model('Demo')]
  [Key('FDeptno')]
  TDept = class(TMappedEntity)
  private
    [Column('DEPTNO', [ReadOnly])]
    FDeptno: TIntegerAttribute;
    [Column('DNAME', 14)]
    FDname: TStringAttribute;
    [Column('LOC', 13)]
    FLoc: TStringAttribute;
    [Column]
    [Collection('TEmp', 'FDept', 'FDeptno', 'FDeptno', srNone, drNone)]
    FEmps: TDeptEmps;
    function GetDeptno: Integer;
    function GetDname: String;
    procedure SetDname(const Value: String);
    function GetLoc: String;
    procedure SetLoc(const Value: String);
  protected
    constructor Create(AMetaType: TMetaType); overload; override;
  public
    constructor Create; overload; override;
    property Deptno: Integer read GetDeptno;
    property Dname: String read GetDname write SetDname;
    property Loc: String read GetLoc write SetLoc;
    property Emps: TDeptEmps read FEmps;
  end;
  [Table('EMP')]
  [Model('Demo')]
  [Key('FEmpno')]
  TEmp = class(TMappedEntity)
  private
    [Column('EMPNO', [ReadOnly])]
    FEmpno: TIntegerAttribute;
    [Column('ENAME', 10)]
    FEname: TStringAttribute;
    [Column('JOB', 9)]
    FJob: TStringAttribute;
    [Column('MGR', [CanBeNull])]
    FMgr: TIntegerNullableAttribute;
    [Column('HIREDATE')]
    FHiredate: TDateTimeAttribute;
    [Column('SAL', [CanBeNull])]
    FSal: TDoubleNullableAttribute;
    [Column('COMM', [CanBeNull])]
    FComm: TDoubleNullableAttribute;
    [Column('DEPTNO', [CanBeNull])]
    FDeptno: TIntegerNullableAttribute;
    [Column]
    [Reference('TDept', 'FEmps', 'FDeptno', 'FDeptno', srNone, drNone)]
    FDept: TMappedReference;
    function GetEmpno: Integer;
    function GetEname: String;
```

```
    procedure SetEname(const Value: String);
    function GetJob: String;
    procedure SetJob(const Value: String);
    function GetMgr: IntegerNullable;
    procedure SetMgr(const Value: IntegerNullable);
    function GetHiredate: TDateTime;
    procedure SetHiredate(const Value: TDateTime);
    function GetSal: DoubleNullable;
    procedure SetSal(const Value: DoubleNullable);
    function GetComm: DoubleNullable;
    procedure SetComm(const Value: DoubleNullable);
    function GetDeptno: IntegerNullable;
    procedure SetDeptno(const Value: IntegerNullable);
  protected
    constructor Create(AMetaType: TMetaType); overload; override;
    function GetDept: TDept;
    procedure SetDept(const Value: TDept);
  public
    constructor Create; overload; override;
    property Empno: Integer read GetEmpno;
    property Ename: String read GetEname write SetEname;
    property Job: String read GetJob write SetJob;
    property Mgr: IntegerNullable read GetMgr write SetMgr;
    property Hiredate: TDateTime read GetHiredate write SetHiredate;
    property Sal: DoubleNullable read GetSal write SetSal;
    property Comm: DoubleNullable read GetComm write SetComm;
    property Deptno: IntegerNullable read GetDeptno write SetDeptno;
    property Dept: TDept read GetDept write SetDept;
  end;
  TDeptEmps = class(TMappedCollection<TEmp>)
  end;
implementation
uses
  EntityDAC.Utils;
{ TDept }
constructor TDept.Create(AMetaType: TMetaType);
begin
  inherited Create(AMetaType);
  FDeptno := TIntegerAttribute.Create(Attributes, MetaType.MetaAttributes.Ge
  FDname := TStringAttribute.Create(Attributes, MetaType.MetaAttributes.Get(
  FLoc := TStringAttribute.Create(Attributes, MetaType.MetaAttributes.Get('L
  FEmps := TDeptEmps.Create(Self, MetaType.MetaCollections.Get('Emps'));
end;
constructor TDept.Create;
begin
  Create(Models.GetMetaType(Self.ClassType));
end;
function TDept.GetDeptno: Integer;
begin
  Result := FDeptno.Value;
end;
function TDept.GetDname: String;
begin
  Result := FDname.Value;
end;
procedure TDept.SetDname(const Value: String);
begin
```

```
  FDname.Value := Value;
end;
function TDept.GetLoc: String;
begin
  Result := FLoc.Value;
end;
procedure TDept.SetLoc(const Value: String);
begin
  FLoc.Value := Value;
end;
{ TEmp }
constructor TEmp.Create(AMetaType: TMetaType);
begin
  inherited Create(AMetaType);
  FEmpno := TIntegerAttribute.Create(Attributes, MetaType.MetaAttributes.Get
  FEname := TStringAttribute.Create(Attributes, MetaType.MetaAttributes.Get(
  FJob := TStringAttribute.Create(Attributes, MetaType.MetaAttributes.Get('J
  FMgr := TIntegerNullableAttribute.Create(Attributes, MetaType.MetaAttribut
  FHiredate := TDateTimeAttribute.Create(Attributes, MetaType.MetaAttributes
  FSal := TDoubleNullableAttribute.Create(Attributes, MetaType.MetaAttribute
  FComm := TDoubleNullableAttribute.Create(Attributes, MetaType.MetaAttribut
  FDeptno := TIntegerNullableAttribute.Create(Attributes, MetaType.MetaAttri
  FDept := TMappedReference.Create(Self, MetaType.MetaReferences.Get('Dept')
end;
constructor TEmp.Create;
begin
  Create(Models.GetMetaType(Self.ClassType));
end;
function TEmp.GetEmpno: Integer;
begin
  Result := FEmpno.Value;
end;
function TEmp.GetEname: String;
begin
  Result := FEname.Value;
end;
procedure TEmp.SetEname(const Value: String);
begin
  FEname.Value := Value;
end;
function TEmp.GetJob: String;
begin
  Result := FJob.Value;
end;
procedure TEmp.SetJob(const Value: String);
begin
  FJob.Value := Value;
end;
function TEmp.GetMgr: IntegerNullable;
begin
  Result := FMgr.Value;
end;
procedure TEmp.SetMgr(const Value: IntegerNullable);
begin
  FMgr.Value := Value;
end;
function TEmp.GetHiredate: TDateTime;
```

```
begin
  Result := FHiredate.Value;
end;
procedure TEmp.SetHiredate(const Value: TDateTime);
begin
  FHiredate.Value := Value;
end;
function TEmp.GetSal: DoubleNullable;
begin
  Result := FSal.Value;
end;
procedure TEmp.SetSal(const Value: DoubleNullable);
begin
  FSal.Value := Value;
end;
function TEmp.GetComm: DoubleNullable;
begin
  Result := FComm.Value;
end;
procedure TEmp.SetComm(const Value: DoubleNullable);
begin
  FComm.Value := Value;
end;
function TEmp.GetDeptno: IntegerNullable;
begin
  Result := FDeptno.Value;
end;
procedure TEmp.SetDeptno(const Value: IntegerNullable);
begin
  FDeptno.Value := Value;
end;
function TEmp.GetDept: TDept;
begin
  Result := FDept.Value as TDept;
end;
procedure TEmp.SetDept(const Value: TDept);
begin
  FDept.Value := Value;
end;
{*******************************************************************************
{  The following code is used for automatic entity mapping
{*******************************************************************************
initialization
  ForceRtti(TDept);
  ForceRtti(TEmp);
end.
```


Request Support DAC Forum Provide Feedback

3.5 Deployment

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

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

Trial Limitation Warning

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

EntityDACXX.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:

EntityDACXX.bpl	always

Additional requirements

If you use XML-mapping in your applicaiton, you must also deploy an XML file containing the data model.

Request Support DAC Forum Provide Feedback

3.6 Documentation

The EntityDAC reference documentation contains detailed ORM information. Many of the EntityDAC classes inherit or implement members from other Delphi classes or interfaces. The ORM documentation includes a summary of all members within each of these classes. For more detailed information about a specific inherited member, see the appropriate topic in the RAD Studio reference. For information about using Devart data providers or separate Data Access Components, see appropriate product documentation.

EntityDAC documentation is fully integrated into RAD Studio environment and available from EntityDAC | Help menu. EntityDAC help system provides information for all the included components, class members description, their properties, methods, and usage samples.

You can also download documentation in CHM or PDF formats from our site on EntityDAC download page.

Request Support DAC Forum Provide Feedback

3.7 Demos Overview

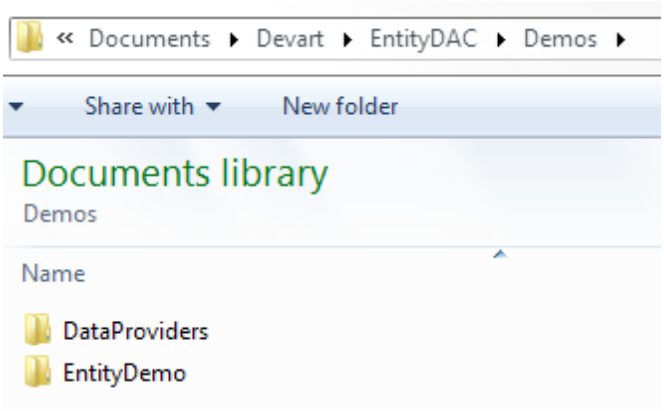
EntityDAC includes a number of demo projects that show off the main EntityDAC functionality and development patterns.

The EntityDAC demo projects consist of one large project called EntityDemo with demos for all main EntityDAC components, use cases, and ORM technologies, and a number of smaller projects on how to integrate EntityDAC with third-party components.

Where are the EntityDAC demo projects located?

The path to the EntityDAC demo projects folder is specified during the EntityDAC installation. By default, EntityDAC demo projects are located in the shared documents folder, for example

"C:\Users\Public\Documents\Devart\EntityDAC\Demos\EntityDemo\".



EntityDemo is the main demo project that shows off all the EntityDAC functionality.

The *DataProviders* directory contains a number of data providers for most known third-party data access components.

Note: The demo data providers may require installation of corresponding third-party components to compile and work properly.

Demo project descriptions

- EntityDemo
- DataProviders

See also:

- EntityDemo
- DataProvidersDemo
- Connection String

Request Support DAC Forum Provide Feedback

3.7.1 EntityDemo

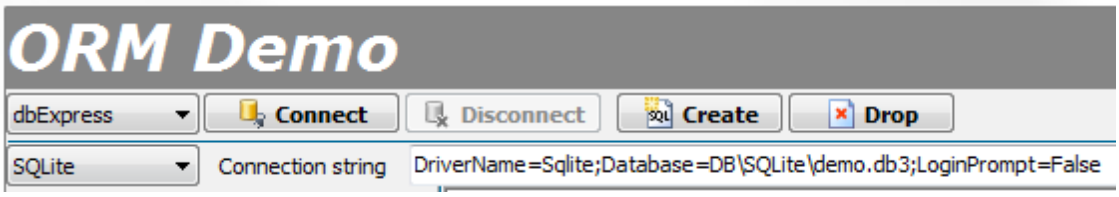
Launching EntityDemo

1. Launch your IDE.
2. In your IDE, choose File | Open Project from the menu bar.
3. Find the directory you installed EntityDAC 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.

Connecting to the EntityDemo database.

To start exploring the main EntityDAC demo:

1. Choose a data provider from the list of installed data providers at the top-left corner of the main demo form.
2. Select desired SQL dialect (if the data provider supports multiple dialects) from the list below.
3. Fill up the connection string for the selected data provider.
4. Press the "Connect" button.
5. If needed, create all necessary database objects by pressing the "Create" button (or re-create database objects by sequentially pressing "Drop" and "Create" buttons).



Description of specific connection string parameters for all supported data providers you can find in the Connection String article.

Note: For simplifying the process of of acquaintance with the possibilities of EntityDAC, we supply a ready-to-work SQLite demo database with the demo project. The database is placed in the "%EntityDemo%\DB\SQLite\" folder.

Also, when the demo starts for the first time, it automatically checks whether UniDAC, LiteDAC or the default dbExpress SQLite driver installed and fills the connection string for the provider found.

EntityDemo content

The main demo project includes a number of demos that show various aspects of EntityDAC. The following table describes all demos contained in the project.

General Demos

Name	Description
Working with DataContext	Demonstrates how to perform general tasks with entities. The section includes four identical sub-sections for the four mapping types supported by EntityDAC.
Working with EntityDataSet	Demonstrates how to bind entities and collections of entities to data-aware controls.
Working with LINQ queries	Demonstrates how to retrieve various entity collections using LINQ queries.

See also:

- DataProvidersDemo
- Connection String

Request Support DAC Forum Provide Feedback

3.7.2 DataProviders Demo

By default, EntityDAC includes data providers for Devart Data Access Components:

- UniDAC
- ODAC
- SDAC
- MyDAC
- IBDAC
- PgDAC
- LiteDAC

And for default data access components supplied with Delphi:

- ADO
- IBX
- dbExpress
- FireDAC

The DataProviders demo folder contains several sample data providers for most known third-party data access components:

- BDE Data Provider
- DOA Data Provider
- FIbPlus Data Provider
- NexusDB Data Provider
- Zeos Data Provider

See the Data Providers Installation article to learn how to install them to your IDE.

See also:

- Data Providers Installation
- Connection String
- EntityDemo

Request Support DAC Forum Provide Feedback

4 Using EntityDAC

4.1 Terms

Database model

Database model is the relational data representation. In other words, it is the set of all database tables, their fields and relations between tables.

Object model

Object model is the set of Delphi classes (see Entity) used for operating model data in the code.

Meta-model

Meta-model is a list of special Delphi classes that describes the database model. It stores description of tables and their data fields, and is used for correct retrieving/storing entities from/to the database.

Entity

Entity is a base class intended to represent a database model object in the object model. Entity is specifically designed for use in EntityDAC, its life cycle is completely controlled by a data context and its use is most simple and convenient for the developer.

Mapping

Mapping is the mechanism that creates the meta-model for the database model and sets the correspondence between entities and meta-model objects.

Data context

Data context is a mechanism that manages the entities within their life cycle. Its functions are: creating and initializing new entity instances, retrieving and storing entities from/to the database, storing used entities in the cache for future use, destroying of unused entities.

Request Support DAC Forum Provide Feedback

4.2 Creating Model

Database-first

Database-first development means that existing database schema is the starting point for the application. Both meta-model and object model are generated depending on database model using Entity Developer - specialized modeling tool designed to work with EntityDAC. To make changes, the database structure has to be changed first, then the meta-model and object model have to be regenerated to use with new database schema.

Model-first

Model-first development starts with a high-level description of the meta-model created using Entity Developer. Object model and database schema are generated from the meta-model. To make changes, developer has to change the model description and then regenerate both object model code and database schema from this model.

Code-first

Code-first development is suitable the existing application adapted for use with EntityDAC. In this case, meta-model and database schema are generated based on existing object model. When the code changed, then the meta-model and object model have to be regenerated.

Request Support DAC Forum Provide Feedback

4.3 Model Mapping

There are four different mapping mechanisms in EntityDAC. Their principal differences are: which classes constitute the object model, and how the process of meta-model creation and its association with the object model is implemented.

Code Mapped Entities

The object model consists of a set of entities (TEntity class descendants). The meta-model is previously generated as a set of meta-data classes in a separate unit. Mapping is hard-coded in the entity classes implementation.

Benefits of this type of mapping are:

- using entities in the object model eliminates the need to keep track of their life cycle, because it is engaged in the data context;
- the meta-model is available at the development stage, that is most clear to the developer;
- mapping is previously hard-coded, that is the best from the performance point.

Disadvantages:

- making changes in the object model becomes more complicated, since changing / adding

entities' properties requires corresponding changes in the meta-data unit.

Attribute-mapped entities

Object model consists of a set of entities (TEntity class descendants), marked with special mapping attributes. A separate meta-model unit is not used. Instead, the meta-model and mapping are generated dynamically at run-time, based on mapping attributes.

Benefits:

- entity mapping is implemented directly in the entity declaration, which eliminates the need to refer to a separate unit to find out the mapping features;
- easier code maintenance, because there is no need to make parallel changes in several units.

Disadvantages:

- entity classes declaration is some complicated with additional attributes;
- the meta-mode and mapping generation take place when the application starts, it takes some time.

XML-mapped entities

Object model consists of a set of entities (TEntity class descendants). A separate meta-model unit is not used. The meta-model and mapping are defined in an external XML-file (for example, an Entity Developer project file).

Benefits:

- the meta-model is available at design-time, which makes possible to open and set up datasets during the application design;
- it is possible to make some mapping changes with no need to re-compile the application.

Disadvantages:

- the meta-mode and mapping generation take place when the application starts, it takes some time;
- the XML-file containing the meta-model is somewhat difficult to understand.

Attribute-mapped objects

The same as "Attribute-mapped entities" mapping, except that the object model does not consist of TEntity, but of TObject descendants.

Benefits:

- it is possible to mark any existing classes with mapping attributes and use them in EntityDAC.

Disadvantages:

- in EntityDAC, working with classes, that are not TEntity descendants, is more difficult.

Request Support DAC Forum Provide Feedback

4.3.1 Attribute-Mapped Entities

There are two main attribute types in EntityDAC used to define entity class mapping. Class attributes are used to specify particular mapping parameters of the whole class. Property attributes are used to define specific mapping parameters of the class properties/fields. Also, attributes can be required and optional.

Table

A basic attribute that marks the class as an entity. When the class is marked with this attribute, the entity manager automatically builds corresponding meta-type and sets up the class mapping. Classes not marked with the *Table* attribute are ignored.

Attribute type:
class attribute, required

Declaration:

```
[Table[(table)]]
```

Parameters:

```
table = string
```

An optional parameter that specifies the name of the database table which stores entity instances. If the parameter is not specified, the table name is automatically generated basing

on the name of the class (if the first character in the name is "T" - the first character is omitted).

Example:

```
[Table('EMP')]
TEmp = class
end;
```

Note:

For entities which implement Table-Per-Hierarchy inheritance, the *Table* attribute with the table name has to be specified only for the basic entity class. All descendant classes have to be marked with the *Table* attribute with blank table name.

Model

An attribute that specifies the meta-model name, which will contain the entity meta-type. The attribute has to be specified after the *Table* attribute.

Attribute type:
class attribute, required

Declaration:

```
[Model(model)]
```

Parameters:

```
model = string
```

A required parameter that specifies the name of the meta-model. If the meta-model with the specified name does not exist, it will be created automatically. The name can not be blank. In this case, an exception will be raised.

Example:

```
[Table]
[Model('TestModel')]
TEmp = class
end;
```

Key

An attribute that specifies a unique key that identifies an entity instance. The key has not necessarily to be corresponding to the unique key of the database table.

Attribute type:
class attribute, required

Declaration:

```
[Key(members)]
```

Parameters:

```
members = string
```

A required parameter that specifies the name (or the comma-separated list of names) of the class properties/fields that constitute the key. All the class members listed as components of the key have to be marked with the *Column* attribute.

Example:

```
[Table]
[Model('TestModel')]
[Key('FId')]
TEmp = class
end;
```

Note:
When declaring inheritance, the attribute has to be specified only for the basic class in the hierarchy. For all descendant classes it has to be omitted. Otherwise, an exception will be raised.

Inheritance

An attribute that specifies the inheritance settings.

Attribute type:
class attribute, optional

Declaration:

```
[Inheritance(type [, link] | (, discriminator, value))]
```

Parameters:

```
type = TInheritanceKind
TInheritanceKind = (ikTablePerType, ikTablePerHierarchy)
```

A required parameter that specifies the type of inheritance: Table-Per-Type (TPT) or Table-Per-Hierarchy(TPH).

```
link = string
```

An optional parameter for TPT inheritance that specifies the name of the database column which is the key for ancestor-descendant link. For the basic class in the hierarchy the parameter can be omitted. In this case, the column which holds the entity key is used in the link.

```
discriminator = string
```

A required parameter for TPH inheritance that specifies the name of the database column which holds the value that uniquely identifies the entity class type in the hierarchy.

```
value = string
```

A required parameter for TPH inheritance that specifies the unique value of the discriminator for the class type.

Example:

```
// TPT ancestor and descendant
 [Table]
 [Model('TestModel')]
 [Key('FId')]
 [Inheritance(ikTablePerType)]
 TTPTBase = class
 private
   [Column]
   FId: integer;
 end;
 [Table]
 [Model('TestModel')]
 [Inheritance(ikTablePerType, 'BASEID')]
 TTPTDerived = class(TTPHBase)
 private
   [Column]
   FBaseId: integer;
 end;
// TPH ancestor and descendant
 [Table]
 [Model('TestModel')]
 [Key('FId')]
 [Inheritance(ikTablePerHierarchy, 'DISCRIMINATOR', '0')]
 TTPHBase = class
 private
   [Column]
   FId: integer;
   [Column]
   FDiscriminator: integer;
 end;
 [Table('')]
 [Model('TestModel')]
 [Inheritance(ikTablePerHierarchy, 'DISCRIMINATOR', '1')]
 TTPHDerived = class(TTPHBase)
 end;
```

TablePerType

An attribute that specifies the Table-Per-Type inheritance settings. The attribute is a special case of the *Inheritance* attribute, and is introduced to simplify the notation.

Attribute type:
class attribute, optional

Declaration:

```
[TablePerType[(link)]]
```

Parameters:

```
link = string
```

An optional parameter that specifies the name of the database column which is the key for ancestor-descendant link. For the basic class in the hierarchy the parameter can be omitted. In this case, the column which holds the entity key is used in the link.

Example:

```
[Table]
[Model('TestModel')]
[Key('FId')]
[TablePerType]
TTPTBase = class
private
  [Column]
  FId: integer;
end;
[Table]
[Model('TestModel')]
[TablePerType('BASEID')]
TTPTDerived = class(TTPHBase)
private
  [Column]
  FBaseId: integer;
end;
```

TablePerHierarchy

An attribute that specifies the Table-Per-Hierarchy inheritance settings. The attribute is a special case of the *Inheritance* attribute, and is introduced to simplify the notation.

Attribute type:
class attribute, optional

Declaration:

```
[TablePerHierarchy(discriminator, value)]
```

Parameters:

```
discriminator = string
```

A required parameter for TPH inheritance that specifies the name of the database column which holds the value that uniquely identifies the entity class type in the hierarchy.

```
value = string
```

A required parameter for TPH inheritance that specifies the unique value of the discriminator for the class type.

Example:

```
[Table]
[Model('TestModel')]
[Key('FId')]
[Inheritance(ikTablePerHierarchy, 'DISCRIMINATOR', '0')]
TTPHBase = class
private
  [Column]
  FId: integer;
  [Column]
  FDiscriminator: integer;
end;
[Table('')]
[Model('TestModel')]
[Inheritance(ikTablePerHierarchy, 'DISCRIMINATOR', '1')]
TTPHDerived = class(TTPHBase)
end;
```

Column

An attribute that defines the mapping of the class property/field to the database table column. All class members that does not have the *Column* attribute are not mapped.

Attribute type:
property attribute, optional

Declaration:

```
[Column([name[(, precision, scale) | , length][, default]][, options])]
```

Parameters:

```
name = string
```

An optional parameter that specifies the name of the database table column. If the name is not specified, the column name is automatically generated based on the class member name

(if the first character in the name is "F" - the first character is omitted).

```
precision = integer
scale = integer
```

An optional parameters that specify the precision and scale for the class member of the numeric type. These parameters are used when comparing properties values and for a database creation.

```
length = integer
```

An optional parameter that specifies the length for the string-type class member. The parameter is used when comparing properties values and for a database creation.

```
default = string
```

An optional parameter that specify the default value for the class member. Since the parameter is of string type, its value is converted into the exact value depending on the class member type. Therefore, additional quotation for string or date-time values is not required.

```
options = TColumnAn attributeOptions
TColumnAn attributeOption = (CanBeNull, ReadOnly)
TColumnAn attributeOptions = set of TColumnAn attributeOption
```

Optional set parameter that specify additional options. By default the parameter value is empty, that means that the class member is non-nullable and writable.

Example:

```
[Table]
[Model('TestModel')]
[Key('FId')]
TEmp = class
private
  [Column('ID')]
  FId: integer;
  [Column('NAME', 50, [CanBeNull])]
  FName: string;
end;
```

Note:

The *Column* attribute can only be used for class members of scalar types. If the class member has any other(class, record, array, set) type, then the attribute is ignored. The only exception is properties of the special *Reference*<> and *Collection*<> types used for specifying entity associations.

Generator

An attribute that sets up the automatic property/field value generation.

Attribute type:

property attribute, optional

Declaration:

```
// gtTable, gtGuid, gtCustom
[Generator([type][, fires])]
// gtSequence, gtSequenceHiLo
[Generator(sequence[, max-lo][, fires])]
// gtTableHiLo
[Generator(table, column, key-field, key-value, max-lo[, fires])]
```

Parameters:

```
type = TGeneratorType
TGeneratorType = (gtTable, gtTableHiLo, gtSequence, gtSequenceHiLo,
                  gtGuid, gtCustom)
```

An optional parameter that specifies the generator type.

Generator type	Algorithm to compute the next value of the class property
gtTable3	Maximum existing value of the table column + 1
gtTableHiLo	The result of the the HiLo algorithm using the specified table column as a "high" value source
gtSequence	The next value of the specified sequence
gtSequenceHiLo	The result of the HiLo algorithm using the specified sequence as a "high" value source
gtGuid	A unique GUID value
gtCustom	The property value is generated in the TEntityContext.OnGetGeneratorValue event handler1

Type parameter can be specified only for *gtTable*, *gtGuid* and *gtCustom* generator types. The default parameter value is *gtCustom*. For *gtTableHiLo*, *gtSequence* and *gtSequenceHiLo* generators the type parameter is omitted.

```
fires = TGeneratorFires
TGeneratorFires = (gfOnCreate, gfOnInsert)
```

An optional parameter that specifies the moment when the generator fires and the property obtains its new value. When the parameter is set to *gfOnCreate*, then the generator fires

immediately on entity creation. When the parameter is set to *gfOnInsert*, then the generator fires when the entity is saved. The default parameter value is *gfOnCreate*.

```
sequence = string
```

A required parameter that specifies the sequence name for *gtSequence* and *gtSequenceHiLo* generators. When the generator is *gtSequence*, then the property obtains its next value of the specified sequence. When the generator is *gtSequenceHiLo*, then the property value is calculated using the HiLo algorithm, and the specified sequence is used as a "high" value source.

```
table, column, key-field, key-value = string
```

Required parameters that specifies the HiLo algorithm options of the *gtTableHiLo* generator. Table and column parameters specify the table and the column that holds the "high" value for the algorithm. *Key-field* specifies the table key field, and *key-value* specifies the table key field value, that identifies the record that holds the "high" value for the specified generator.

```
max-lo = integer
```

A required parameter for *gtTableHiLo* and *gtSequenceHiLo* generators that specifies the "max-low" value for the HiLo algorithm (the maximum "low" value, when the "high" value needs to be increased).

The HiLo (High/Low) algorithm is used to generate unique number series using two values: the "high"and the "low". The high value is used as a base for a series (or range) of numbers, while the size of this series is donated by the low value. A HiLo generator calculates a unique result value using the following steps:
- obtains and atomically increments the "high" value (from the sequence or from the specified table column);
*- consequentially increments the "low" value from 0 to "max-low" and calculates the result as the "high*max-low+low";*
- when the "low" value exceeds the "max-low" limit, the algorithm goes back to the first step.

Reference

An attribute that defines a "side" of the One-To-One or One-To-Many entity association, and marks a class property/field as a reference to another entity. Property that represents the reference, has to be of the *Reference<T: class>* type, a special generic record type declared in the *ObjectContext* unit. *Reference<T: class>* has the only public property used to access the referenced entity instance.

```
Reference<T: class> = record
public
  property Value: T;
end;
```

Attribute type:

property attribute, optional

Declaration:

```
[Reference(other-class, other-member, this-key, other-key[, save, delete])]
```

Parameters:

```
other-class = string
```

A required parameter that specifies the class name of the entity, which is the "opposite side" of the association.

```
other-member = string
```

A required parameter that specifies the name of corresponding member of another entity class, which represents the link to this entity class. The opposite property/field has to be either of type *Reference<T: class>* (One-To-One association), or of type *Collection<T: class>* (One-To-Many association).

```
this-key = string
```

A required parameter that specifies the name (or the comma-separated list of names) of the class member, which is used as a key for class association. When the reference is the "main side" in the One-To-One association, specified class members have to constitute the unique entity key. When the reference is a dependent in the One-To-One association (or is a side in the One-To-Many association), specified class members have to constitute the foreign key to the opposite class. All the class members listed as components of the key have to be marked with the *Column* attribute.

```
other-key = string
```

A required parameter that specifies the name (or the comma-separated list of names) of the member of the related class, which is used as the corresponding key for class association. All the opposite class members listed as components of the key have to be marked with the Column attribute.

```
save = TSaveRule
TSaveRule = (srNone, srCascade)
```

An optional parameter that specifies saving rules for the referenced class. When the property

is set to *srCascade*, the Save method is called cascade for the referenced entity, and for all its associated entities. The default value is *srNone*.

```
delete = TDeleteRule
TDeleteRule = (drNone, drCascade, drRestrict, drNoAction,
               drSetNull, drSetDefault)
```

An optional parameter that specifies an action applied for the referenced entity when this entity is deleted.

drNone	Nothing happened with the referenced entity when this entity was deleted
drCascade	The Delete method is called as a cascade for the referenced entity
drRestrict	The exception is raised when attempting to delete the entity
drNoAction	The entity is not deleted, no exception raised
drSetNull	The value of the opposite class member is set to the null value
drSetDefault	The value of the opposite class member is set to its default value

The default value is *drNone*.

Example:

```
[Table]
[Model('TestModel')]
[Key('FId')]
TEmp = class
private
  [Column('ID')]
  FId: integer;
  [Column('NAME', 50, [CanBeNull])]
  FName: string;
  [Column('DEPTNO')]
  FDeptno: Integer;
  [Column]
  [Reference('TDept', 'FEmps', 'FDeptno', 'FDeptno', srCascade, drNone)]
  FDept: Reference<TDept>;
end;
```

Note:

The *Reference* attribute has to be applied only on a class property/field having the *Column* attribute. There is no need to mark the corresponding properties of both related classes using association attributes. Since a property/field of one class is marked as *Reference*, the

corresponding property of the opposite class requires only the *Column* attribute.

Collection

An attribute that defines a "side" of the One-To-Many or Many-To-Many entity association, and marks a class property/field as a list of another entities. Property that represents the list, has to be of the *Collection<T: class>* type, a special generic record type declared in the *ObjectContext* unit. *Collection<T: class>* implements several properties and methods for access the collection members.

```
Collection<T: class> = record
public
  property Count: integer;
  property Value[Index: integer]: T; default;
  procedure Clear;
  procedure Add(const Value: T);
  function Contains(const Value: T): boolean;
  property ToEnumeration: IObjectCollection<T>;
end;
```

Attribute type:

property attribute, optional

Declaration:

```
[Collection(other-class, other-member, other-key[, save, delete])]
```

Parameters:

```
other-class = string
```

A required parameter that specifies the class name of the entity, which is the "opposite side" of the association.

```
other-member = string
```

A required parameter that specifies the name of corresponding member of another entity class, which represents the link to this entity class. The opposite property/field has to be either of type *Reference<T: class>* (One-To-Many association), or of type *Collection<T: class>* (Many-To-Many association).

```
other-key = string
```

A required parameter that specifies the name (or the comma-separated list of names) of the member of the related class, which is used as a key for class association. All the opposite class members listed as components of the key have to be marked with the *Column* attribute.

```
save = TSaveRule
TSaveRule = (srNone, srCascade)
```

An optional parameter that specifies saving rules for entities within the collection. When the property is set to *srCascade*, the Save method is called cascade for all classes in the collection. The default value is *srNone*.

```
delete = TDeleteRule
TDeleteRule = (drNone, drCascade, drRestrict, drNoAction,
               drSetNull, drSetDefault)
```

An optional parameter that specifies an action applied for entities within the collection when thies entity is deleted.

drNone	Nothing happened with associated entities when the entity is deleted
drCascade	The Delete method is called as a cascade for associated entities
drRestrict	The exception is raised when trying to delete the entity
drNoAction	The entity is not deleted, no exception raised
drSetNull	The value of the opposite classes members is set to the null value
drSetDefault	The value of the opposite classes members is set to their default value

The default value is *drNone*.

Example:

```
[Table]
[Model('TestModel')]
[Key('FDeptno')]
TDept = class
private
  [Column('DEPTNO')]
  FDeptno: integer;
  [Column('NAME', 50, [CanBeNull])]
  FName: string;
  [Column]
  [Collection('TEmp', 'FDept', 'FDeptno', 'FDeptno', srCascade, drCascade)]
  FEmps: Collection<TEmp>;
end;
```

Note:

The *Collection* attribute has to be applied only on a class property/field having the *Column* attribute. There is no need to specify the "this-key" parameter for *Collection* as it is required for *Reference*, because Collection always uses the unique entity key for association.

There is no need to mark the corresponding properties of both related classes using association attributes. Since a property/field of one class is marked as the *Collection*, the corresponding property of the opposite class requires only the *Column* attribute.

When defining the Many-To-Many association, the *LinkClass* attribute has also be specified for both association sides.

LinkClass

An attribute that specify the junction(cross-reference) entity class parameters for the Many-To-Many association. The attribute is used as an auxiliary attribute for the *Collection* attribute.

Attribute type:
property attribute, optional

Declaration:

```
[LinkClass(link-class, link-member, this-key, link-key)]
```

Parameters:

```
link-class = string
```

A required parameter that specifies the junction class name.

```
link-member = string
```

A required parameter that specifies the name of junction class member, which represents the reference to this entity class. The junction class property/field has to be of type *Reference<T: class>*.

```
this-key = string
```

A required parameter that specifies the name (or the comma-separated list of names) of the class member, which is used as a key for link with the junction class. All the class members listed as components of the key have to be marked with the *Column* attribute.

```
other-key = string
```

A required parameter that specifies the name (or the comma-separated list of names) of the junction class member, which is used as a foreign key for link. All the junction class members listed as components of the key have to be marked with the *Column* attribute.

Example:

```
[Table]
[Model('TestModel')]
```

```
[Key('FIdAuthor')]
TAuthor = class
private
  [Column('ID_AUTHOR')]
  FIdAuthor: Integer;
  [Column('AUTHORNAME', 128, [CanBeNull])]
  FAuthorname: String;
  [Column]
  [Collection('TBook', 'FAuthors')]
  [LinkClass('TAuthorsBooks', 'FAuthors', 'FIdAuthor', 'FAuthorId')]
  FBooks: Collection<TBook>;
end;
[Table]
[Model('TestModel')]
[Key('FIdBook')]
TBook = class
private
  [Column('ID_BOOK')]
  FIdBook: Integer;
  [Column('BOOKNAME', 128, [CanBeNull])]
  FBookname: String;
  [Column]
  [Collection('TAuthor', 'FBooks')]
  [LinkClass('TAuthorsBooks', 'FBooks', 'FIdBook', 'FBookId')]
  FAuthors: Collection<TAuthor>;
end;
[Table('AUTHORS_BOOKS')]
[Model('TestModel')]
[Key('FAuthorId, FBookId')]
TAuthorsBooks = class
private
  [Column('AUTHOR_ID')]
  FAuthorId: Integer;
  [Column('BOOK_ID')]
  FBookId: Integer;
  [Column]
  FAuthors: Reference<TAuthor>;
  [Column]
  FBooks: Reference<TBook>;
end;
```

Note:

The *LinkClass* attribute has to be applied only to a class property/field which has the *Column* attribute.

ColumnType

A special attribute designed to fix RTTI bug in Delphi version from XE to XE3. This attribute has to be specified for attribute-mapped object fields of type TBytes. The attribute has to precede to the Column attribute.

Attribute type:

property attribute, optional

Declaration:

```
{$IFDEF FIXBYTES}
  [ColumnType(TypeInfo(TBytes))]
{$ENDIF}
```


Request Support DAC Forum Provide Feedback

4.3.2 XML-Mapped Entities

When using the XML mapping, only entity classes unit are required in Delphi, and the mapping are defined in an external XML file of special format. This type of mapping are suitable when there is need to set up EntityDAC data-aware components at design-time. To using the XML mapping at run-time, the corresponding entity classes have to be marked with the *[XmlMapped]* class attribute.

Overall file structure

The overall structure of the document looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<Database>
  <Table></Table>
  <Table></Table>
  ...
</Database>
```

The document begins with the standard XML declaration. Body of the document consists of the root *Database* element which describes the meta-model parameters. The *Document* element includes a set of *Table* elements. Each Table element describes a particular entity class mapping.

Database

The root document element which describes the model parameters. Contains one or more *Table* elements.

Declaration:

```
<Database attributes>
  ...
</Database>
```

Attributes:

```
Name="model-name"
```

A required attribute which specifies the name of the meta-model being created.

Example:

```
<Database Name="TestMetaModel">
</Database>
```

Table

The document element which describes a particular entity class mapping. Located inside the *Document* element. Contains the one *Type* element which describes the meta-type, its attributes, associations and inheritance.

Declaration:

```
<Table attributes>
  <Type>
    ...
  </Type>
</Table>
```

Attributes:

```
Name="table-name"
```

A required attribute which specifies the name of the database table to which the meta-type is mapped.

Example:

```
<Database Name="TestMetaModel">
  <Table Name="EMP">
  </Table>
</Database>
```

Type

The document element which describes the meta-type, its attributes, associations and inheritance. Located inside the *Table* element. Contains one or many *Column* elements which describes the meta-type attributes. Also, can contain *Association* elements which describes associations, and nested *Type* elements which describes inherited meta-types.

Declaration:

```
<Type attributes>
  <Column>
    ...
  </Column>
```

```
  <InheritanceColumn/>
  <Association/>
  <Type>
    ...
  </Type>
</Type>
```

Attributes:

```
Name="type-name"
```

A required attribute which specifies the meta-type name. The specified meta-type name with the "T" prefix is used to locate the corresponding entity class to map to.

```
ed:Guid="guid"
```

A required attribute which specifies the unique identifier of the meta-type. The identifier has to be specified in the UUID format and is used internally when processing the XML document.

```
ed:InheritanceGuid="guid"
```

A required attribute for an inherited meta-type, which specifies the unique identifier of the meta-type in the hierarchy. The attribute has to be specified only for nested Type element which describes an inherited meta-type inside the parent Type element. The identifier has to be specified in the UUID format and is used internally when processing the XML document.

```
TableName="table-name"
```

A required attribute for a Table-Per-Type descendant meta-type, which specifies the database table name that stores corresponding mapped entities. The attribute has to be specified only for a nested *Type* element which describes the Table-Per-Type descendant meta-type.

```
InheritanceCode="value"
```

A required attribute for a Table-Per-Hierarchy meta-type, which specifies the unique value of the discriminator for the meta-type. The attribute has to be specified for all *Type* elements which describes the Table-Per-Hierarchy inheritance.

Example:

```
// a simple entity mapping definition
<Type Name="Emp" ed:Guid="ba491fdd-b2ae-4105-9ea3-fcb43d5bc8ad">
</Type>
// a TPT inheritance definition
<Type Name="Emp" ed:Guid="ba491fdd-b2ae-4105-9ea3-fcb43d5bc8ad"
      ed:InheritanceGuid="0d372a81-5ffe-476a-ae48-fb3ec75f5127">
  <Type Name="EmpInherited" ed:Guid="41cefe39-ac7f-4c35-8b9c-92d066820feb"
        ed:InheritanceGuid="20e82c82-284c-4301-b214-b3c5ea72eedc"
        TableName="EMP_INHERITED">
  </Type>
</Type>
```

```
// a TPH inheritance definition
 <Type Name="Emp" ed:Guid="ba491fdd-b2ae-4105-9ea3-fcb43d5bc8ad"
       ed:InheritanceGuid="0d372a81-5ffe-476a-ae48-fb3ec75f5127"
       InheritanceCode="0">
   <Type Name="EmpInherited" ed:Guid="41cefe39-ac7f-4c35-8b9c-92d066820feb"
         ed:InheritanceGuid="20e82c82-284c-4301-b214-b3c5ea72eedc"
         InheritanceCode="1">
   </Type>
 </Type>
```

Column

The document element which describes the meta-type attribute. Located inside the *Type* element. Can contain the *Generator* element which describes the meta-attribute value generator.

Declaration:

```
 <Column attributes>
   <Generator>
     ...
   </Generator>
 </Column>
```

Attributes:

```
 Name="column-name"
```

A required attribute which specifies the table column name which stores the corresponding meta-attribute values.

```
 Member="member-name"
```

A required attribute which specifies the meta-attribute name.

```
 Type="type-name"
```

A required attribute which specifies the meta-attribute type. The type name can be one of the following names:

- *AnsiMemo*
- *AnsiString*
- *BCD*
- *Boolean*
- *Blob*
- *Byte*

- *Bytes*
- *Currency*
- *Date*
- *DateTime*
- *Double*
- *Extended*
- *GUID*
- *Int64*
- *Integer*
- *LongWord*
- *Memo*
- *Object*
- *ShortInt*
- *Single*
- *SmallInt*
- *SQLTimeStamp*
- *String*
- *Time*
- *UInt64*
- *WideMemo*
- *WideString*
- *Word*
- *XML*

```
CanBeNull="true/false"
```

A required attribute which specifies, whether the meta-attribute can be set to a null value.

```
ed:Guid="guid"
```

A required attribute which specifies the unique identifier of the meta-attribute. The identifier

has to be specified in the UUID format and is used internally when processing the XML document.

```
IsPrimaryKey="true/false"
```

An optional attribute which specifies, whether the meta-attribute constitutes the entity primary key.

```
IsDbGenerated="true/false"
```

An optional attribute which specifies, whether the meta-attribute value is generated by the database, and the meta-attribute is read-only.

```
MaxLength="value"
```

An optional attribute for string meta-attributes which specifies the maximum value length.

```
Precision="value"
```

An optional attribute for numeric meta-attributes which specifies the value precision.

```
Scale="value"
```

An optional attribute for numeric meta-attributes which specifies the value scale.

```
IsDiscriminator="true/false"
```

An optional attribute which defines the meta-attribute as the discriminator for the meta-type. The attribute has to be specified for one of the meta-attributes of the base meta-type in the Table-Per-Hierarchy inheritance.

Example:

```
<Column Name="ID" Member="Id" Type="Integer" IsPrimaryKey="true"
        CanBeNull="false" ed:Guid="fcbbb349-6e86-4fa8-bf71-1fd735b0e22e" />
```

InheritanceColumn

The document element which describes the link between the descendant and base meta-types in the Table-Per-Type inheritance. Has to be the first sub-element of the Type element for all descendant meta-types in the Table-Per-Type hierarchy.

Declaration:

```
<InheritanceColumn attributes />
```

Attributes:

```
ThisName="column-name"
```

A required attribute which specifies the database table column of the descendant meta-type, which constitutes the link.

```
BaseName="column-name"
```

A required attribute which specifies the database table column of the base meta-type, which constitutes the link.

Example:

```
<InheritanceColumn ThisName="ID_REF" BaseName="ID" />
```

Generator

The document element which describes the meta-attribute value generator. Can be located inside the Column element. Contains one or more GeneratorParameter attributes which describe its parameters.

Declaration:

```
<Generator Name="generator-name">
  <GeneratorParameter/>
</Generator>
```

Attributes:

```
Name="generator-name"
```

A required attribute which specifies generator type. The attribute can have one of the following values:

Attribute value	Algorithm to compute the next value of the meta-attribute
Identity	Maximum existing value of the table column + 1
TableHiLo	The result of the the HiLo algorithm using the specified table column as a "high" value source
Sequence	The next value of the specified sequence
SequenceHiLo	The result of the HiLo algorithm using the specified sequence as a "high" value source
Guid	A unique GUID value
Custom	The meta-attribute value is generated in the TEntityContext.OnGetGeneratorValue event handler

Example:

```
<Generator Name="Identity" />
```

GeneratorParameter

The document element which describes the generator parameter. Located inside the Generator element.

Declaration:

```
 <GeneratorParameter Name="parameter-name" Value="parameter-value" />
```

A GeneratorParameter element necessarily has two attributes: Name and Value.

Attributes:

```
 Name="generator-name"
```

A required attribute which specifies the parameter name.

```
 Value="generator-value"
```

A required attribute which specifies the parameter value.

Example:

```
<GeneratorParameter Name="GeneratorFires" Value="OnCreate" />
```

Parameter combinations for different generator types:

Generator type	Parameter name	Parameter value	Description
All generators	GeneratorFires	OnCreate	The generator fires and the property obtains its new value immediately on entity creation
		OnSave	The generator fires and the property obtains its new value immediately when the entity is saved
TableHiLo	Table	<table-name>	The table that holds the "high" value for the HiLo algorithm
	Column	<column-name>	The table column that holds the "high" value for the HiLo algorithm

	KeyField	<key-field>	The table key field
	KeyFieldValue	<key-value>	The table key field value, that identifies the record that holds the "high" value
	MaxLo	<max-value>	The "max-low" value for the HiLo algorithm (the maximum "low" value, when the "high" value needs to be increased)
Sequence	Sequence	<sequence-name>	The sequence from which the property obtains its next value
SequenceHiLo	Sequence	<sequence-name>	The sequence which is used as a "high" value source for the HiLo algorithm
	MaxLo	<max-value>	The "max-low" value for the HiLo algorithm (the maximum "low" value, when the "high" value needs to be increased)

The HiLo (High/Low) algorithm is used to generate unique number series using two values: the "high"and the "low". The high value is used as a base for a series (or range) of numbers, while the size of this series is donated by the low value. A HiLo generator calculates a unique result value using the following steps:
- obtains and atomically increments the "high" value (from the sequence or from the specified table column);
- consequentially increments the "low" value from 0 to "max-low" and calculates the result as the "high*max-low +low";
- when the "low" value exceeds the "max-low" limit, the algorithm goes back to the first step.

Association

The document element which describes an association "side". Located inside the Type element. Full association declaration consists of two Association elements in the corresponding Type-s. The association type (One-To-Many, One-To-One, Many-To-Many) depends on its Cardinality attribute.

Declaration:

```
<Association attributes />
```

Attributes:

```
Name="association-name"
```

A required attribute which specifies the association name.

```
ed:AssociationGuid="guid"
```

A required attribute which specifies the unique identifier of the association. The corresponding Association element of the opposite "side" of the association must have the same AssociationGuid. The identifier has to be specified in the UUID format.

```
Cardinality="cardinality"
```

A required attribute for One-To-One and Many-To-many associations which specifies the association type. Both sides of the One-To-One association have to be marked with the Cardinality="One" attribute. Both sides of the Many-To-many association have to be marked with the Cardinality="Many" attribute. For the One-To-Many association the attribute has to be omitted.

```
Type="type-name"
```

A required attribute which specifies the meta-type name of the opposite side of the association.

```
IsForeignKey="true"
```

A required attribute for the One-To-Many association which specifies the "many" side of the association. For the "one" side the attribute has to be omitted.

```
Member="member-name"
```

A required attribute which specifies the meta-data member (either the meta-reference for "one" and the meta-collection for "many") name for the association "side".

```
ThisKey="member-name"
```

A required attribute which specifies the name of the meta-attribute which constitutes the link to the other "side" of the association.

```
OtherKey="member-name"
```

A required attribute which specifies the name of the opposite meta-attribute which constitutes the link.

```
LinkTableName="table-name"
```

A required attribute for the Many-To-Many association that specifies the name of the junction(cross-reference) table that contains links to both association sides.

```
LinkThisKey="column-name"
```

A required attribute for the Many-To-Many association that specifies the column name of the junction(cross-reference) table that contains the link to "this" side of the association.

```
LinkOtherKey="column-name"
```

A required attribute for the Many-To-Many association that specifies the column name of the junction(cross-reference) table that contains the link to the opposite side of the association.

```
Cascade="true/false"
```

A required attribute for the "parent" side of the association that specifies whether "child" entities have to be saved cascade when the parent entity saved.

```
DeleteRule="delete-rule"
```

A required attribute for the "parent" side of the association that specifies the behavior of "child" entities when attempting to delete the parent entity:

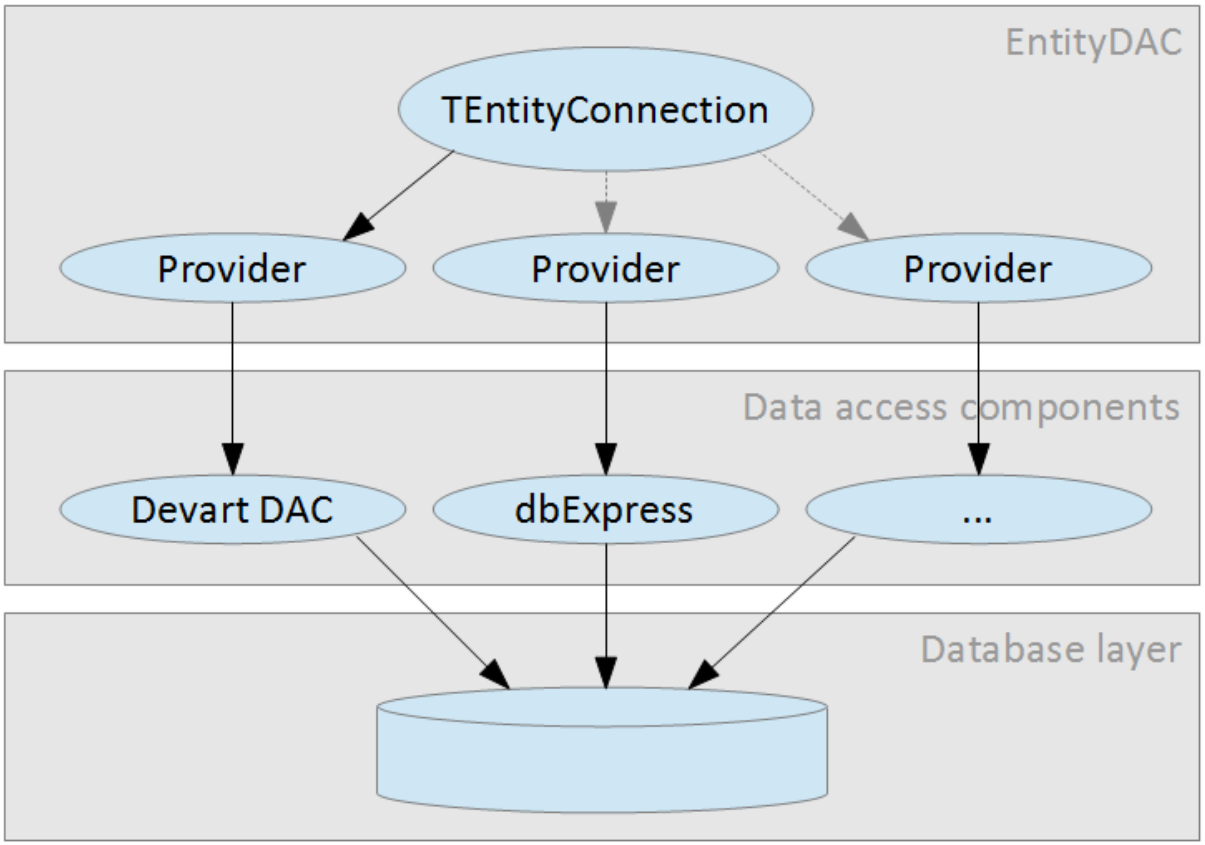
Attribute value	Description
Cascade	Child entities will also be deleted
Restrict	If child entities exist, the parent entity will not be deleted and the exception will be raised
NoAction	The parent entity will be deleted, child entities will not be affected
SetNULL	Link properties of child entities will be filled with the NULL value
SetDefault	Link properties of child entities will be filled with the default value

Request Support DAC Forum Provide Feedback

4.4 Database Connection

Since EntityDAC is an ORM framework, it is abstracted from the database layer, and the TEntityConnection component is not a database connection component itself. It uses uniform interface for database operations, and direct interaction with the database is handled by

specialized component packages. EntityDAC is designed to work with a wide set of data access components (DAC), such as Devart Data Access Components, ADO, dbExpress, IBX, FireDAC, ZEOS, FibPlus, AnyDAC etc. To provide communication between the TEntityConnection component and a particular DAC, a special class called "data provider" is used.



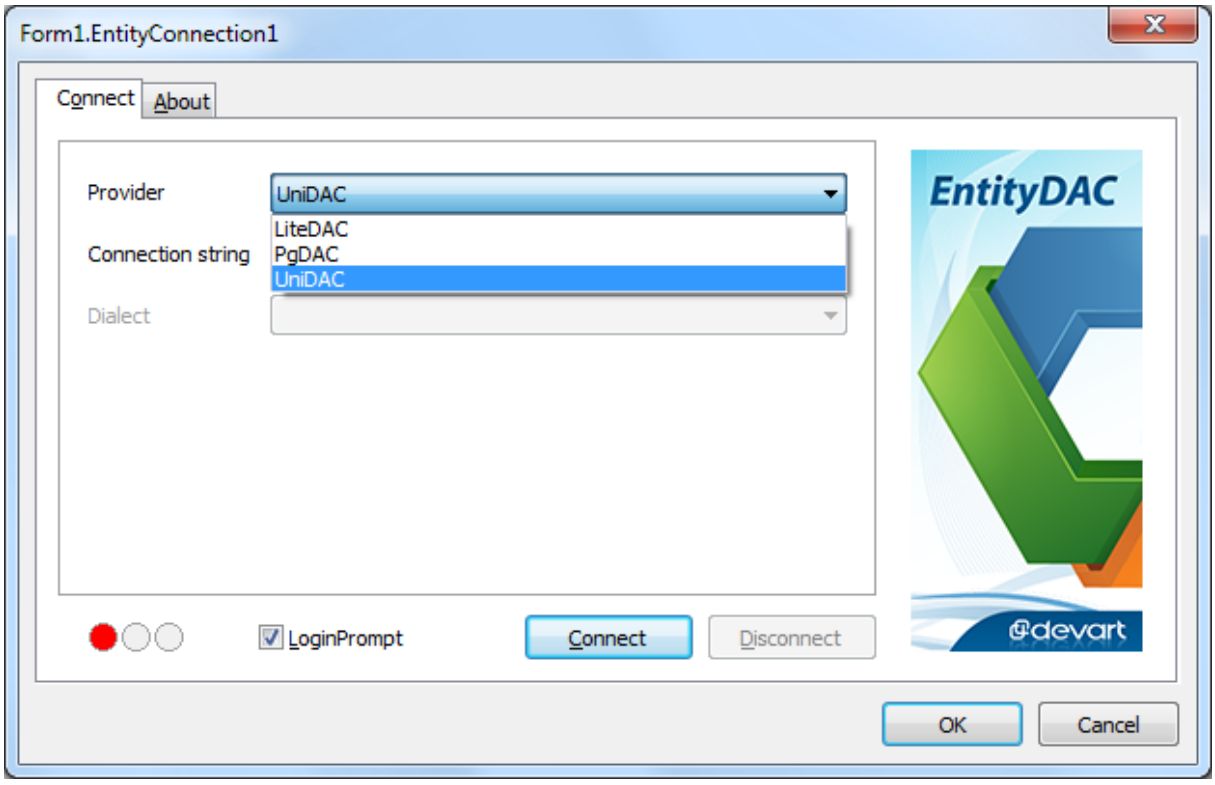
Data providers

Data provider is a special class that implements database operation functions needed by TEntityConnection for a particular data access component set. In order to the provider can be registered and used, either its unit has to be added to the form USES clause or the provider component has to be placed on the form.

```
uses
  UnidacDataProvider;
// ...
var
  Connection: TEntityConnection;
begin
  // create the connection
```

```
  Connection := TEntityConnection.Create(nil);
  // set the UniDAC data provider as the used provider
  Connection.ProviderName := 'UniDAC';
end;
```

Also, the desired provider can be set at design-time using the connection editor.



EntityDAC has predefined providers for most widespread component packs. Also, it is possible to create a custom EntityDAC data provider for using with any data access components. Examples of custom providers are included in EntityDAC demos.

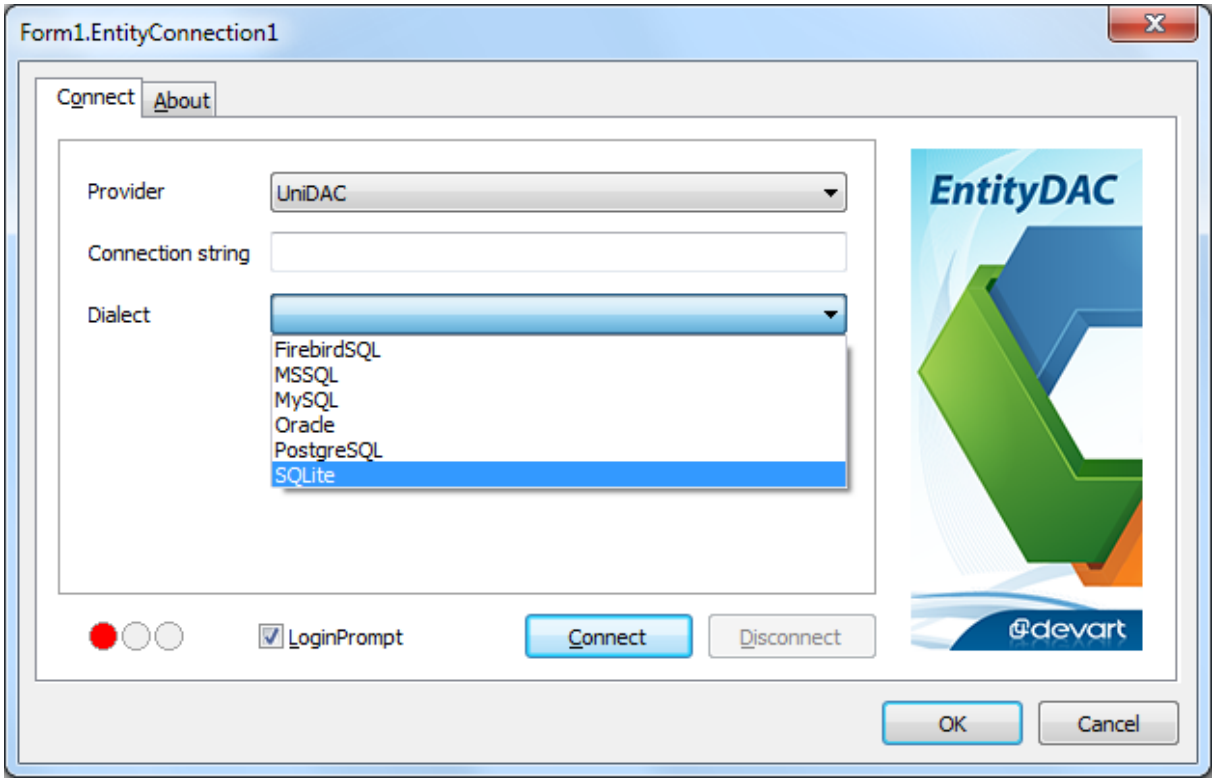
Dialects

While some data access components are designed to work with one particular database, other can interact with various databases (Devart UniDAC, dbExpress etc.). In order to specify an exact desired database, a data provider has a special property called "Dialect".

```
uses
  UnidacDataProvider;
// ...
var
  Connection: TEntityConnection;
begin
  Connection := TEntityConnection.Create(nil);
  Connection.ProviderName := 'UniDAC';
```

```
  // set the exact SQL dialect
  Connection.Dialect := 'SQLite';
end;
```

Or the same using the connection editor.



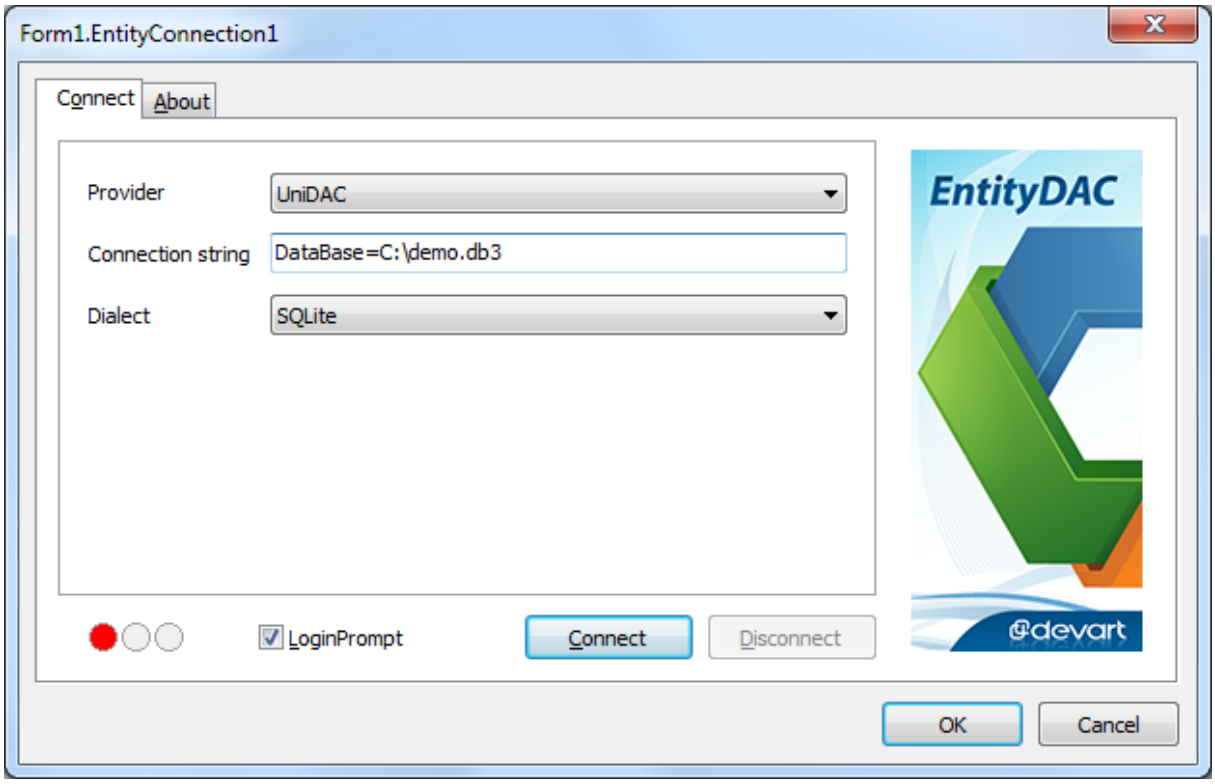
Connection string

Taking into account that the used data access components have different sets of properties to configure a database connection, EntityDAC provides the unified method to create a connection - connection string. Connection string consists of "parameter"="value" pairs separated by a semicolon. Each pair specifies the name and value of one of the connection parameters for a given data access component.

```
uses
  UnidacDataProvider;
// ...
var
  Connection: TEntityConnection;
begin
  Connection := TEntityConnection.Create(nil);
  Connection.ProviderName := 'UniDAC';
  Connection.Dialect := 'SQLite';
  // set up the connection parameters using the connection string
  Connection.ConnectionString := 'DataBase=C:\demo.db3';
```

```
end;
```

In the connection editor.



Description of specific connection string parameters for all supported data providers you can find in the Connection String article.

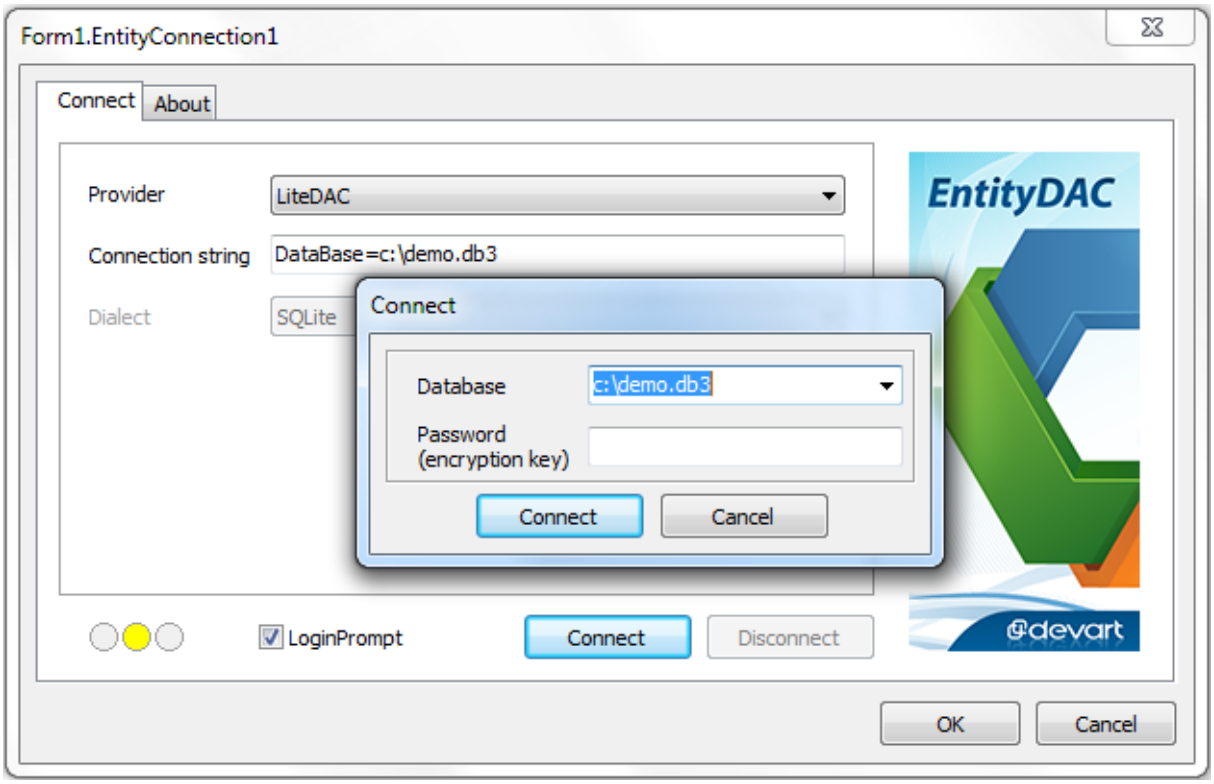
Connection dialog

As it is described above, EntityDAC works with different data access components which have different connection parameters. Therefore, EntityDAC does not have a built-in connection dialog. When the LoginPrompt property of TEntityConnection is set to True, the internal connection dialog of the used data access components will be displayed.

```
uses
  UnidacDataProvider;
// ...
var
  Connection: TEntityConnection;
begin
  Connection := TEntityConnection.Create(nil);
  Connection.ProviderName := 'UniDAC';
  Connection.Dialect := 'SQLite';
  Connection.ConnectionString := 'DataBase=C:\demo.db3';
  Connection.LoginPrompt := True;
```

```
  Connection.Connected := True;
end;
```

At design-time.



Request Support DAC Forum Provide Feedback

4.4.1 Using dbExpress Drivers

EntityDAC supports the work with various data-access components and dbExpress drivers. In order to use dbExpress driver as a data provider, you should assign the dbExpress value to the EntityConnection.ProviderName property, specify the used dialect in the EntityConnection.DialectName property and add in the USES section the following additional modules:

Devart dbExpress Drivers

Provider Name	USES contains
Devart dbExpress Driver for InterBase and Firebird	DbxDevartInterBase

Devart dbExpress Driver for SQLServer	DbxDevartSQLServer
Devart dbExpress Driver for MySQL	DbxDevartMySql
Devart dbExpress Driver for Oracle	DbxDevartOracle
Devart dbExpress Driver for SQLite	DbxDevartSQLite
Devart dbExpress Driver for PostgreSQL	DbxDevartPostgreSQL

Embarcadero dbExpress Drivers

Provider Name	USES contains
Embarcadero dbExpress Driver for Firebird	Data.DbxFirebird
Embarcadero dbExpress Driver for InterBase or To-Go databases	Data.DbxInterBase
Embarcadero dbExpress Driver for MS SQL Server	Data.DbxMSSQL
Embarcadero dbExpress Driver for MySQL	Data.DbxMySQL
Embarcadero dbExpress Driver for Oracle	Data.DbxOracle
Embarcadero dbExpress Driver for SQLite	Data.DbxSqlite

Request Support DAC Forum Provide Feedback

4.5 ConnectionString

Data Provider and SQL Dialect

DataProviders are components that the ORM uses to connect to a particular database, for example: UniDAC Data Provider for any database, ODAC Data Provider - for Oracle database, etc.

SQLDialect defines the used database name (e.g., SQLite, Oracle, SQL Server, etc.) to use the syntax and features of each particular database.

By default, an SQLite database and the TLiteDACDataProvider are used in the demo. You can use the following data providers in the EntityDemo:

- UniDAC Data Provider
- ODAC Data Provider
- SDAC Data Provider
- MyDAC Data Provider
- IBDAC Data Provider
- PgDAC Data Provider
- LiteDAC Data Provider
- ADO Data Provider
- IBX Data Provider
- dbExpress Data Provider
- FireDAC Data Provider

ConnectionString

To connect to your database, you should set the ConnectionString property for any used provider. It must include the used data provider name, SQL dialect*, and the database connection parameters string (depends on the used data provider). In addition, the Login Prompt parameter (True by default) can be specified.

For those providers, that don't support working with various databases, SQL Dialect will be set automatically.

TUniDACDataProvider

For TUniDACDataProvider set Data Provider=UniDAC and the other ConnectionString parameters for UniDAC devided by semicolon.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=UniDAC;SQL Dialect=SQLi
```

Available SQL Dialect: InterBase/Firebird, SQLite, MySQL, Oracle, PostgreSQL, SQL Server.

UniDAC Connection String: Connection String is similar to the one used in UniDAC

More details about ConnectionString for UniDAC can be found at http://www.devart.com/UniDAC/docs/devart.dac.tcustomdaconnection.connectstring.htm

TODACDataProvider

For TODACDataProvider set Data Provider=ODAC and the other ConnectionString parameters for ODAC devided by semicolon. SQL Dialect can be not specified.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=ODAC;SQL Dialect=Oracle
```

ODAC Connection String: Connection String is similar to the one used in ODAC

More details about ConnectionString for ODAC can be found at http://www.devart.com/ODAC/docs/devart.dac.tcustomdaconnection.connectstring.htm

TSDACDataProvider

For TSDACDataProvider set Data Provider=SDAC and the other ConnectionString parameters for SDAC devided by semicolon. SQL Dialect can be not specified.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=SDAC;SQL Dialect=SQL Se
```

SDAC Connection String: Connection String is similar to the one used in SDAC

More details about ConnectionString for SDAC can be found at http://www.devart.com/SDAC/docs/devart.dac.tcustomdaconnection.connectstring.htm

TMyDACDataProvider

For TMyDACDataProvider set Data Provider=MyDAC and the other ConnectionString parameters for MyDAC devided by semicolon. SQL Dialect can be not specified.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=MyDAC;SQL Dialect=MySQL
```

MyDAC Connection String: Connection String is similar to the one used in MyDAC

More details about ConnectionString for MyDAC can be found at http://www.devart.com/MyDAC/docs/devart.dac.tcustomdaconnection.connectstring.htm

TIBDACDataProvider

For TIBDACDataProvider set Data Provider=IBDAC and the other ConnectionString parameters for IBDAC devided by semicolon. SQL Dialect can be not specified.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=IBDAC;SQL dialect=Inter
```

More details about ConnectionString for IBDAC can be found at http://www.devart.com/ibdac/docs/devart.dac.tcustomdaconnection.connectstring.htm

TPgDACDataProvider

For TPgDACDataProvider set Data Provider=PgDAC and the other ConnectionString parameters for PgDAC devided by semicolon. SQL Dialect can be not specified.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=PgDAC;SQL Dialect=Postg
```

PgDAC Connection String: Connection String is similar to the one used in PgDAC

More details about ConnectionString for PgDAC can be found at http://www.devart.com/PgDAC/docs/devart.dac.tcustomdaconnection.connectstring.htm

TLiteDACDataProvider

For TLiteDACDataProvider set Data Provider=LiteDAC and the other ConnectionString parameters for LiteDAC devided by semicolon. SQL Dialect can be not specified.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=LiteDAC;SQL Dialect=SQL
```

LiteDAC Connection String: Connection String is similar to the one used in LiteDAC

More details about ConnectionString for LiteDAC can be found at http://www.devart.com/litedac/docs/devart.dac.tcustomdaconnection.connectstring.htm

TADODataProvider

For TADODataProvider set Data Provider=ADO and the other ConnectionString parameters for ADO devided by semicolon.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=ADO;SQL Dialect=XXX;Log
```

Available SQL Dialect: InterBase/Firebird, SQLite, MySQL, Oracle, PostgreSQL, SQL Server.

ADO Connection String: Connection String is similar to the one used in ADO

TDBXDataProvider

For TDBXDataProvider set Data Provider=dbExpress and the other ConnectionString parameters for dbExpress devided by semicolon.

The Connection Driver/ConnectionDriver parameter points to the registered connection in Data Explorer and sets it in the TSQLConnection.ConnectionName property.

The DriverName, VendorLib, LibraryName, GetDriverFunc parameters also correspond to analogue class properties of the TSQLConnectionin class.

Also, parameters supported in TSQLConnection.Params can be used as parameters here.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=dbExpress;SQL Dialect=X
```

Available SQL Dialect: InterBase/Firebird, SQLite, MySQL, Oracle, PostgreSQL, SQL Server.

TIBXDataProvider

For TIBXDataProvider set Data Provider=IBX and the other ConnectionString parameters for IBX devided by semicolon. SQL Dialect can be not specified.

The DatabaseName/Database Name/Database parameter sets the TIBDatabase.DatabaseName; property value.

Also, parameters supported in TIBDatabase.Params can be used as parameters here.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=IBX;SQL Dialect=InterBa
```

TFireDACDataProvider

For TFireDACDataProvider set Data Provider=FireDAC and the other ConnectionString parameters for FireDAC devided by semicolon.

Sample:

```
EntityConnection1.ConnectionString := 'Data Provider=FireDAC;SQL Dialect=XXX
```

Available SQL Dialect: InterBase/Firebird, SQLite, MySQL, Oracle, PostgreSQL, SQL Server.

FireDAC Connection String: Connection String is similar to the one used in FireDAC

ConnectionDefName parameter is added. It defines the connection name added in the ConnectionDef list.

4.6 Database Management

When the application is created using the Code-first development approach, then any changes in the object model need to be reflected in the database structure. The TEntityConnection component provides functionality for creating and re-creating database objects based on used meta-model information.

Create database

To create the database structure, the CreateDatabase method is used. It automatically generates a DDL script based on the used meta-model, and executes the script that creates all needed database objects.

```
var
  Connection: TEntityConnection;
begin
  // create and initialize the connection
  // ...
  // create the database
  Connection.CreateDatabase(nil, [moCommitEachStatement]);
end;
```

There are a peculiarity of using this method, that should be considered: the method creates only those database objects, which are needed by the model: tables, primary and foreign keys. Any other objects that exist in the database before (triggers etc.), will not be recreated.

Drop database

To drop all the database objects used by the model, the DropDatabase method is used.

```
var
  Connection: TEntityConnection;
begin
  // create and initialize the connection
  // ...
  // drop the database
  Connection.DropDatabase(nil, [moCommitEachStatement]);
```

```
end;
```


Request Support DAC Forum Provide Feedback

4.7 Data Management

The basic unit of data that EntityDAC operates is "entity". Although entity is an ordinary Delphi class, it behavior is some different. In EntityDAC, all entities are managed by the data context, that performs entity creation, holds created entities in the cache for future use, performs entity loading and storing in the database, and carries about their destruction.

Create entity

The object model class hierarchy looks like the following.

The base class that implements entity functionality is TEntity. TEntity is an abstract class, it does not have any properties and methods. The TMappedEntity class extends the TEntity functionality and adds the data context interaction. The object model should consist of a TMappedEntity descendants, which have to implement the properties for storing entity data.

```
interface
type
  TEmp = class(TMappedEntity)
  private
    FEmpno: TIntegerAttribute;
    FEname: TStringAttribute;
  protected
    function GetEmpno: integer;
    procedure SetEmpno(const Value: integer);
    function GetEname: string;
    procedure SetEname(const Value: string);
  public
    constructor Create; overload; override;
    property Empno: integer read GetEmpno write SetEmpno;
    property Ename: string read GetEname write SetEname;
  end;
implementation
{ TEmp }
constructor TEmp.Create;
begin
  inherited Create(MetaModel['Emp']);
  FEmpno := TintegerAttribute.Create(Attributes,
    MetaModel['Emp'].MetaAttributes.Get('Empno'));
  FEname := TstringAttribute.Create(Attributes,
    MetaModel['Emp'].MetaAttributes.Get('Ename'));
end;
function TEmp.GetEmpno: Integer;
```

```
begin
  Result := FEmpno.Value;
end;
procedure TEmp.SetEmpno(const Value: Integer);
begin
  FEmpno.Value := Value;
end;
function TEmp.GetEname: String;
begin
  Result := FEname.Value;
end;
procedure TEmp.SetEname(const Value: String);
begin
  FEname.Value := Value;
end;
```

An entity instance can be created in the same way as the trivial class instance, using the constructor.

```
var
  Emp: TEmp;
begin
  // create new entity
  Emp := TEmp.Create;
end;
```

When creating an entity, all its properties are initialized with their default values. It can be possible to set the entity primary key value once when the entity is created. In this case, a primary key value can be specified as the constructor parameter.

```
var
  Emp: TEmp;
begin
  // create new entity with the specified primary key value
  Emp := TEmp.Create([1]);
end;
```

Also, an entity instance can be created using methods of the data context.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // create new entity with the specified primary key value
  Emp := Context.CreateEntity<TEmp>([1]);
end;
```

Attach entity

After an entity is created, the data context has to be notified of this entity, so that data context can place the entity into the cache and take on further functions to manage it. In order to

understand the "attach" mechanism, there is need to explain the entity caching. Every used entity is stored in the entity cache implemented by the data context. When it becomes necessary to reuse the same entity, there is no need to refer to the database again to reload entity data, the entity will be initialized from the cache. Data context checks the uniqueness of entities being cached and prohibits to store two entities with the same primary key value. Therefore, it would be impossible to place an entity to the cache automatically on create, because all newly created entities have the same default primary key value.

An entity can be attached to the data context using corresponding entity method.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // create new entity
  Emp := TEmp.Create;
  // set the entity primary key
  Emp.Empno.AsInteger := 1;
  // attach the entity
  Emp.Attach(Context);
end;
```

Or using the data context method.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // create new entity
  Emp := TEmp.Create;
  // set the entity primary key
  Emp.Empno.AsInteger := 2;
  // attach the entity
  Context.Attach(Emp);
end;
```

Exception is the situation, when the entity primary key obtains unique value immediately on entity creation, for example, when the key value is exactly known when creating or when using a key generator. In this case, it is possible to create already attached entity using the following data context method.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
```

```
  // ...
  // create attached entity with the specified primary key value
  Emp := Context.CreateAttachedEntity<TEmp>([1]);
end;
```

Since the entity is attached and placed into the cache, it must not be explicitly destroyed in the code, because it will be automatically destroyed by the data context. Otherwise, the "Invalid pointer operation" exception will be raised when the application closes.

Get entity

There are three main ways to get a single entity from the data context.

An entity instance can be obtained by its primary key value.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // get single entity by the primary key
  Emp := Context.GetEntity<TEmp>([1]);
end;
```

Or, an entity instance can be obtained by a condition. If more than one entity matched specified condition, the exception will be raised.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // get single entity by the condition
  Emp := Context.GetEntity<TEmp>('empno = 1');
end;
```

The last, more complex but the most multipurpose method is to obtain an entity by a LINQ query. As in the previous sample, if the query returns more than one entity, then the appropriate exception will be raised.

```
var
  Context: TEntityContext;
  Query: ILinqQueryable;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // create the query
  Query := Linq.From(Context['Emp'])
               .Where(Context['Emp']['Empno'] = 1)
```

```
                .Select;
  // get single entity by the query
  Emp := Context.GetEntity<TEmp>(Query);
end;
```

In all cases, obtained entity will be automatically attached to the data context, thus it must not be destroyed manually in the code.

Get entities

For holding a list of entities, EntityDAC provides special IEntityEnumerable interface, which is the IEnumerable descendant. It declares methods to iterate through the list and to access list items.

In the simplest case, a whole list of all entities of a given type can be obtained.

```
var
  Context: TEntityContext;
  List: IEntityEnumerable<TEmp>;
  Emp: TEmp;
  i: integer;
begin
  // create and initialize the data context
  // ...
  // get a whole list of TEmp entities
  List := Context.GetEntities<TEmp>;
end;
```

A simple condition can be specified to limit the list.

```
var
  Context: TEntityContext;
  List: IEntityEnumerable<TEmp>;
  Emp: TEmp;
  i: integer;
begin
  // create and initialize the data context
  // ...
  // get the list by the condition
  List := Context.GetEntities<TEmp>('empno > 1');
end;
```

A list can be obtained as the result of a LINQ query execution.

```
var
  Context: TEntityContext;
  Query: ILinqQueryable;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // create the query
  Query := Linq.From(Context['Emp'])
```

```
                .Where(Context['Emp']['Empno'] > 1)
                .Select;
  // get the list by the query
  Emp := Context.GetEntities<TEmp>(Query);
end;
```

After obtaining the list, each entity can be accessed with its index.

```
for i := 0 to List.Count - 1 do begin
  Emp := List[i];
  // do something
  // ...
end;
```

In Delphi 2010 and higher, it also possible to use the "for ... in ..." statement to iterate through the list.

```
for Emp in List do begin
  // do something
  // ...
end;
```

Save entity

Modifying entity in the code does not affect corresponding database objects. To reflect in the database all changes made for the entity, it has to be saved.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // creation and initialization of the context
  // ...
  Emp := TEmp.Create;
  // set the unique value to the entity primary key
  Emp.Empno.AsInteger := 1;
  // attach the entity
  Emp.Attach(Context);
  // save the entity in the database
  Emp.Save;
end;
```

Or using the data context method.

```
var
  Context: TEntityContext;
  Emp: TEmp ;
begin
  // creation and initialization of the context
  // ...
  Emp := TEmp.Create;
  // set the unique value to the entity primary key
  Emp.Empno.AsInteger := 1;
  // attach the entity
  Context.Attach(Emp);
```

```
  // save the entity in the database
  Context.Save(Emp);
end;
```

Commonly, performing attach before saving an entity is not required because the Save method implicitly calls Attach. However, if an error occurs, it can be difficult to determine at what stage it occurs (when attaching or saving).

Delete entity

Deletion of an entity is a two-phase process in EntityDAC. Since all entities are stored in the cache, the entity first needs to be deleted from it. Then, to apply the deletion in the database, the entity has to be saved.

```
var
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // get single entity by the primary key
  Emp := Context.GetEntity<TEmp>([1]);
  // delete entity from the cache
  Emp.Delete;
  // apply deletion in the database
  Emp.Save;
end;
```

Or using the data context method.

```
var
  Context: TEntityContext;
  Emp: TEmp ;
begin
  // create and initialize the data context
  // ...
  // get single entity by the primary key
  Emp := Context.GetEntity<TEmp>([1]);
  // delete entity from the cache
  Context.Delete(Emp);
  // apply deletion in the database
  Context.Save(Emp);
end;
```

Cancel entity changes

As it was described above, all modification operations with an entity (changing, deleting) have to be confirmed (saved) to reflect in the database. Therefore, until the object has not been saved it is possible to cancel it changes.

```
var
```

```
  Context: TEntityContext;
  Emp: TEmp;
begin
  // create and initialize the data context
  // ...
  // get single entity by the primary key
  Emp := Context.GetEntity<TEmp>([1]);
  // change the entity property
  Emp.Ename.AsString := 'new name';
  // cancel changes
  Emp.Cancel;
end;
```

Or using the data context method.

```
var
  Context: TEntityContext;
  Emp: TEmp ;
begin
  // create and initialize the data context
  // ...
  // get single entity by the primary key
  Emp := Context.GetEntity<TEmp>([1]);
  // change the entity property
  Emp.Ename.AsString := 'new name';
  // cancel changes
  Context.Cancel(Emp);
end;
```

Submit changes

When creating applications there is a quite common situation where many objects have to be saved simultaneously, and perform save for each of them is not suitable (for example, when several related objects should be stored at the same time at the completion of a dialog form). In this case, the data context has the special SubmitChanges method for applying massive changes.

```
var
  Context: TEntityContext;
begin
  // create and initialize the data context
  // ...
  // making changes to entities
  // ...
  // submit all changes
  Context.SubmitChanges;
end;
```

Executing the method is the same as the coherent execution of the Save method for each of the modified entities.

Reject changes

And in contrast to the previous method, the RejectChanges method performs opposite action. It simultaneously cancels all changes made to entities.

```
var
  Context: TEntityContext;
begin
  // create and initialize the data context
  // ...
  // making changes to entities
  // ...
  // reject all changes
  Context.RejectChanges;
end;
```

Executing the method is the same as the coherent execution of the Cancel method for each of the modified entities.

Request Support DAC Forum Provide Feedback

4.8 Entity Behavior Customization

Generated Entity Customization

The default approach to change the behavior of an existing class in Delphi is to use inheritance. However, when working with EntityDAC, inheritance can not always be applicable. In EntityDAC, to be able to use any entity class, its mapping has to be defined. Therefore, when creating new entity class descendant, you also need to define the corresponding meta-type and define the mapping for it. Such an approach is not always rational, because for minor changes in the descendant, it is necessary to write a lot of additional code.

Of course, behavior of any class can simply be changed by manually changing its code. But, using classes generated with Entity Developer, all previously made changes will be lost when the database model changes, and the classes code is completely re-generated.

Class Helpers

In such a situation, when it is need to extend the behavior of an existing class without modifying its code, the default approach is to use class helpers. You can learn http://

docwiki.embarcadero.com/RADStudio/XE5/en/Class_and_Record_Helpers for more information about class helpers in Delphi.

```
type
  TEntityHelper = class helper for TEntity
    ...
  end;
```

However, class helpers usage has several restrictions. For example, class helpers can not contain any instance data, class helpers can not override virtual methods etc.

Entity Customization

EntityDAC provides its own mechanism for customizing generated entity classes, which eliminates the problem with classes re-generation and does not have disadvantages of class helpers. The main idea is to implement a class descendant with some extended functionality and "redirect" existing class mapping to the descendant class without defining new mapping for it.

Let's look at an example of the entity customization. Assume that we have the TCustomEntity class declared like shown below:

```
TCustomEntity = class(TMappedEntity)
private
  FValue: TIntegerAttribute;
  function GetSomeValue: integer;
protected
  procedure SetSomeValue(const Value: integer); virtual;
public
  property SomeValue: integer read GetSomeValue write SetSomeValue;
end;
```

Also, we have the corresponding meta-type declaration for the class, and corresponding class mapping is defined. The meta-type has the 'CustomEntity' name.

Now, suppose that we want to implement some additional logic when the Value property changes. The simplest way is to write corresponding code in the SetValue implementation. But, the class code is generated automatically by Entity Developer and will be completely re-generated when the database model changes.

So, let's declare the TCustomEntity class descendant and write desired code in the overridden SetValue method:

```
interface
type
  TExtendedEntity = class(TCustomEntity)
  protected
```

```
    procedure SetSomeValue(const Value: integer); override;
  end;
implementation
procedure TExtendedEntity.SetSomeValue(const Value: integer);
var
  NewValue: integer;
begin
  NewValue := Value + 100;
  inherited SetSomeValue(NewValue);
end;
```

Now, we have to "redirect" the TCustomEntity class mapping to the newly declared class. To do this, we use the RegisterEntityClass method of the 'CustomEntity' meta-type. Insert the following line somewhere in the application code, before first usage of the TCustomEntity class:

```
Context.Model['CustomEntity'].RegisterEntityClass(TExtendedEntity);
```

Where Context is the TEntityContext instance used in the application.

From now, TExtendedEntity class is mapped to the previously declared 'CustomEntity' meta-type and there is no need to declare and map a separate meta-type for it. We can test our customized entity behavior:

```
var
  Entity: TExtendedEntity;
begin
  Context.Model['CustomEntity'].RegisterEntityClass(TExtendedEntity);
  Entity := Context.CreateEntity<TExtendedEntity>;
  Entity.SomeValue := 1;
  Context.Attach(Entity);
  ShowMessage(IntToStr(Entity.SomeValue)); // will show '101'
end;
```

The feature is available in EntityDAC Professional and Standard editions. It is not available in EntityDAC Express Edition.

Request Support DAC Forum Provide Feedback

4.9 Memory Management

In EntityDAC, all used entities are managed by the data context. It means that the data context stores every created entity or entities loaded from the database in its internal cache. Accordingly, the data context cares about automatic destroying the entities.

Attaching the entity

In order to the data context became aware about the entity and took on further management functions, the entity should be attached to the data context.

Single entity

A manually created entity can be attached either with its Attach method, or using the Attach method of the data context. An entity which is loaded from the database using the GetEntity data context method, is initially attached. When the entity is saved to the database using the data context Save method, it also automatically become attached.

An attached entity must not be destroyed manually, otherwise the exception is raised.

```
var
  Emp,
  Emp1,
  Emp2: TEmp;
begin
  // the entity is not attached, and it has to be destroyed manually
  Emp := TEmp.Create;
  Emp.Free;
  // the entity is loaded from the database and is initially attached
  // manual destroying is not needed
  Emp1 := Context.GetEntity<TEmp>(1);
  // the entity is loaded from the database and is attached on Save
  // manual destroying is not needed
  Emp2 := Context.CreateEntity<TEmp>;
  Emp2.Save;
end;
```

Entity list

In EntityDAC, any list object which is the result of the data context GetEntities method execution, is the TInterfacedObject descendant. Therefore, there is no need to care about its destroying. Also, all entities which constitute the list, are initially attached to the data context and managed by it.

```
var
  Emps: IEntityEnumerable<TEmp>;
begin
  // its no need to manually destroy the list instance and its items
  Emps := Context.GetEntities<TEmp>;
end;
```

LINQ query

Furthermore, EntityDAC implements the LINQ syntax using special helper classes which are also inherited from TInterfacedObject, so their manual destroying is also not required.

```
var
  Query: ILinqQueryable;
  Emps: IEntityEnumerable<TEmp>;
begin
  // the query expression instance will be destroyed automatically
  Query := Linq.From('Emp').Where('Emp.DeptNo = 1').Select;

  Emps := Context.GetEntities<TEmp>(Query);
end;
```


Request Support DAC Forum Provide Feedback

4.10 SQL Executing

Although EntityDAC is an ORM framework and eliminates the need for direct access to database data, for cases where it is still necessary there are several methods for direct interaction with the database, which are implemented in TEntityConnection.

Transaction management

Since TEntityConnection provides methods to access the database, certain functionality for transaction management is present. The StartTransaction, CommitTransaction and RollbackTransaction methods allow to control transactions.

```
var
  Connection: TEntityConnection;
begin
  // create and initialize the connection
  // ...
  // begin the transaction
  Connection.StartTransaction;
  try
    // ...
    // commit the transaction
    Connection.CommitTransaction;
  except
    // rollback the transaction in case of an error
    Connection.RollbackTransaction;
  end;
end;
```

SQL query

The ExecuteSQL method allows to easily execute a SQL statement, which does not return data.

```
var
```

```
  Connection: TEntityConnection;
begin
  // create and initialize the connection
  // ...
  // execute the simple SQL statement
  Connection.ExecuteSQL('insert into EMP(ENAME) values(''Sample'')');
end;
```

In more complex case, the SQL statement can be parametrized (for example when there is need to return parameters as the result of a SQL statement execution).

```
// add necessary units to be able to use the TDBParams class
uses
  SQLDialect, EntityTypes;
var
  Connection: TEntityConnection;
  Params: TDBParams;
begin
  // create and initialize the connection
  // ...
  // create and fill query parameters
  Params := TDBParams.Create;
  try
    with Params.Add do begin
      Name := 'ret_param';
      ParamType := ptOutput;
      DataType := dbInteger;
    end;
    Connection.ExecuteSQL ('insert into EMP(ENAME) values(''Sample'') return
    ShowMessage('Return = ' + Params[0].Value.AsString);
  finally
    Params.Free;
  end;
end;
```

Stored procedure

A stored procedure can be executed using the ExecuteStoredProc method. The procedure parameters are passed as when executing of the parametrized query.

```
uses
  SQLDialect, EntityTypes;
var
  Connection: TEntityConnection;
  Params: TDBParams;
begin
  // create and initialize the connection
  // ...
  // create and fill procedure parameters
  Params := TDBParams.Create;
  try
    with Params.Add do begin
      Name := 'proc_param';
      ParamType := ptInput;
      DataType := dbInteger;
```

```
    end;
    Connection.ExecuteStoredProc('some_procedure', Params);
  finally
    Params.Free;
  end;
end;
```

SQL script

For executing a number of sequential SQL statements, the ExecuteScript method is used. The SQL script can be passes either as the string parameter or as TStrings. The script cannot be parametrized.

```
var
  Connection: TEntityConnection;
  Script: TStrings;
begin
  // create and initialize the connection
  // …
  // create and fill the script
  Script := TStringList.Create;
  try
    Script.Add('insert into EMP(ENAME) values(''Sample'');');
    Script.Add('insert into EMP(ENAME) values(''Sample 1'');');
    Connection.ExecuteScript(Script);
  finally
    Script.Free;
  end;
end;
```


Request Support DAC Forum Provide Feedback

5 LINQ Queries

- Range Variables, References & Collections
  - Declaration and initialization of a range variable that defines an initial sequence for a LINQ query.
  - Accessing entity attributes in a query.
  - Accessing a linked entity and its attributes.
  - Accessing a linked entity collection and attributes of a separate entity in the collection.
- LINQ Query Syntax
  - Query clauses

- From
- Join, LeftJoin, RightJoin, FullJoin, On
- Where
- GroupBy
- OrderBy, OrderByDescending, ThenBy, ThenByDescending
- Select
- Result fields naming. The As clause
- Union, Concat, Except, Intersect
- Skip, Take, ElementAt, First
- Any, All
- Count, Max, Min, Average
- Contains

- Specifying LINQ Query Arguments As String

Request Support DAC Forum Provide Feedback

5.1 Range Variables, References And Collections

Declaration and initialization of a range variable that defines an initial sequence for a LINQ query

Declaration:

```
var
  range-variable: IMetaDataExpression;
```

Initialization:

```
meta-type-name = string value

range-variable := data-context.Types[meta-type-name]
| range-variable := data-context[meta-type-name]
```

Sample:

```
  var
    Emp,
    Dept: IMetaDataExpression;
```

```
  begin
    Emp  := DataContext.Types['Emp'];
    Dept := DataContext['Dept'];
  end;
```

Accessing entity attributes in a query

```
attribute-name = string value

entity-attribute ::=
    range-variable.Attribute[attribute-name]
  | range-variable[attribute-name]
```

Sample:

```
  Emp.Attribute['EmpNo']
  Dept['DeptNo']
```

Accessing a linked entity and its attributes

```
reference-name = string value

reference ::=
    range-variable.Reference[reference-name]

reference-attribute ::=
    reference.Attribute[attribute-name]
  | reference[attribute-name]
```

Sample:

```
  Emp.Reference['Dept']
  Emp.Reference['Dept'].Attribute['DeptNo']
  Emp.Reference['Dept']['DeptNo']
```

Accessing a linked entity collection and attributes of a separate entity in the collection

```
collection-name = string value

collection ::=
  range-variable.Collection[collection-name]
collection-entity-attribute ::=
    collection.Rowtype.Attribute[attribute-name]
  | collection.Rowtype[attribute-name]
```

Sample:

```
  Dept.Collection['Emps']
  Dept.Collection['Emps'].Rowtype.Attribute['DeptNo']
  Dept.Collection['Emps'].Rowtype['DeptNo']
```

5.2 LINQ Query Syntax

Common query syntax.

```
count = integer value
index = integer value

query ::=
  query-result
    [ {    .Union(query)
         |  .Concat(query)
         |  .Except(query)
         |  .Intersect(query) } ]

query-result ::=
  query-sequence
  [ ( (  .Skip(count)
         .Take(count) )
       |  .ElementAt(index)
       |  .First ) ]
  |
  [ (    .Any [ (condition) ]
       |  .All(condition) ) ]
  |
  [ (    .Count
       |  .Max
       |  .Min
       |  .Sum
       |  .Average
       |  .Distinct ) ]

query-sequence ::=
  [Linq.]
      From(range-variable)
  [ { From(range-variable) } ]
  [ {    .Join(range-variable) [ .On(condition) ]
       |  .LeftJoin(range-variable).On(condition)
       |  .RightJoin(range-variable).On(condition)
       |  .FullJoin(range-variable).On(condition) } ]
  [   .Where(condition) ]
  [ { .GroupBy(group-expression) } ]
  [ { .OrderBy(order-expression) } ]
  [ { .OrderByDescending(order-expression) } ]
  [ { .ThenBy(order-expression) } ]
  [ { .ThenByDescending(order-expression) } ]
      .Select [ (select-expression) ]
```

Query clause arguments.

```
Pattern = string value

attribute ::=
    entity-attribute
  | reference-attribute
  | collection-entity-attribute

operand ::=
    attribute
  | string value
  | number value
  | (expression)
  | null (* only as a right operand in "=" and "<>" expressions *)

expression ::=
    -operand
  | operand + operand
  | operand - operand
  | operand * operand
  | operand / operand
  | operand mod operand
  | operand div operand
  | operand shl operand
  | operand shr operand
  | operand = operand
  | operand <> operand
  | operand > operand
  | operand < operand
  | operand >= operand
  | operand <= operand
  | operand or operand
  | operand and operand
  | operand xor operand
  | operand not operand
  | trunc(operand)
  | round(operand)
  | attribute.Contains(pattern)

condition = expression
group-expression = expression

order-expression ::=
    expression
  | [expression, {expression}]

select-expression ::=
    expression
  | [expression, {expression}]
```

Query clauses.

From

```
[Linq.]From(range-variable) [ .From(range-variable) ]
```

The clause defines the data source for the further query constructing and introduces a range variable, that should be used for making further clauses arguments.

From is a required clause. Every query statement begins with a From clause. The initial From clause can be followed by zero or more From clauses for combining multiple data sources. In this case, the sources are combined with CROSS JOIN.

The required clause argument is a previously declared and initialized range variable (see. Range variables, references and collections).

Sample:

```
  Linq.From(Emp)
  Linq.From(Emp).From(Dept)
```

Join, LeftJoin, RightJoin, FullJoin, On

```
[ {    .Join(range-variable) [ .On(condition) ]
    |  .LeftJoin(range-variable).On(condition)
    |  .RightJoin(range-variable).On(condition)
    |  .FullJoin(range-variable).On(condition) } ]
```

Join clauses correlate new data source with the source of the preceding clause. The *On* clause determines the matching criterion of elements of both sources.

A join clause is an optional clause. It can be placed either after a *From* clause or after other join clause. Using *On* with *Join* is optional, whereas using it with *LeftJoin*, *RightJoin*, *FullJoin* is required.

Join types created by clauses:

- Join: CROSS JOIN (if On is not used), INNER JOIN (if On is used);
- LeftJoin: LEFT OUTER JOIN;

- RightJoin: RIGHT OUTER JOIN;
- FullJoin: FULL OUTER JOIN.

The required join clause argument is a previously declared and initialized range variable (see. Range variables, references and collections).

The required *On* clause argument is a logical expression that defines the correspondence between joined sources (like "equals" in LINQ).

Sample:

```
Linq.From(Emp)
    .Join(Dept)
Linq.From(Emp)
    .Join(Dept).On(Emp['DeptNo'] = Dept['DeptNo'])
Linq.From(Emp)
    .LeftJoin(Dept).On(Emp['DeptNo'] = Dept['DeptNo'])
```

Where

```
.Where(condition)
```

The clause defines a filter that excludes items from the result of the preceding clauses.

Where is an optional clause. It can be placed immediately after a source clause (*From* or *Join*).

The required *Where* clause argument is a logical expression that defines the condition which each of result elements must conform.

Sample:

```
Linq.From(Emp)
    .Where(Emp['Sal'] > 1000)
```

GroupBy

```
[ { .GroupBy(group-expression) } ]
```

The clause groups the elements of a result of the preceding clauses according to a specified expression.

GroupBy is an optional clause. It can be placed after a source clause (*From* or *Join*) or after a Where clause The required *GroupBy* clause argument is an expression that defines the grouping criteria.

Sample:

```
Linq.From(Emp)
    .GroupBy(Emp['DeptNo'])
```

OrderBy, OrderByDescending, ThenBy, ThenByDescending

```
[ { .OrderBy(order-expression) } ]
[ { .OrderByDescending(order-expression) } ]
[ { .ThenBy(order-expression) } ]
[ { .ThenByDescending(order-expression) } ]
```

The clauses specify an ordering for the result sequence.

An order clause is an optional clause. It can be placed after a source clause (*From* or *Join*) or after a limiting clause (*Where* or *GroupBy*).

OrderBy sets an ascending sorting of the result elements, whereas *OrderByDescending* sets a descending sorting. *ThenBy* and *ThenByDescending* clauses are complete analogues of *OrderBy* and *OrderByDescending* accordingly, and introduced only for compatibility with classical IEnumerable extension methods. To allow multiple sort criteria, any number of order operators can be applied after each other.

The required order clause argument is an expression that defines a sort criteria of result elements. The argument also can be specified as an array of expressions to allow multiple sort criteria.

Sample:

```
Linq.From(Emp)
    .OrderBy(Emp['DeptNo'])
    .ThenBy([Emp['Sal'], Emp['Comm']])
```

Select

```
.Select [ (select-expression) ]
```

The clauses populates members of the query result sequence.

Select is a required clause. Every query statement must contain a *Select* clause.

The optional *Select* clause argument is an expression or array of expressions that specifies the data fields of the result sequence element. If no argument specified, the result element consists of all data fields of all query sources specified in *From* and join clauses.

Sample:

```
Linq.From(Emp)
    .Select
Linq.From(Emp)
```

```
    .Select(Emp['EName'])
Linq.From(Emp)
    .Select([Emp['EName'], Emp.Reference['Dept']['DName']])
```

Result fields naming. The As clause.

```
field-name = string value
attribute.As(field-name)
TExpression(operand).As(field-name)
```

The clause defines the name of the result element field.

The *As* clause is not a LINQ query clause itself, but it is closely related to the Select clause, as it allows to control names of the result sequence fields.

Commonly, if no explicit name for a result field is specified, the field obtains its name automatically corresponding to the following rules:

- the field that contains an entity attribute obtains the name of the attribute;
- each subsequent field that contains an entity attribute with the same name will obtain the name of the attribute with the numeric suffix, for example: "Id", "Id1", "Id2", etc.;
- fields that contain a complex expression like a mathematical operation or an aggregate function obtain the "C" name with the sequential numeric suffix, for example: "C1", "C2", "C3" etc.

In order to specify the field name manually, the *As* clause has to be used. The use of As can be implemented in two ways:

- for entity attributes, reference attributes and collection entity attributes, As can be specified directly as the method of the attribute;
- for complex expressions, the *TExpression.As* class method has to be used.

Sample:

```
Linq.From(Emp)
    .Select([Emp['EName'].As('Name'),
    TExpression(Emp['EmpNo'] + 1).As('Id')])
```

Union, Concat, Except, Intersect

```
[ {    .Union(query)
    |  .Concat(query)
    |  .Except(query)
    |  .Intersect(query) } ]
```

The clauses combines the result sequence of the query with a result sequence of another query.

A union clause is an optional clause. It can be placed immediately after a *Select* clause or after any of "post-Select" clauses (aggregates, partitioning methods or quantifiers).

Union combines two sequences and excludes duplicates from the return set. *Concat* returns all the elements in the input sequences including duplicates. *Except* returns those elements in the main sequence that do not appear in the combined sequence. It does not also return those elements in the combined sequence that do not appear in the main sequence. *Intersect* returns a set that contains all the elements of the main sequence that also appear in the combined sequence, but no other elements.

The required clause argument must be a complete query statement that returns the combined sequence with the same result fields count and field types as the main sequence.

Sample:

```
Linq.From(Emp)
    .Select(Emp['EmpNo'])
    .Union(
DataContext.From(Dept)
    .Select(Dept['DeptNo'])
    )
```

Skip, Take, ElementAt, First

```
[ ( (  .Skip(count)
       .Take(count) )
    |  .ElementAt(index)
    |  .First ) ]
```

Partitioning clauses are used to extract a number of elements from the query result sequence.

Skip clause bypasses a specified number of elements in an input sequence and then returns the remaining elements. *Skip* can be applied on the result of the *Select* or *Union* clause. *Take* returns a specified number of contiguous elements from the input sequence. *Take* can be used alone after the *Select* or *Union* clause, and in this case it will return a number of elements form the start of a sequence. But, usually, the *Skip* and *Take* methods are functional complements and used together in order to perform result sequence pagination. In this case, *Take* has to be applied after *Skip*. *ElementAt* is used to obtain the sequence element at the specified index. *First* returns the first element in the specified sequence.

The required *Skip* argument is a number of elements that have to be bypassed. The required *Take* argument is a number of elements that have to be returned. The required *ElementAt* argument is a zero-based index of the element that has to be returned.

Sample:

```
Linq.From(Emp)
    .Select([Emp['EmpNo'], Emp['EName']])
    .Skip(10)
Linq.From(Emp)
    .Select([Emp['EmpNo'], Emp['EName']])
    .Take(5)
Linq.From(Emp)
    .Select([Emp['EmpNo'], Emp['EName']])
    .Skip(10)
    .Take(5)
Linq.From(Emp)
    .Select([Emp['EmpNo'], Emp['EName']])
    .ElementAt(5)
Linq.From(Emp)
    .Select([Emp['EmpNo'], Emp['EName']])
    .First
```

Any, All

```
[ (    .Any [ (condition) ]
   |   .All(condition) ) ]
```

Quantifiers are used to check the conformity of result sequence elements to a certain condition.

Any clause (if used without argument) returns *True* if the input sequence contains any elements, otherwise *False*. When *Any* is called with an argument, it determines whether any element of a sequence satisfies the specified condition and returns *True* in this case, otherwise False. *All* clause returns True if every element of the input sequence satisfies the specified condition, or if the sequence is empty. Otherwise it returns False.

The optional *Any* argument and required All argument is a logical expression that defines the condition which sequence elements must conform.

Sample:

```
Linq.From(Emp)
    .Select([Emp['EmpNo'], Emp['EName']])
    .Any
Linq.From(Emp)
    .Select([Emp['EmpNo'], Emp['EName']])
    .Any(Emp['Sal'] = 1000)
Linq.From(Emp)
```

```
    .Select([Emp['EmpNo'], Emp['EName']])
    .All(Emp['Sal'] >= 1000)
```

Count, Max, Min, Sum, Average

```
  [ (    .Count
       |  .Max
       |  .Min
       |  .Sum
       |  .Average ) ]
```

Count returns the number of elements in the input sequence. *Max* returns the maximum value in a sequence. *Min* returns the minimum value in a sequence. *Sum* computes the sum of a sequence values. *Average* computes the average of a sequence values.

None of aggregate clauses has arguments.

Sample:

```
Linq.From(Emp)
    .Select(Emp['EmpNo'])
    .Count
Linq.From(Emp)
    .Select(Emp['Sal'])
    .Max
Linq.From(Emp)
    .Select(Emp['Sal'])
    .Min
Linq.From(Emp)
    .Select(Emp['Sal'])
    .Sum
Linq.From(Emp)
    .Select(Emp['Sal'])
    .Average
```

Contains

```
attribute.Contains(pattern | query | [expression, {expression}])
```

Contains is used as an expression of the *Where* clause and defines the condition, which each of result elements must conform.

The argument of *Contains* can be a search pattern, a query or a set of expressions.

When the argument is a search pattern, *Contains* checks the corresponding attribute of each result element for conformity with this pattern. In this case, *Contains* will be thanslated into the LIKE SQL clause, so the search pattern can contain wildcards applicable in the LIKE clause.

When the argument is a set of expressions or a query, *Contains* checks whether the corresponding attribute of each result element is included in the specified set or a set

returned by the query. In this case, *Contains* will be thanslated into the IN SQL clause.

Sample:

```
Linq.From(Emp)
    .Where(Emp['EName'].Contains('%dams'))
    .Select
Linq.From(Emp)
    .Where(Emp['DeptNo'].Contains(Linq.From(Dept).Select([Dept['DeptNo']]).E
    .Select
Linq.From(Dept)
    .Where(Dept['DeptNo'].Contains([1, 2, 3]).Expression))
    .Select
```


Request Support DAC Forum Provide Feedback

5.3 Specifying LINQ Query Arguments As String

There is another way to specify LINQ query arguments instead of using range variables and expressions.

In From and Join clauses that define the data source for the further query, the argument can be specified as a corresponding meta type name instead of a range variable.

Sample:

```
Linq.From('Emp')
    .Join('Dept')
```

In all clauses that require conditional arguments (On, Where, Any, All), the argument can be specified as a string containing Delphi-style conditional statement. In this case, every string constant inside the argument has to be additionally quoted. Otherwise, it will be treated as an meta-data identifier.

Sample:

```
Linq.From('Emp')
    .Join('Dept').On('Dept.DeptNo = Emp.DeptNo')
    .Where('(Emp.EmpNo > 100) and (Emp.EName <> ''Smith'')')
```

In clauses that accept either single or array-type arguments (GroupBy, OrderBy, ThenBy, Select), the string argument has to be specified using following rules.

Every string constant inside the argument has to be additionally quoted. Otherwise, it is treated as a meta-data identifier.

Sample:

```
Linq.From('Emp')
    .Select('''something''')
// will select the 'something' string

Linq.From('Emp')
    .Select('something')
// will try to select a list of the unknown 'something' meta type
// and raise the exception

Linq.From('Emp')
    .Select('Emp.EName')
// will select the list of Emp.EName values
```

If the string operand has to be specified as an array, the array brackets must be within the string. Otherwise, the argument is treated as an array with a single string element.

Sample:

```
Linq.From('Emp')
    .Select('[Emp.EmpNo, Emp.EName]')
// will select the list of Emp.EmpNo and Emp.EName values

Linq.From('Emp')
    .Select(['Emp.EmpNo, Emp.EName'])
// will try to select a list of 'Emp.EmpNo, Emp.EName' meta-data values
// and raise the exception
```


Request Support DAC Forum Provide Feedback

6 Reference

This page shortly describes units that exist in EntityDAC.

Units

Unit Name	Description
EntityDAC.ComplexData	The unit contains implementation of classes that provides BLOB, TEXT and XML data handling.
EntityDAC.Context	The unit contains implementation of base classes that provides data context functionality.
EntityDAC.DataProvider	Contains implementation of Data Provider functionality

EntityDAC.DataProvider.ADO	Links the data provider for ADO to an application.
EntityDAC.DataProvider.DBX	Links the data provider for dbExpress to an application.
EntityDAC.DataProvider.FireDAC	Links the data provider for FireDAC to an application.
EntityDAC.DataProvider.IBDAC	Links the data provider for Devart InterBase Data Access Components to an application.
EntityDAC.DataProvider.IBX	Links the data provider for IBX data access components to an application.
EntityDAC.DataProvider.LiteDAC	Links the data provider for Devart SQLite Data Access Components to an application.
EntityDAC.DataProvider.MyDAC	Links the data provider for Devart MySQL Data Access Components to an application.
EntityDAC.DataProvider.ODAC	Links the data provider for Devart Oracle Data Access Components to an application.
EntityDAC.DataProvider.PgDAC	Links the data provider for Devart PostgreSQL Data Access Components to an application.
EntityDAC.DataProvider.SDAC	Links the data provider for Devart SQL Server Data Access Components to an application.
EntityDAC.DataProvider.UniDAC	Links the data provider for Devart Universal Data Access Components to an application.
EntityDAC.Entity	The unit contains implementation of the entity management.
EntityDAC.EntityAttributes	List of TEntityAttribute.
EntityDAC.EntityConnection	The unit contains implementation of the

	connection component functionality.
EntityDAC.EntityContext	This unit contains implementation of objects lifecycle management.
EntityDAC.EntityDataSet	The unit contains implementation of the entity dataset functionality.
EntityDAC.EntityXMLModel	The unit contains implementation of the XML mapping.
EntityDAC.Enumerable	The unit contains implementation of the entity enumeration.
EntityDAC.Linq	The unit contains implementation of the LINQ query syntax.
EntityDAC.MetaData	The unit contains implementation of the metadata management.
EntityDAC.NullableTypes	This unit contains implementation of nullable types management.
EntityDAC.ObjectContext	The unit contains implementation of the data context functionality.
EntityDAC.SQLDialect	The base unit that contains information about SQL implementations for various DBMS.
EntityDAC.Values	The unit contains implementation of classes that allow storing of any data.

Request Support DAC Forum Provide Feedback

6.1 EntityDAC.Context

The unit contains implementation of base classes that provides data context functionality.

Classes

Name	Description
TCacheOptions	Allows to control entity caching.
TContextOptions	Used to specify the behavior of the TEntityContext class.
TCustomCollection<T>	The base class which declares functionality for entity collections.
TCustomContext	The base class that provides the data context functionality.
TDataContext	The class provides functions for data context management.
TEntityCollection<T>	Provides functionality for handling a collection of entities.
TObjectCollection<T>	Provides functionality for handling a collection of entities which are not TEntity descendants.

Interfaces

Name	Description
ICustomCollection<T>	The base interface which declares functionality for entity collections.

Types

Name	Description
TCollectionOptions	Allows to control behavior of linked entity collections. The property is a set of TCollectionOption flags.

Enumerations

Name	Description
TCollectionOption	A set of TCollectionOptions

6.1.1 Classes

Classes in the **EntityDAC.Context** unit.

Classes

Name	Description
TCacheOptions	Allows to control entity caching.
TContextOptions	Used to specify the behavior of the TEntityContext class.
TCustomCollection<T>	The base class which declares functionality for entity collections.
TCustomContext	The base class that provides the data context functionality.
TDataContext	The class provides functions for data context management.
TEntityCollection<T>	Provides functionality for handling a collection of entities.
TObjectCollection<T>	Provides functionality for handling a collection of entities which are not TEntity descendants.

6.1.1.1 TCacheOptions Class

Allows to control entity caching.

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

Unit

`EntityDAC.Context`

Syntax

```
TCacheOptions = class(TPersistent);
```

Remarks

The mechanism of entity cachingis described in details in the "Attach entity" section of the Using EntityDAC -> Data Management article.

Request Support DAC Forum Provide Feedback

6.1.1.1.1 Members

TCacheOptions class overview.

Properties

Name	Description
Enabled	Allows to enable or disable entity caching.

Request Support DAC Forum Provide Feedback

6.1.1.1.2 Properties

Properties of the **TCacheOptions** class.

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

Published

Name	Description
Enabled	Allows to enable or disable entity caching.

See Also

- TCacheOptions Class
- TCacheOptions Class Members

Request Support DAC Forum Provide Feedback

Reserved.

6.1.1.1.2.1 Enabled Property

Allows to enable or disable entity caching.

Class

TCacheOptions

Syntax

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

Remarks

If Enabled is set to True, then entity caching is enabled, and on attempt to retrieve an entity (entity list) for the second time using GetEntity or GetEntities methods, the corresponding entities will be taken from the cache.

If set to False, then caching is disabled, and using GetEntity or GetEntities methods will return entities from the database every time.

True by default.

See Also

- TMetaType.AllowCaching

6.1.1.2 TContextOptions Class

Used to specify the behavior of the TEntityContext class.

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

Unit

EntityDAC.Context

Syntax

```
TContextOptions = class(TPersistent);
```

Remarks

Set the properties of Options to specify the behavior of the TEntityContext class.

See Also

- TEntityContext

 Request Support DAC Forum Provide Feedback

6.1.1.2.1 Members

TContextOptions class overview.

Properties

Name	Description
Cache	Allows to enable or disable entity caching.
CollectionOptions	Allows to control behavior of linked entity collections. The property is a set of TCollectionOption flags.

 Request Support DAC Forum Provide Feedback

6.1.1.2.2 Properties

Properties of the **TContextOptions** class.

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

Published

Name	Description
Cache	Allows to enable or disable entity caching.
CollectionOptions	Allows to control behavior of linked entity collections. The property is a set of

	TCollectionOption flags.

See Also

- TContextOptions Class
- TContextOptions Class Members

Request Support DAC Forum Provide Feedback

6.1.1.2.2.1 Cache Property

Allows to enable or disable entity caching.

Class

TContextOptions

Syntax

```
property Cache: TCacheOptions;
```

Remarks

If Enabled is set to True, then entity caching is enabled, and on attempt to retrieve an entity (entity list) for the second time using GetEntity or GetEntities methods, the corresponding entities will be taken from the cache.

If set to False, then caching is disabled, and using GetEntity or GetEntities methods will return entities from the database every time.

True by default.

See Also

- TMetaType.AllowCaching

Request Support DAC Forum Provide Feedback

6.1.1.2.2.2 CollectionOptions Property

Allows to control behavior of linked entity collections. The property is a set of TCollectionOption flags.

Class

TContextOptions

Syntax

```
property CollectionOptions: TCollectionOptions default
[coKeyOrdered];
```

Remarks

The defulat value is coKeyOrdered

Request Support DAC Forum Provide Feedback

6.1.1.3 TCustomCollection<T> Class

The base class which declares functionality for entity collections.

For a list of all members of this type, see TCustomCollection<T> members.

Unit

EntityDAC.Context

Syntax

```
TCustomCollection<T: class> = class(TObjectEnumerable,
ICustomCollection, INotifiableCollection);
```

Remarks

The TCustomCollection{T} class inherits from TObjectEnumerable<T>, so it has methods for iterating through a collection of entities and access its items. Also, the class supports the ICustomCollection<T> interface and implements methods for modifying the entity collection: adding and removing items.

Since the collection operates with entities as elements, it has to be associated with a data

context.

TCustomCollection{T} is a base class, so it should not be used directly. For handling entities in the code, TCustomCollection{T} descendants such as TEntityCollection<T> and TObjectCollection<T> have to be used.

See Also

- TObjectEnumerable<T>
- TEntityCollection<T>
- TObjectCollection<T>

Request Support DAC Forum Provide Feedback

6.1.1.3.1 Members

TCustomCollection<T> class overview.

Methods

Name	Description
Add	Adds an element to the end of the collection.
Context	Indicates the data context with which the collection is associated.
Count	Returns the number of elements in a collection.
Delete	Removes the element at the specified index of the collection.
GetDeleted	Returns a subset of deleted elements in the collection.
Insert	Inserts an element into the collection at the specified index.
MetaType	Returns the meta-type of elements in a collection.
Remove	Removes the specified element from the collection.

6.1.1.3.2 Methods

Methods of the **TCustomCollection<T>** class.

For a complete list of the **TCustomCollection<T>** class members, see the TCustomCollection<T> Members topic.

Public

Name	Description
Add	Adds an element to the end of the collection.
Context	Indicates the data context with which the collection is associated.
Count	Returns the number of elements in a collection.
Delete	Removes the element at the specified index of the collection.
GetDeleted	Returns a subset of deleted elements in the collection.
Insert	Inserts an element into the collection at the specified index.
MetaType	Returns the meta-type of elements in a collection.
Remove	Removes the specified element from the collection.

See Also

- TCustomCollection<T> Class
- TCustomCollection<T> Class Members

6.1.1.3.2.1 Add Method

Adds an element to the end of the collection.

Class

TCustomCollection<T>

Syntax

```
function Add(Item: T): Integer; overload;
```

Parameters

Item

The element to be added to the end of the collection.

Return Value

A zero-based index of the added element.

Remarks

Use the method to add a new element to the collection.

Request Support DAC Forum Provide Feedback

6.1.1.3.2.2 Context Method

Indicates the data context with which the collection is associated.

Class

TCustomCollection<T>

Syntax

```
function Context: TCustomContext; virtual;
```

Remarks

Use the method to determine the data context with which the collection is associated.

Request Support DAC Forum Provide Feedback

6.1.1.3.2.3 Count Method

Returns the number of elements in a collection.

Class

TCustomCollection<T>

Syntax

```
function Count: Integer; override;
```

Remarks

Use the method to determine the number of elements in the collection.

Request Support DAC Forum Provide Feedback

6.1.1.3.2.4 Delete Method

Removes the element at the specified index of the collection.

Class

TCustomCollection<T>

Syntax

```
procedure Delete(Index: Integer);
```

Parameters

Index

The zero-based index of the element to remove.

Remarks

Use the method to remove the element at the specified index of the collection. If the specified index is less that zero or is equal to or greater than Count, than the appropriate exception will be raised.

See Also

- Count

Request Support DAC Forum Provide Feedback

6.1.1.3.2.5 GetDeleted Method

Returns a subset of deleted elements in the collection.

Class

TCustomCollection<T>

Syntax

```
function GetDeleted: IObjectEnumerable<T>;
```

Remarks

Use the method to retrieve a subset of the collection elements which was deleted using TCustomEntityContext.Delete but not saved yet. The list of deleted elements is returned as IObjectEnumerable<T>.

See Also

- IObjectEnumerable<T>
- TCustomEntityContext.Delete

Request Support DAC Forum Provide Feedback

6.1.1.3.2.6 Insert Method

Inserts an element into the collection at the specified index.

Class

TCustomCollection<T>

Syntax

```
procedure Insert(Index: Integer; Item: T);
```

Parameters

Index

The zero-based index at which the item should be inserted.

Item
The element to insert.

Remarks

Use the method to insert an element into the collection at the specified index. If the specified index is less that zero or is equal to or greater than Count, than the appropriate exception will be raised.

See Also

- Count

Request Support DAC Forum Provide Feedback

6.1.1.3.2.7 MetaType Method

Returns the meta-type of elements in a collection.

Class

TCustomCollection<T>

Syntax

```
function MetaType: TMetaType; override;
```

Remarks

Use the method to determine the meta-type of elements in the collection.

Request Support DAC Forum Provide Feedback

6.1.1.3.2.8 Remove Method

Removes the specified element from the collection.

Class

TCustomCollection<T>

Syntax

```
procedure Remove(Item: T);
```

Parameters

Item
The element to remove.

Remarks

Use the method to remove the specified element from the collection.

 Request Support DAC Forum Provide Feedback

6.1.1.4 TCustomContext Class

The base class that provides the data context functionality.

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

Unit

EntityDAC.Context

Syntax

```
TCustomContext = class(TComponent);
```

Remarks

TCustomContext class provides functionality for setting up the context environment and options. Also, it provides methods for direct database interaction.

Since TCustomContext is the base class, it should not be used directly. Instead, TCustomContext descendants such as TEntityContext have to be used.

See Also

- TEntityContext

 Request Support DAC Forum Provide Feedback

6.1.1.4.1 Members

TCustomContext class overview.

Properties

Name	Description
Connection	Identifies the connection component with which the data context is associated.
Dialect	Indicates the current SQL dialect used by the connection data provider.
Model	Specifies the meta model used by the data context.
ModelName	Specifies the name of the meta model used by the data context.

Methods

Name	Description
Create	Overloaded.Description is not available at the moment.
ExecuteQuery<T>	Overloaded.Description is not available at the moment.
ExecuteSQL	Overloaded.Description is not available at the moment.

Events

Name	Description
OnGetGeneratorValue	Occurs when an entity attribute value generator of type "custom" needs to generate its value.

6.1.1.4.2 Properties

Properties of the **TCustomContext** class.

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

Public

Name	Description
Connection	Identifies the connection component with which the data context is associated.
Dialect	Indicates the current SQL dialect used by the connection data provider.
Model	Specifies the meta model used by the data context.
ModelName	Specifies the name of the meta model used by the data context.

See Also

- TCustomContext Class
- TCustomContext Class Members

Request Support DAC Forum Provide Feedback

6.1.1.4.2.1 Connection Property

Identifies the connection component with which the data context is associated.

Class

TCustomContext

Syntax

```
property Connection: TEntityConnection;
```

Remarks

Use the property to access properties, events and methods of the connection associated with the data context. Set the property to associate the data context with the TEntityConnection component.

See Also

- TEntityConnection

 Request Support DAC Forum Provide Feedback

6.1.1.4.2.2 Dialect Property

Indicates the current SQL dialect used by the connection data provider.

Class

TCustomContext

Syntax

```
property Dialect: TSQLDialect;
```

Remarks

Read the Dialect property to access the instance of the current SQL dialect. The property is the same as the TEntityConnection.Dialect property. To set the current dialect, the TEntityConnection.DialectName property is used.

See Also

- TEntityConnection.Dialect
- TEntityConnection.DialectName

 Request Support DAC Forum Provide Feedback

6.1.1.4.2.3 Model Property

Specifies the meta model used by the data context.

Class

TCustomContext

Syntax

```
property Model: TMetaModel;
```

Remarks

The property specifies the meta model which is used by the data context.

Read the Model property to access the instance of the meta model. Application should not set the meta model using the Model property value. To set used meta model, the ModelName property is used. Unless the meta model is not specified using ModelName, the default connection meta model specified by the TEntityConnection.DefaultModelName property is used.

See Also

- ModelName
- TEntityConnection.DefaultModelName

Request Support DAC Forum Provide Feedback

6.1.1.4.2.4 ModelName Property

Specifies the name of the meta model used by the data context.

Class

TCustomContext

Syntax

```
property ModelName: string;
```

Remarks

The property specifies the name of the meta model which is used by the data context.

Read the ModelName property to determine the name of used meta model. Set the name of the property to specify the used meta model. When the valid model name is set, the corresponding meta model instance can be accessed through the Model property. Unless the meta model name is not specified using ModelName, the default connection meta model specified by the TEntityConnection.DefaultModelName property is used.

See Also

- Model
- TEntityConnection.DefaultModelName

 Request Support DAC Forum Provide Feedback

6.1.1.4.3 Methods

Methods of the **TCustomContext** class.

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

Public

Name	Description
Create	Overloaded.Description is not available at the moment.

See Also

- TCustomContext Class
- TCustomContext Class Members

 Request Support DAC Forum Provide Feedback

6.1.1.4.3.1 Create Method

Class

`TCustomContext`

Overload List

Name	Description
Create	Creates a new instance of the data context.
Create(AOwner: TComponent)	Creates a new instance of the data context component.
Create(AOwner: TComponent; AModel: TMetaModel; AConnection: TEntityConnection)	Creates a new instance of the data context.

Create(AModel: TMetaModel; AConnection: TEntityConnection)	Creates a new instance of the data context.

Creates a new instance of the data context.

Class

TCustomContext

Syntax

```
constructor Create; reintroduce; overload; virtual;
```

Remarks

Use the constructor to create a new data context instance at run-time. After creating a data context instance, it should be configured by setting TCustomContext.Connection, TCustomContext.ModelName, TCustomContext.Options properties etc.

To create a data context instance and simultaneously associate it with a connection and a meta-model, use overloaded constructors instead.

See Also

- Create
- Create
- Create
- TCustomContext.Connection
- TCustomContext.ModelName
- TCustomContext.Options

Creates a new instance of the data context component.

Class

TCustomContext

Syntax

```
constructor Create(AOwner: TComponent); overload; override;
```

Remarks

The constructor is used to create a new TCustomContext descendant such as TEntityContext at design-time. After creating a data context component instance, it should be configured by setting TCustomContext.Connection, TCustomContext.ModelName, TCustomContext.Options properties etc.

To create a data context component instance and simultaneously associate it with a connection and a meta-model, use overloaded constructors instead.

See Also

- Create
- Create
- Create
- TEntityContext
- TCustomContext.Connection
- TCustomContext.ModelName
- TCustomContext.Options

Creates a new instance of the data context.

Class

TCustomContext

Syntax

```
constructor Create(AOwner: TComponent; AModel: TMetaModel;
AConnection: TEntityConnection); reintroduce; overload; virtual;
```

Remarks

The constructor is used to create a new TCustomContext descendant such as TEntityContext at design-time and simultaneously associate it with a connection and a meta-model. After creating a data context component instance, it can be additionally configured by setting the TCustomContext.Options property.

See Also

- Create
- Create
- Create
- TEntityContext
- TCustomContext.Connection
- TCustomContext.ModelName
- TCustomContext.Options
- TMetaModel
- TEntityConnection

Creates a new instance of the data context.

Class

TCustomContext

Syntax

```
constructor Create(AModel: TMetaModel; AConnection:
TEntityConnection); reintroduce; overload; virtual;
```

Remarks

Use the constructor to create a new data context instance at run-time and simultaneously associate it with a connection and a meta-model. After creating a data context instance, it can be additionally configured by setting the TCustomContext.Options property.

See Also

- Create
- Create
- Create
- TCustomContext.Connection
- TCustomContext.ModelName
- TCustomContext.Options
- TMetaModel
- TEntityConnection

6.1.1.4.3.2 ExecuteQuery<T> Method

Class

TCustomContext

Overload List

Name	Description
ExecuteQuery<T>	Execute a query directly in the database and returns the result collection.
ExecuteQuery<T>	Execute a query directly in the database and returns the result collection.

Execute a query directly in the database and returns the result collection.

Unit

Syntax

Remarks

Call the ExecuteQuery method to execute a query directly in the database and obtain its result as a collection that implements the IEntityEnumerable interface. Supply the statement as a TSQLStatement class instance, which encapsulates the query text and its parameters. The TSQLStatement.Params array must contain all IN and OUT parameters defined in the query. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the method.

See Also

- IEntityEnumerable
- TSQLStatement
- TSQLStatement.Params

Request Support DAC Forum Provide Feedback

Execute a query directly in the database and returns the result collection.

Unit

Syntax

Remarks

Call the ExecuteQuery method to execute a query directly in the database and obtain its result as a collection that implements the IEntityEnumerable interface. Supply the Params collection with the parameters accordingly to the ones in the query which itself is passed in the SQL string parameter. The Params array must contain all IN and OUT parameters defined in the query. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the method.

See Also

- IEntityEnumerable
- TSQLStatement
- TSQLStatement.Params

 Request Support DAC Forum Provide Feedback

6.1.1.4.3.3 ExecuteSQL Method

Class

TCustomContext

Overload List

Name	Description
ExecuteSQL	Executes a SQL statement directly in the database.
ExecuteSQL	Execute a SQL statement directly in the database.

 Request Support DAC Forum Provide Feedback

Executes a SQL statement directly in the database.

Unit

Syntax

Remarks

Call the ExecuteSQL method to execute a SQL statement directly in the database. Supply the statement as a TSQLStatement class instance, which encapsulates the SQL statement text and its parameters. The TSQLStatement.Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the method.

See Also

- TSQLStatement
- TSQLStatement.Params

 Request Support DAC Forum Provide Feedback

Execute a SQL statement directly in the database.

Unit

Syntax

Remarks

Call the ExecuteSQL method to execute a SQL statement directly in the database. Supply the Params collection with the parameters accordingly to the ones in the SQL statement which itself is passed in the SQL string parameter. The Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the method.

 Request Support DAC Forum Provide Feedback

6.1.1.4.4 Events

Events of the **TCustomContext** class.

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

Public

Name	Description
OnGetGeneratorValue	Occurs when an entity attribute value generator of type "custom" needs to generate its value.

See Also

- TCustomContext Class
- TCustomContext Class Members

6.1.1.4.4.1 OnGetGeneratorValue Event

Occurs when an entity attribute value generator of type "custom" needs to generate its value.

Class

TCustomContext

Syntax

```
property OnGetGeneratorValue: TGetGeneratorValueEvent;
```

Remarks

Write the event handler to implement the algorithm of generating the next value of an entity attribute for which the "custom" generator is defined.

Parameters:

Attribute

Specifies the corresponding meta-attribute.

Value

Specifies the variable in which the next value has to be returned.

Success

Should be set to True if the value has been generated successfully.

6.1.1.5 TDataContext Class

The class provides functions for data context management.

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

Unit

EntityDAC.Context

Syntax

```
TDataContext = class(TCustomContext);
```

Inheritance Hierarchy

TCustomContext

TDataContext

Request Support DAC Forum Provide Feedback

6.1.1.5.1 Members

TDataContext class overview.

Properties

Name	Description
Connection (inherited from **TCustomContext**)	Identifies the connection component with which the data context is associated.
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
ModelName (inherited from **TCustomContext**)	Specifies the name of the meta model used by the data context.
Types	The property is designed to determine a meta-type by a meta-type name.

Methods

Name	Description
Create (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.

ExecuteQuery<T> (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
ExecuteSQL (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
RejectChanges	The method is designed to cancel changes in all attached entities
SubmitChanges	The method is designed for saving changes in all attached entities.

Events

Name	Description
OnGetGeneratorValue (inherited from **TCustomContext**)	Occurs when an entity attribute value generator of type "custom" needs to generate its value.

Request Support DAC Forum Provide Feedback

6.1.1.5.2 Properties

Properties of the **TDataContext** class.

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

Public

Name	Description
Connection (inherited from **TCustomContext**)	Identifies the connection component with which the data context is associated.
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
ModelName (inherited from **TCustomContext**)	Specifies the name of the meta model used by the data context.

Types	The property is designed to determine a meta-type by a meta-type name.

See Also

- TDataContext Class
- TDataContext Class Members

Request Support DAC Forum Provide Feedback

6.1.1.5.2.1 Types Property(Indexer)

The property is designed to determine a meta-type by a meta-type name.

Class

TDataContext

Syntax

```
property Types[Name: string]: IMetaType; default;
```

Parameters

Name

The name of the meta-type.

Remarks

The property returns a meta-type by a specified name. The returned meta-type is used when calling the TCustomEntityContext.CreateEntity, TCustomEntityContext.CreateAttachedEntity, TCustomEntityContext.GetEntity, TCustomEntityContext.GetEntities methods for type identification of a retrieved entity.

The specified meta-type name is case-insensitive. If a meta-type with the specified name doesn't exist, an exception will be generated.

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.CreateAttachedEntity

- TCustomEntityContext.GetEntity
- TCustomEntityContext.GetEntities

6.1.1.5.3 Methods

Methods of the **TDataContext** class.

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

Public

Name	Description
Create (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
RejectChanges	The method is designed to cancel changes in all attached entities
SubmitChanges	The method is designed for saving changes in all attached entities.

See Also

- TDataContext Class
- TDataContext Class Members

6.1.1.5.3.1 RejectChanges Method

The method is designed to cancel changes in all attached entities

Class

`TDataContext`

Syntax

```
procedure RejectChanges;
```

Remarks

The method cancels changes, that were not yet saved with TCustomEntityContext.Save or TDataContext.SubmitChanges, in all entities attached to the data context. Calling of the method is equivalent to consequent calling of the TCustomEntityContext.Cancel method for each entity attached to the data context, that was modified or deleted with TCustomEntityContext.Delete.

See Also

- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- SubmitChanges

Request Support DAC Forum Provide Feedback

6.1.1.5.3.2 SubmitChanges Method

The method is designed for saving changes in all attached entities.

Class

TDataContext

Syntax

```
procedure SubmitChanges;
```

Remarks

The method performs permanent saving of changes in all entities attached to the data context. Calling of the method is equivalent to consequent calling of the TCustomEntityContext.Save method for each entity attached to the data context, that was modified or deleted with TCustomEntityContext.Delete.

See Also

- TCustomEntityContext.Delete

- TCustomEntityContext.Save
- RejectChanges

 Request Support DAC Forum Provide Feedback

6.1.1.6 TEntityCollection<T> Class

Provides functionality for handling a collection of entities.

For a list of all members of this type, see TEntityCollection<T> members.

Unit

EntityDAC.Context

Syntax

```
TEntityCollection<T: class> = class(TCustomCollection,
IEntityCollection, IEntityEnumerable, ICustomCollection,
IObjectEnumerable, IEnumerable, IEnumerable);
```

Remarks

The TEntityCollection{T} class implements methods for iterating through a collection of entities and access its items. Also, the class implements methods for modifying the entity collection: adding and removing items.

 Request Support DAC Forum Provide Feedback

6.1.1.6.1 Members

TEntityCollection<T> class overview.

 Request Support DAC Forum Provide Feedback

6.1.1.7 TObjectCollection<T> Class

Provides functionality for handling a collection of entities which are not TEntity descendants.

For a list of all members of this type, see TObjectCollection<T> members.

Unit

EntityDAC.Context

Syntax

```
TObjectCollection<T: class> = class(TCustomCollection,
IObjectCollection, IObjectEnumerable, IEnumerable);
```

Remarks

The TObjectCollection{T} class implements methods for iterating through a collection of entities and access its items. Also, the class implements methods for modifying the entity collection: adding and removing items.

Request Support DAC Forum Provide Feedback

6.1.1.7.1 Members

TObjectCollection<T> class overview.

Methods

Name	Description
GetFiltered	Returns a subset of the collection elements filtered by the specified expression.

Request Support DAC Forum Provide Feedback

6.1.1.7.2 Methods

Methods of the **TObjectCollection<T>** class.

For a complete list of the **TObjectCollection<T>** class members, see the TObjectCollection<T> Members topic.

Public

Name	Description

GetFiltered	Returns a subset of the collection elements filtered by the specified expression.

See Also

- TObjectCollection<T> Class
- TObjectCollection<T> Class Members

6.1.1.7.2.1 GetFiltered Method

Returns a subset of the collection elements filtered by the specified expression.

Class

TObjectCollection<T>

Syntax

```
function GetFiltered(AFilter: ICompiledExpressionStatement): IObjectEnumerable<T>; override;
```

Parameters

AFilter

Remarks

Use the method to retrieve a subset of the collection elements filtered by the specified expression.

See Also

- ICompiledExpressionStatement

6.1.2 Interfaces

Interfaces in the **EntityDAC.Context** unit.

Interfaces

Name	Description
ICustomCollection<T>	The base interface which declares functionality for entity collections.

 Request Support DAC Forum Provide Feedback

6.1.2.1 ICustomCollection<T> Interface

The base interface which declares functionality for entity collections.

Unit

EntityDAC.Context

Syntax

```
ICustomCollection<T: class> = interface(IObjectEnumerable)
[ '<08017590-4BF1-40AE-82AC-DB6B0FC30195>' ];
```

Remarks

The ICustomCollection{T} interface declares properties and methods for implementing entity collections. All collection classes used in EntityDAC, such as TEntityCollection<T> implement ICustomCollection{T} interface.

See Also

- TEntityCollection<T>

 Request Support DAC Forum Provide Feedback

6.1.2.1.1 Members

ICustomCollection<T> class overview.

Methods

Name	Description
Add	Adds an element to the end of the collection.
Context	Indicates the data context with which the collection is associated.
Delete	Removes the element at the specified index of the collection.
GetDeleted	Returns a subset of deleted elements in the collection.
Insert	Inserts an element into the collection at the specified index.
Remove	Removes the specified element from the collection.

Request Support DAC Forum Provide Feedback

6.1.2.1.2 Methods

Methods of the **ICustomCollection<T>** class.

For a complete list of the **ICustomCollection<T>** class members, see the ICustomCollection<T> Members topic.

Public

Name	Description
Add	Adds an element to the end of the collection.
Context	Indicates the data context with which the collection is associated.
Delete	Removes the element at the specified index of the collection.
GetDeleted	Returns a subset of deleted elements in the collection.

Insert	Inserts an element into the collection at the specified index.
Remove	Removes the specified element from the collection.

See Also

- ICustomCollection<T> Interface
- ICustomCollection<T> Interface Members

6.1.2.1.2.1 Add Method

Adds an element to the end of the collection.

Class

ICustomCollection<T>

Syntax

```
function Add(Item: T): Integer;
```

Parameters

Item

Adds an element to the end of the collection.

Remarks

Use the method to add a new element to the collection.

6.1.2.1.2.2 Context Method

Indicates the data context with which the collection is associated.

Class

ICustomCollection<T>

Syntax

```
function Context: TCustomContext;
```

Remarks

Indicates the data context with which the collection is associated.

Request Support DAC Forum Provide Feedback

6.1.2.1.2.3 Delete Method

Removes the element at the specified index of the collection.

Class

ICustomCollection<T>

Syntax

```
procedure Delete(Index: Integer);
```

Parameters

Index

Removes the element at the specified index of the collection.

Remarks

Use the method to remove the element at the specified index of the collection. If the specified index is less that zero or is equal to or greater than elements count, than the appropriate exception will be raised.

Request Support DAC Forum Provide Feedback

6.1.2.1.2.4 GetDeleted Method

Returns a subset of deleted elements in the collection.

Class

ICustomCollection<T>

Syntax

```
function GetDeleted: IObjectEnumerable<T>;
```

Remarks

Use the method to retrieve a subset of the collection elements which was deleted using Delete but not saved yet. The list of deleted elements is returned as IObjectEnumerable<T>.

See Also

- Delete
- IObjectEnumerable<T>

 Request Support DAC Forum Provide Feedback

6.1.2.1.2.5 Insert Method

Inserts an element into the collection at the specified index.

Class

ICustomCollection<T>

Syntax

```
procedure Insert(Index: Integer; Item: T);
```

Parameters

Index

The zero-based index at which the item should be inserted.

Item

The element to insert.

Remarks

Use the method to insert an element into the collection at the specified index. If the specified index is less that zero or is equal to or greater than elements count, than the appropriate exception will be raised.

 Request Support DAC Forum Provide Feedback

6.1.2.1.2.6 Remove Method

Removes the specified element from the collection.

Class

ICustomCollection<T>

Syntax

```
procedure Remove(Item: T);
```

Parameters

Item

Removes the specified element from the collection.

Remarks

Use the method to remove the specified element from the collection.

6.1.3 Types

Types in the **EntityDAC.Context** unit.

Types

Name	Description
TCollectionOptions	Allows to control behavior of linked entity collections. The property is a set of TCollectionOption flags.

6.1.3.1 TCollectionOptions Set

Allows to control behavior of linked entity collections. The property is a set of TCollectionOption flags.

Unit

EntityDAC.Context

Syntax

```
TCollectionOptions = set of TCollectionOption;
```

Remarks

The defulat value is coKeyOrdered

Request Support DAC Forum Provide Feedback

6.1.4 Enumerations

Enumerations in the **EntityDAC.Context** unit.

Enumerations

Name	Description
TCollectionOption	A set of TCollectionOptions

Request Support DAC Forum Provide Feedback

6.1.4.1 TCollectionOption Enumeration

A set of TCollectionOptions

Unit

EntityDAC.Context

Syntax

```
TCollectionOption = (coKeyOrdered, coLoadForNewEntities);
```

Values

Value	Meaning
coKeyOrdered	Defines parameters of linked entity collection sorting. If the flag is selected, then on retrieving a linked entity collection from the database it will be sorted by the primary key. If the flag is cleared,

	then the entity collection will be retrieved unsorted, depending on specifities of the used DBMS. Selected by default.
coLoadForNewEntities	Defines parameters of linked entity collection loading for a newly created entity. If the flag is selected, then on calling a linked collection of a newly created entity, the context will attempt to retrieve the collection from the database. If the flag is cleared, then on calling a linked collection of a newly created entity there will be no attempts to access the database The flag affects linked entity collection loading of newly created entities only. If the entity was retrieved from the database using GetEntity or GetEntities methods, then on calling a linked collection of this entity, the context will always read out the linked collection from the database. The flag is cleared by default.

6.2 EntityDAC.DataProvider

Contains implementation of Data Provider functionality

Classes

Name	Description
TDataProvider	The base class that provides the data providers functionality.

Types

Name	Description
TDataProviderClass	Basic class implementing data provider functionality

6.2.1 Classes

Classes in the **EntityDAC.DataProvider** unit.

Classes

Name	Description
TDataProvider	The base class that provides the data providers functionality.

Request Support DAC Forum Provide Feedback

6.2.1.1 TDataProvider Class

The base class that provides the data providers functionality.

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

Unit

EntityDAC.DataProvider

Syntax

```
TDataProvider = class(TComponent);
```

Remarks

TDataProvider class provides functionality to manage data providers.

Since TDataProvider is the base class, it should not be used directly. Instead, TDataProvider descendants such as TUniDACDataProvider have to be used.

See Also

- TEntityContext

Request Support DAC Forum Provide Feedback

6.2.1.1.1 Members

TDataProvider class overview.

Methods

Name	Description

MultiDialect	Specifies whether the data provider supports multiple SQL dialects.
ProviderName	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.2.1.1.2 Methods

Methods of the **TDataProvider** class.

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

Public

Name	Description
MultiDialect	Specifies whether the data provider supports multiple SQL dialects.
ProviderName	Specifies the name of the current data provider used for establishing a connection.

See Also

- TDataProvider Class
- TDataProvider Class Members

 Request Support DAC Forum Provide Feedback

6.2.1.1.2.1 MultiDialect Method

Specifies whether the data provider supports multiple SQL dialects.

Class

TDataProvider

Syntax

```
class function MultiDialect: boolean; virtual;
```

Remarks

The MultiDialectSupported property is used to determine whether the data provider supports working with several SQL dialects.

Request Support DAC Forum Provide Feedback

6.2.1.1.2.2 ProviderName Method

Specifies the name of the current data provider used for establishing a connection.

Class

TDataProvider

Syntax

```
class function ProviderName: string; virtual;
```

Remarks

Since EntityDAC is abstracted from the data access layer, direct interaction with the database is handled by specialized component packages such as Devart Data Access Components. A data provider is the specialized class that provides interaction between TEntityConnection and used data access components. All data providers used in an application are registered by the data provider manager, and available through their names.

Read the property value to determine the current provider name. Set the ProviderName property value to specify the current data provider name.

When a valid provider name is set, the corresponding data provider instance can be accessed through the TEntityConnection.Provider property. And vice versa, when the P:Devart.EntityDAC.TEntityConnection.DataProvider property is set, the corresponding provider name is assigned to the ProviderName property.

To define a database-specific behavior of the current data provider,

TEntityConnection.DialectName and TEntityConnection.Dialect properties are used.

See Also

- TEntityConnection.Provider
- TEntityConnection.DialectName
- TEntityConnection.Dialect
- TDataProvider

 Request Support DAC Forum Provide Feedback

6.2.2 Types

Types in the **EntityDAC.DataProvider** unit.

Types

Name	Description
TDataProviderClass	Basic class implementing data provider functionality

 Request Support DAC Forum Provide Feedback

6.2.2.1 TDataProviderClass Class Reference

Basic class implementing data provider functionality

Unit

EntityDAC.DataProvider

Syntax

```
TDataProviderClass = class of TDataProvider;
```

 Request Support DAC Forum Provide Feedback

6.3 EntityDAC.DataProvider.ADO

Links the data provider for ADO to an application.

Classes

Name	Description
TADODataProvider	Links the data provider for ADO components to an application.

 Request Support DAC Forum Provide Feedback

6.3.1 Classes

Classes in the **EntityDAC.DataProvider.ADO** unit.

Classes

Name	Description
TADODataProvider	Links the data provider for ADO components to an application.

 Request Support DAC Forum Provide Feedback

6.3.1.1 TADODataProvider Class

Links the data provider for ADO components to an application.

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

Unit

EntityDAC.DataProvider.ADO

Syntax

```
TADODataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TADODataProvider

 Request Support DAC Forum Provide Feedback

6.3.1.1.1 Members

TADODataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.4 EntityDAC.DataProvider.DBX

Links the data provider for dbExpress to an application.

Classes

Name	Description
TDBXDataProvider	Links the data provider for dbExpress components to an application.

 Request Support DAC Forum Provide Feedback

6.4.1 Classes

Classes in the **EntityDAC.DataProvider.DBX** unit.

Classes

Name	Description
TDBXDataProvider	Links the data provider for dbExpress components to an application.

 Request Support DAC Forum Provide Feedback

6.4.1.1 TDBXDataProvider Class

Links the data provider for dbExpress components to an application.

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

Unit

EntityDAC.DataProvider.DBX

Syntax

```
TDBXDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TDBXDataProvider

 Request Support DAC Forum Provide Feedback

6.4.1.1.1 Members

TDBXDataProvider class overview.

Methods

Name	Description

MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.5 EntityDAC.DataProvider.FireDAC

Links the data provider for FireDAC to an application.

Classes

Name	Description
TFireDACDataProvider	Links the data provider for FireDAC to an application.

 Request Support DAC Forum Provide Feedback

6.5.1 Classes

Classes in the **EntityDAC.DataProvider.FireDAC** unit.

Classes

Name	Description
TFireDACDataProvider	Links the data provider for FireDAC to an application.

 Request Support DAC Forum Provide Feedback

6.5.1.1 TFireDACDataProvider Class

Links the data provider for FireDAC to an application.

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

Unit

EntityDAC.DataProvider.FireDAC

Syntax

```
TFireDACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TFireDACDataProvider

 Request Support DAC Forum Provide Feedback

6.5.1.1.1 Members

TFireDACDataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.6 EntityDAC.DataProvider.IBDAC

Links the data provider for Devart InterBase Data Access Components to an application.

Classes

Name	Description
TIBDACDataProvider	Links the data provider for

	Devart InterBase Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.6.1 Classes

Classes in the **EntityDAC.DataProvider.IBDAC** unit.

Classes

Name	Description
TIBDACDataProvider	Links the data provider for Devart InterBase Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.6.1.1 TIBDACDataProvider Class

Links the data provider for Devart InterBase Data Access Components to an application.

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

Unit

EntityDAC.DataProvider.IBDAC

Syntax

```
TIBDACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TIBDACDataProvider

 Request Support DAC Forum Provide Feedback

6.6.1.1.1 Members

TIBDACDataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.7 EntityDAC.DataProvider.IBX

Links the data provider for IBX data access components to an application.

Classes

Name	Description
TIBXDataProvider	Links the data provider for InterBase Express components to an application.

 Request Support DAC Forum Provide Feedback

6.7.1 Classes

Classes in the **EntityDAC.DataProvider.IBX** unit.

Classes

Name	Description
TIBXDataProvider	Links the data provider for InterBase Express components to an

	application.

Request Support DAC Forum Provide Feedback

6.7.1.1 TIBXDataProvider Class

Links the data provider for InterBase Express components to an application.

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

Unit

EntityDAC.DataProvider.IBX

Syntax

```
TIBXDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TIBXDataProvider

Request Support DAC Forum Provide Feedback

6.7.1.1.1 Members

TIBXDataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

Request Support DAC Forum Provide Feedback

Reserved.

6.8 EntityDAC.DataProvider.LiteDAC

Links the data provider for Devart SQLite Data Access Components to an application.

Classes

Name	Description
TLiteDACDataProvider	Links the data provider for Devart SQLite Data Access Components to an application.

Request Support DAC Forum Provide Feedback

6.8.1 Classes

Classes in the **EntityDAC.DataProvider.LiteDAC** unit.

Classes

Name	Description
TLiteDACDataProvider	Links the data provider for Devart SQLite Data Access Components to an application.

Request Support DAC Forum Provide Feedback

6.8.1.1 TLiteDACDataProvider Class

Links the data provider for Devart SQLite Data Access Components to an application.

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

Unit

EntityDAC.DataProvider.LiteDAC

Syntax

```
TLiteDACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TLiteDACDataProvider

 Request Support DAC Forum Provide Feedback

6.8.1.1.1 Members

TLiteDACDataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.9 EntityDAC.DataProvider.MyDAC

Links the data provider for Devart MySQL Data Access Components to an application.

Classes

Name	Description
TMyDACDataProvider	Links the data provider for Devart MySQL Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.9.1 Classes

Classes in the **EntityDAC.DataProvider.MyDAC** unit.

Classes

Name	Description
TMyDACDataProvider	Links the data provider for Devart MySQL Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.9.1.1 TMyDACDataProvider Class

Links the data provider for Devart MySQL Data Access Components to an application.

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

Unit

EntityDAC.DataProvider.MyDAC

Syntax

```
TMyDACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TMyDACDataProvider

 Request Support DAC Forum Provide Feedback

6.9.1.1.1 Members

TMyDACDataProvider class overview.

Methods

Name	Description

MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.10 EntityDAC.DataProvider.ODAC

Links the data provider for Devart Oracle Data Access Components to an application.

Classes

Name	Description
TODACDataProvider	Links the data provider for Devart Oracle Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.10.1 Classes

Classes in the **EntityDAC.DataProvider.ODAC** unit.

Classes

Name	Description
TODACDataProvider	Links the data provider for Devart Oracle Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.10.1.1 TODACDataProvider Class

Links the data provider for Devart Oracle Data Access Components to an application.

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

Unit

EntityDAC.DataProvider.ODAC

Syntax

```
TODACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TODACDataProvider

 Request Support DAC Forum Provide Feedback

6.10.1.1.1 Members

TODACDataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

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

6.11 EntityDAC.DataProvider.PgDAC

Links the data provider for Devart PostgreSQL Data Access Components to an application.

Classes

Name	Description
TPgDACDataProvider	Links the data provider for Devart PostgreSQL Data Access Components to an application.

Request Support DAC Forum Provide Feedback

6.11.1 Classes

Classes in the **EntityDAC.DataProvider.PgDAC** unit.

Classes

Name	Description
TPgDACDataProvider	Links the data provider for Devart PostgreSQL Data Access Components to an application.

Request Support DAC Forum Provide Feedback

6.11.1.1 TPgDACDataProvider Class

Links the data provider for Devart PostgreSQL Data Access Components to an application.

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

Unit

EntityDAC.DataProvider.PgDAC

Syntax

```
TPgDACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TPgDACDataProvider

 Request Support DAC Forum Provide Feedback

6.11.1.1.1 Members

TPgDACDataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.12 EntityDAC.DataProvider.SDAC

Links the data provider for Devart SQL Server Data Access Components to an application.

Classes

Name	Description
TSDACDataProvider	Links the data provider for Devart SQL Server Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.12.1 Classes

Classes in the **EntityDAC.DataProvider.SDAC** unit.

Classes

Name	Description
TSDACDataProvider	Links the data provider for Devart SQL Server Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.12.1.1 TSDACDataProvider Class

Links the data provider for Devart SQL Server Data Access Components to an application.

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

Unit

EntityDAC.DataProvider.SDAC

Syntax

```
TSDACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TSDACDataProvider

 Request Support DAC Forum Provide Feedback

6.12.1.1.1 Members

TSDACDataProvider class overview.

Methods

Name	Description

MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

 Request Support DAC Forum Provide Feedback

6.13 EntityDAC.DataProvider.UniDAC

Links the data provider for Devart Universal Data Access Components to an application.

Classes

Name	Description
TUniDACDataProvider	Links the data provider for Devart Universal Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.13.1 Classes

Classes in the **EntityDAC.DataProvider.UniDAC** unit.

Classes

Name	Description
TUniDACDataProvider	Links the data provider for Devart Universal Data Access Components to an application.

 Request Support DAC Forum Provide Feedback

6.13.1.1 TUniDACDataProvider Class

Links the data provider for Devart Universal Data Access Components to an application.

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

Unit

EntityDAC.DataProvider.UniDAC

Syntax

```
TUniDACDataProvider = class(TDataProvider);
```

Inheritance Hierarchy

TDataProvider

TUniDACDataProvider

 Request Support DAC Forum Provide Feedback

6.13.1.1.1 Members

TUniDACDataProvider class overview.

Methods

Name	Description
MultiDialect (inherited from **TDataProvider**)	Specifies whether the data provider supports multiple SQL dialects.
ProviderName (inherited from **TDataProvider**)	Specifies the name of the current data provider used for establishing a connection.

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

6.14 EntityDAC.Entity

The unit contains implementation of the entity management.

Classes

Name	Description
TEntity	The base class for representing entity instances.
TEntityReference	The base class for representing the entity reference.
TEntityReferences	The base class for representing a list of the entity references.
TUnmappedEntity	The class represents an unmapped entity instance.

Request Support DAC Forum Provide Feedback

6.14.1 Classes

Classes in the **EntityDAC.Entity** unit.

Classes

Name	Description
TEntity	The base class for representing entity instances.
TEntityReference	The base class for representing the entity reference.
TEntityReferences	The base class for representing a list of the entity references.
TUnmappedEntity	The class represents an unmapped entity instance.

Request Support DAC Forum Provide Feedback

6.14.1.1 TEntity Class

The base class for representing entity instances.

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

Unit

EntityDAC.Entity

Syntax

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

Remarks

TEntity is a base class for representing entities and should not be used directly. For operating entities in the code, TMappedEntity and TUnmappedEntity classes that are TEntity descendants, are used. TMappedEntity holds the instance of an entity that is mapped to the particular database table. TUnmappedEntity holds the instance of an entity that is the result of a query execution and can not be mapped to the particular table.

See Also

- TMappedEntity
- TUnmappedEntity

Request Support DAC Forum Provide Feedback

6.14.1.1.1 Members

TEntity class overview.

Properties

Name	Description
Attributes	The property represents the entity attributes collection.
Collections	The property represents a list of the entity collections.
EntityState	The property indicates the entity state.

MetaType	The property indicates the entity meta-type.
References	The property represents a list of the entity references.
UpdateState	The property indicates the entity update state.

Methods

Name	Description
AttributeByName	Returns an entity attribute by its name.
Compare	The method is designed for comparing the entity key with the specified key.
Create	Overloaded. The constructor is designed for creating a new entity instance.
FromKey	The method is designed for setting the entity key value.
IsAttached	The method is designed to determine whether the entity is attached.
ToKey	The method is designed for filling the specified key with the entity key value.

6.14.1.1.2 Properties

Properties of the **TEntity** class.

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

Public

Name	Description
Attributes	The property represents the entity attributes collection.
Collections	The property represents a list of the entity collections.

EntityState	The property indicates the entity state.
MetaType	The property indicates the entity meta-type.
References	The property represents a list of the entity references.
UpdateState	The property indicates the entity update state.

See Also

- TEntity Class
- TEntity Class Members

Request Support DAC Forum Provide Feedback

6.14.1.1.2.1 Attributes Property

The property represents the entity attributes collection.

Class

TEntity

Syntax

```
property Attributes: TEntityAttributes;
```

Remarks

The TEntityAttributes class provides methods for iterating and accessing entity attributes. Also, a particular entity attribute can be accessed using the AttributeByName method.

See Also

- AttributeByName

Request Support DAC Forum Provide Feedback

6.14.1.1.2.2 Collections Property

The property represents a list of the entity collections.

Class

TEntity

Syntax

```
property Collections: TEntityEnumerables;
```

Remarks

The TEntityCollection class represents a list of the entity collections and provides methods for iterating and accessing the list elements.

 Request Support DAC Forum Provide Feedback

6.14.1.1.2.3 EntityState Property

The property indicates the entity state.

Class

TEntity

Syntax

```
property EntityState: TEntityState;
```

 Request Support DAC Forum Provide Feedback

6.14.1.1.2.4 MetaType Property

The property indicates the entity meta-type.

Class

TEntity

Syntax

```
property MetaType: TMetaType;
```

See Also

- TMetaType

 Request Support DAC Forum Provide Feedback

6.14.1.1.2.5 References Property

The property represents a list of the entity references.

Class

TEntity

Syntax

```
property References: TEntityReferences;
```

Remarks

The TEntityReferences class represents a list of the entity references and provides methods for iterating and accessing the list elements.

 Request Support DAC Forum Provide Feedback

6.14.1.1.2.6 UpdateState Property

The property indicates the entity update state.

Class

TEntity

Syntax

```
property UpdateState: TUpdateState;
```

 Request Support DAC Forum Provide Feedback

6.14.1.1.3 Methods

Methods of the **TEntity** class.

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

Public

Name	Description
AttributeByName	Returns an entity attribute by its name.
Compare	The method is designed for comparing the entity key with the specified key.
Create	Overloaded. The constructor is designed for creating a new entity instance.
FromKey	The method is designed for setting the entity key value.
IsAttached	The method is designed to determine whether the entity is attached.
ToKey	The method is designed for filling the specified key with the entity key value.

See Also

- TEntity Class
- TEntity Class Members

6.14.1.1.3.1 AttributeByName Method

Returns an entity attribute by its name.

Class

`TEntity`

Syntax

```
function AttributeByName(const Name: string): TEntityAttribute;
```

Parameters

Name

Name of the attribute. The name is case-insensitive.

Remarks

Call AttributeByName to retrieve attribute information for an entity attribute when only its name is known. Name is the name of an existing attribute. AttributeByName returns the TEntityAttribute instance for the specified attribute. If the specified attribute does not exist, AttributeByName returns nil.

See Also

- TEntityAttribute

Request Support DAC Forum Provide Feedback

6.14.1.1.3.2 Compare Method

The method is designed for comparing the entity key with the specified key.

Class

TEntity

Syntax

```
function Compare(Key: TCustomKey): Integer;
```

Parameters

Key

The key to be compared with the entity key.

Return Value

When the entity key value is null, the result is 0 if the key value is also null, 1 otherwise. When the entity key value is not null, the result is -1 if the key value is null or is greater, 1 if the key value is less, and 0 when the values are equal.

Remarks

The method compares the entity key value with the specified key value. The specified key has to be of the same meta-type as the entity. Otherwise, the exception will be raised. When the

entity key is complex and consists of several entity attributes, each entity key attribute is compared with the corresponding key attribute. When all attributes are equal, the method returns zero. Otherwise, the method returns the result of the comparison of the first non-equal attribute pair.

 Request Support DAC Forum Provide Feedback

6.14.1.1.3.3 Create Method

The constructor is designed for creating a new entity instance.

Class

TEntity

Overload List

Name	Description
Create	The constructor is designed for creating a new entity instance.
Create(**const** KeyValue: Variant)	The constructor is designed for creating a new entity instance.
Create(**const** KeyValues: **array of** Variant)	The constructor is designed for creating a new entity instance.

 Request Support DAC Forum Provide Feedback

The constructor is designed for creating a new entity instance.

Class

TEntity

Syntax

```
constructor Create; overload; virtual;
```

Remarks

The method creates a new entity instance. The entity attributes forming the primary key will

be initialized by the default values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache, the TCustomEntityContext.Attach method should be used.

The method is the analogue of the TCustomEntityContext.CreateEntity method.

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach

Request Support DAC Forum Provide Feedback

The constructor is designed for creating a new entity instance.

Class

TEntity

Syntax

```
constructor Create(const KeyValue: Variant); overload;
```

Remarks

The method creates a new entity instance. The entity key attribute will be initialized by the specified value. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache, the

TCustomEntityContext.Attach method should be used.

The method is the analogue of the TCustomEntityContext.CreateEntity method.

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.CreateEntity

The constructor is designed for creating a new entity instance.

Class

TEntity

Syntax

```
constructor Create(const KeyValues: array of Variant);
overload;
```

Remarks

The method creates a new entity instance. The entity key attribute will be initialized by the specified value. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache, the TCustomEntityContext.Attach method should be used.

The method is the analogue of the TCustomEntityContext.CreateEntity method.

See Also

- TCustomEntityContext.Attach

- TCustomEntityContext.CreateEntity

 Request Support DAC Forum Provide Feedback

6.14.1.1.3.4 Destroy Destructor

The destructor is designed for destroying an entity instance.

Class

TEntity

Syntax

```
destructor Destroy; override;
```

Remarks

The method destroys an entity instance. The method can be used only for those entities which are not attached to the data context using the TCustomEntityContext.Attach method. Once the entity is attached, it is managed by the data context and should not be destroyed manually.

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Attach

 Request Support DAC Forum Provide Feedback

6.14.1.1.3.5 FromKey Method

The method is designed for setting the entity key value.

Class

TEntity

Syntax

```
procedure FromKey(Key: TCustomKey);
```

Parameters

Key

The key instance containing entity key value.

Remarks

The method sets the entity key from the specified key value. The specified key has to be of the same meta-type as the entity. Otherwise, the exception will be raised. To fill a key with the entity key value, the ToKey method is used.

See Also

- ToKey

Request Support DAC Forum Provide Feedback

6.14.1.1.3.6 IsAttached Method

The method is designed to determine whether the entity is attached.

Class

TEntity

Syntax

```
function IsAttached: boolean;
```

Remarks

The method returns True if the entity is attached to the data context using the TCustomEntityContext.Attach method.

See Also

- TCustomEntityContext.Attach

Request Support DAC Forum Provide Feedback

6.14.1.1.3.7 ToKey Method

The method is designed for filling the specified key with the entity key value.

Class

TEntity

Syntax

```
procedure ToKey(Key: TCustomKey);
```

Parameters

Key
The key to be filled.

Remarks

The method fills the specified key with the entity key value. The specified key has to be of the same meta-type as the entity. Otherwise, the exception will be raised. To set the entity key with a key value, the FromKey method is used.

Request Support DAC Forum Provide Feedback

6.14.1.2 TEntityReference Class

The base class for representing the entity reference.

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

Unit

EntityDAC.Entity

Syntax

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

Remarks

The class represents the entity reference. A reference is the link from the entity to another entity in one-to-one associations, or the link to the parent entity in one-to-many associations. TEntityReference is the base class and it should not be used directly. For operating

references in the code TMappedReference class is used.

See Also

- TMappedReference

 Request Support DAC Forum Provide Feedback

6.14.1.2.1 Members

TEntityReference class overview.

Properties

Name	Description
IsModified	Indicates whether the reference is modified.
MetaReference	Contains the meta description of the reference.
Value	Provides access to the referenced entity instance.

Methods

Name	Description
MetaType	Indicates the meta-type of the entity.

 Request Support DAC Forum Provide Feedback

6.14.1.2.2 Properties

Properties of the **TEntityReference** class.

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

Public

Name	Description

IsModified	Indicates whether the reference is modified.
MetaReference	Contains the meta description of the reference.
Value	Provides access to the referenced entity instance.

See Also

- TEntityReference Class
- TEntityReference Class Members

Request Support DAC Forum Provide Feedback

6.14.1.2.2.1 IsModified Property

Indicates whether the reference is modified.

Class

TEntityReference

Syntax

```
property IsModified: boolean;
```

Remarks

The property indicates whether the reference is modified. The property is set to True when the Value property is changed: either another entity instance is assigned or the value is deleted.

See Also

- TEntity

Request Support DAC Forum Provide Feedback

6.14.1.2.2.2 MetaReference Property

Contains the meta description of the reference.

Class

TEntityReference

Syntax

```
property MetaReference: TMetaReference;
```

Remarks

The property returns the TMetaReference instance that contains the reference meta description.

See Also

- TMetaReference

6.14.1.2.2.3 Value Property

Provides access to the referenced entity instance.

Class

TEntityReference

Syntax

```
property Value: TEntity;
```

Remarks

The property provides access to the referenced entity instance.

See Also

- TEntity

Reserved.

6.14.1.2.3 Methods

Methods of the **TEntityReference** class.

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

Public

Name	Description
MetaType	Indicates the meta-type of the entity.

See Also

- TEntityReference Class
- TEntityReference Class Members

Request Support DAC Forum Provide Feedback

6.14.1.2.3.1 MetaType Method

Indicates the meta-type of the entity.

Class

TEntityReference

Syntax

```
function MetaType: TMetaType;
```

Remarks

The property indicates the meta-type of the entity, which the reference belongs to.

See Also

- TEntity
- TMetaType

 Request Support DAC Forum Provide Feedback

6.14.1.3 TEntityReferences Class

The base class for representing a list of the entity references.

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

Unit

EntityDAC.Entity

Syntax

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

Remarks

TEntityReferences contains a list of TEntityReference and provides methods for iterating and accessing the list elements. TEntityReferences is the base class and should not be used directly. For operating entity references in the code TMappedReferences class is used.

See Also

- TEntityReference
- TMappedReferences

 Request Support DAC Forum Provide Feedback

6.14.1.3.1 Members

TEntityReferences class overview.

Properties

Name	Description
Count	Indicates elements count in the list.
Items	Returns the reference by its index.

Methods

Name	Description
Find	Returns the reference by its name.

Request Support DAC Forum Provide Feedback

6.14.1.3.2 Properties

Properties of the **TEntityReferences** class.

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

Public

Name	Description
Count	Indicates elements count in the list.
Items	Returns the reference by its index.

See Also

- TEntityReferences Class
- TEntityReferences Class Members

Request Support DAC Forum Provide Feedback

6.14.1.3.2.1 Count Property

Indicates elements count in the list.

Class

`TEntityReferences`

Syntax

```
property Count: Integer;
```

Remarks

The property indicates the number of TEntityReference elements contained in the list.

See Also

- TEntityReference

 Request Support DAC Forum Provide Feedback

6.14.1.3.2.2 Items Property(Indexer)

Returns the reference by its index.

Class

TEntityReferences

Syntax

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

Parameters

Index

The index of the reference.

Remarks

The function returns the TEntityReference element by its specified index.

See Also

- TEntityReference

 Request Support DAC Forum Provide Feedback

6.14.1.3.3 Methods

Methods of the **TEntityReferences** class.

For a complete list of the **TEntityReferences** class members, see the TEntityReferences

Members topic.

Public

Name	Description
Find	Returns the reference by its name.

See Also

- TEntityReferences Class
- TEntityReferences Class Members

 Request Support DAC Forum Provide Feedback

6.14.1.3.3.1 Find Method

Returns the reference by its name.

Class

TEntityReferences

Syntax

```
function Find(const Name: string): TEntityReference;
```

Parameters

Name

The name of the reference.

Remarks

The function returns the TEntityReference element by its specified name. The name is case sensitive.

See Also

- TEntityReference

 Request Support DAC Forum Provide Feedback

6.14.1.4 TUnmappedEntity Class

The class represents an unmapped entity instance.

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

Unit

EntityDAC.Entity

Syntax

```
TUnmappedEntity = class(TEntity);
```

Remarks

TUnmappedEntity is intended to hold the instance of an entity that is the result of a query execution and can not be mapped to the particular table. TUnmappedEntity is not updatable, it can not be saved to the database or deleted. Those entities, that are mapped to a particular database table, are represented with the TMappedEntity class instances.

Inheritance Hierarchy

TEntity

TUnmappedEntity

See Also

- TEntity
- TMappedEntity

6.14.1.4.1 Members

TUnmappedEntity class overview.

Properties

Name	Description
Attributes (inherited from **TEntity**)	The property represents the entity attributes collection.

Collections (inherited from **TEntity**)	The property represents a list of the entity collections.
EntityState (inherited from **TEntity**)	The property indicates the entity state.
MetaType (inherited from **TEntity**)	The property indicates the entity meta-type.
References (inherited from **TEntity**)	The property represents a list of the entity references.
UpdateState (inherited from **TEntity**)	The property indicates the entity update state.

Methods

Name	Description
AttributeByName (inherited from **TEntity**)	Returns an entity attribute by its name.
Compare (inherited from **TEntity**)	The method is designed for comparing the entity key with the specified key.
Create (inherited from **TEntity**)	Overloaded. The constructor is designed for creating a new entity instance.
FromKey (inherited from **TEntity**)	The method is designed for setting the entity key value.
IsAttached (inherited from **TEntity**)	The method is designed to determine whether the entity is attached.
ToKey (inherited from **TEntity**)	The method is designed for filling the specified key with the entity key value.

6.15 EntityDAC.EntityAttributes

List of TEntityAttribute.

Classes

Name	Description
TEntityAttribute	A basic class responsible

	for data storing in TEntity.
TEntityAttributes	List of TEntityAttribute.

Request Support DAC Forum Provide Feedback

6.15.1 Classes

Classes in the **EntityDAC.EntityAttributes** unit.

Classes

Name	Description
TEntityAttribute	A basic class responsible for data storing in TEntity.
TEntityAttributes	List of TEntityAttribute.

Request Support DAC Forum Provide Feedback

6.15.1.1 TEntityAttribute Class

A basic class responsible for data storing in TEntity.

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

Unit

EntityDAC.EntityAttributes

Syntax

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


Request Support DAC Forum Provide Feedback

6.15.1.1.1 Members

TEntityAttribute class overview.

Properties

Name	Description
AsAnsiString	Allows to get and set the value as AnsiString.
AsAnsiStringNullable	Allows to get and set the value as AnsiString or Null.
AsBcd	Allows to get and set the value as TBCD.
AsBcdNullable	Allows to get and set the value as TBCD or Null.
AsBoolean	Allows to get and set the value as Boolean.
AsBooleanNullable	Allows to get and set the value as Boolean or Null.
AsByte	Allows to get and set the value as Byte.
AsByteNullable	Allows to get and set the value as Byte or Null.
AsBytes	Allows to get and set the value as Bytes.
AsBytesNullable	Allows to get and set the value as Bytes or Null.
AsCurrency	Allows to get and set the value as Currency.
AsCurrencyNullable	Allows to get and set the value as Currency or Null.
AsDate	Allows to get and set the value as TDate.
AsDateNullable	Allows to get and set the value as TDate or Null.
AsDateTime	Allows to get and set the value as TDateTime.
AsDateTimeNullable	Allows to get and set the value as TDateTime or Null.
AsDouble	Allows to get and set the value as Double.
AsDoubleNullable	Allows to get and set the value as Double or Null.
AsExtended	Allows to get and set the value as Extended.
AsExtendedNullable	Allows to get and set the value as Extended or Null.
AsGUID	Allows to get and set the value as TGUID.

AsGUIDNullable	Allows to get and set the value as TGUID or Null.
AsInt64	Allows to get and set the value as Int64.
AsInt64Nullable	Allows to get and set the value as Int64 or Null.
AsInteger	Allows to get and set the value as Integer.
AsIntegerNullable	Allows to get and set the value as Integer or Null.
AsLongWord	Allows to get and set the value as LongWord.
AsLongWordNullable	Allows to get and set the value as LongWord or Null.
AsShortInt	Allows to get and set the value as ShortInt.
AsShortIntNullable	Allows to get and set the value as ShortInt or Null.
AsSingle	Allows to get and set the value as Single.
AsSingleNullable	Allows to get and set the value as Single or Null.
AsSmallInt	Allows to get and set the value as SmallInt.
AsSmallIntNullable	Allows to get and set the value as SmallInt or Null.
AsString	Allows to get and set the value as String.
AsStringNullable	Allows to get and set the value as String or Null.
AsTime	Allows to get and set the value as TTime.
AsTimeNullable	Allows to get and set the value as TTime or Null.
AsTimeStamp	Allows to get and set the value as TimeStamp.
AsTimeStampNullable	Allows to get and set the value as TimeStamp or Null.
AsUInt64	Allows to get and set the value as UInt64.
AsUInt64Nullable	Allows to get and set the value as UInt64 or Null.
AsVariant	Allows to get and set the

	value as Variant.
AsWideString	Allows to get and set the value as WideString.
AsWideStringNullable	Allows to get and set the value as WideString or Null.
AsWord	Allows to get and set the value as Word.
AsWordNullable	Allows to get and set the value as Word or Null.
AsXML	Allows to get and set the value as XML.
IsModified	The property returns True if the attribute value was modified.
IsNull	The property returns True if the value is Null; otherwise, the property returns False.
MetaAttribute	Meta-information about an attribute.
Name	Attribute name.

Methods

Name	Description
Compare	Compares its own value with AValue.
FromValue	Allows to copy data from TEDValue to an attribute.
ToString	The method converts a value to string representation, so that it is suitable for display.
ToValue	Allows to save data from an attribute to TEDValue.

Request Support DAC Forum Provide Feedback

6.15.1.1.2 Properties

Properties of the **TEntityAttribute** class.

For a complete list of the **TEntityAttribute** class members, see the TEntityAttribute Members

topic.

Public

Name	Description
AsAnsiString	Allows to get and set the value as AnsiString.
AsAnsiStringNullable	Allows to get and set the value as AnsiString or Null.
AsBcd	Allows to get and set the value as TBCD.
AsBcdNullable	Allows to get and set the value as TBCD or Null.
AsBoolean	Allows to get and set the value as Boolean.
AsBooleanNullable	Allows to get and set the value as Boolean or Null.
AsByte	Allows to get and set the value as Byte.
AsByteNullable	Allows to get and set the value as Byte or Null.
AsBytes	Allows to get and set the value as Bytes.
AsBytesNullable	Allows to get and set the value as Bytes or Null.
AsCurrency	Allows to get and set the value as Currency.
AsCurrencyNullable	Allows to get and set the value as Currency or Null.
AsDate	Allows to get and set the value as TDate.
AsDateNullable	Allows to get and set the value as TDate or Null.
AsDateTime	Allows to get and set the value as TDateTime.
AsDateTimeNullable	Allows to get and set the value as TDateTime or Null.
AsDouble	Allows to get and set the value as Double.
AsDoubleNullable	Allows to get and set the value as Double or Null.
AsExtended	Allows to get and set the value as Extended.

AsExtendedNullable	Allows to get and set the value as Extended or Null.
AsGUID	Allows to get and set the value as TGUID.
AsGUIDNullable	Allows to get and set the value as TGUID or Null.
AsInt64	Allows to get and set the value as Int64.
AsInt64Nullable	Allows to get and set the value as Int64 or Null.
AsInteger	Allows to get and set the value as Integer.
AsIntegerNullable	Allows to get and set the value as Integer or Null.
AsLongWord	Allows to get and set the value as LongWord.
AsLongWordNullable	Allows to get and set the value as LongWord or Null.
AsShortInt	Allows to get and set the value as ShortInt.
AsShortIntNullable	Allows to get and set the value as ShortInt or Null.
AsSingle	Allows to get and set the value as Single.
AsSingleNullable	Allows to get and set the value as Single or Null.
AsSmallInt	Allows to get and set the value as SmallInt.
AsSmallIntNullable	Allows to get and set the value as SmallInt or Null.
AsString	Allows to get and set the value as String.
AsStringNullable	Allows to get and set the value as String or Null.
AsTime	Allows to get and set the value as TTime.
AsTimeNullable	Allows to get and set the value as TTime or Null.
AsTimeStamp	Allows to get and set the value as TimeStamp.
AsTimeStampNullable	Allows to get and set the value as TimeStamp or Null.
AsUInt64	Allows to get and set the

	value as UInt64.
AsUInt64Nullable	Allows to get and set the value as UInt64 or Null.
AsVariant	Allows to get and set the value as Variant.
AsWideString	Allows to get and set the value as WideString.
AsWideStringNullable	Allows to get and set the value as WideString or Null.
AsWord	Allows to get and set the value as Word.
AsWordNullable	Allows to get and set the value as Word or Null.
AsXML	Allows to get and set the value as XML.
IsModified	The property returns True if the attribute value was modified.
IsNull	The property returns True if the value is Null; otherwise, the property returns False.
MetaAttribute	Meta-information about an attribute.
Name	Attribute name.

See Also

- TEntityAttribute Class
- TEntityAttribute Class Members

6.15.1.1.2.1 AsAnsiString Property

Allows to get and set the value as AnsiString.

Class

`TEntityAttribute`

Syntax

```
property AsAnsiString: AnsiString;
```

Remarks

If the value cannot be converted to AnsiString, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.2 AsAnsiStringNullable Property

Allows to get and set the value as AnsiString or Null.

Class

TEntityAttribute

Syntax

```
property AsAnsiStringNullable: AnsiStringNullable;
```

Remarks

If the value cannot be converted to AnsiString, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.3 AsBcd Property

Allows to get and set the value as TBCD.

Class

TEntityAttribute

Syntax

```
property AsBcd: TBcd;
```

Remarks

If the value cannot be converted to TBCD, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.4 AsBcdNullable Property

Allows to get and set the value as TBCD or Null.

Class

TEntityAttribute

Syntax

```
property AsBcdNullable: TBcdNullable;
```

Remarks

If the value cannot be converted to TBCD, then an exception will be raised.

6.15.1.1.2.5 AsBoolean Property

Allows to get and set the value as Boolean.

Class

TEntityAttribute

Syntax

```
property AsBoolean: Boolean;
```

Remarks

If the value cannot be converted to Boolean, then an exception will be raised.

6.15.1.1.2.6 AsBooleanNullable Property

Allows to get and set the value as Boolean or Null.

Class

TEntityAttribute

Syntax

```
property AsBooleanNullable: BooleanNullable;
```

Remarks

If the value cannot be converted to Boolean, then an exception will be raised.

6.15.1.1.2.7 AsByte Property

Allows to get and set the value as Byte.

Class

TEntityAttribute

Syntax

```
property AsByte: Byte;
```

Remarks

If the value cannot be converted to Byte, then an exception will be raised.

6.15.1.1.2.8 AsByteNullable Property

Allows to get and set the value as Byte or Null.

Class

TEntityAttribute

Syntax

```
property AsByteNullable: ByteNullable;
```

Remarks

If the value cannot be converted to Byte, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.9 AsBytes Property

Allows to get and set the value as Bytes.

Class

TEntityAttribute

Syntax

```
property AsBytes: TBytes;
```

Remarks

If the value cannot be converted to Bytes, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.10 AsBytesNullable Property

Allows to get and set the value as Bytes or Null.

Class

TEntityAttribute

Syntax

```
property AsBytesNullable: TBytesNullable;
```

Remarks

If the value cannot be converted to Bytes, then an exception will be raised.

6.15.1.1.2.11 AsCurrency Property

Allows to get and set the value as Currency.

Class

TEntityAttribute

Syntax

```
property AsCurrency: Currency;
```

Remarks

If the value cannot be converted to Currency, then an exception will be raised.

6.15.1.1.2.12 AsCurrencyNullable Property

Allows to get and set the value as Currency or Null.

Class

TEntityAttribute

Syntax

```
property AsCurrencyNullable: CurrencyNullable;
```

Remarks

If the value cannot be converted to Currency, then an exception will be raised.

6.15.1.1.2.13 AsDate Property

Allows to get and set the value as TDate.

Class

TEntityAttribute

Syntax

```
property AsDate: TDate;
```

Remarks

If the value cannot be converted to TDate, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.15.1.1.2.14 AsDateNullable Property

Allows to get and set the value as TDate or Null.

Class

TEntityAttribute

Syntax

```
property AsDateNullable: TDateNullable;
```

Remarks

If the value cannot be converted to TDate, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.15.1.1.2.15 AsDateTime Property

Allows to get and set the value as TDateTime.

Class

TEntityAttribute

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

If the value cannot be converted to TDateTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.16 AsDateTimeNullable Property

Allows to get and set the value as TDateTime or Null.

Class

TEntityAttribute

Syntax

```
property AsDateTimeNullable: TDateTimeNullable;
```

Remarks

If the value cannot be converted to TDateTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.17 AsDouble Property

Allows to get and set the value as Double.

Class

TEntityAttribute

Syntax

```
property AsDouble: Double;
```

Remarks

If the value cannot be converted to Double, then an exception will be raised.

6.15.1.1.2.18 AsDoubleNullable Property

Allows to get and set the value as Double or Null.

Class

TEntityAttribute

Syntax

```
property AsDoubleNullable: DoubleNullable;
```

Remarks

If the value cannot be converted to Double, then an exception will be raised.

6.15.1.1.2.19 AsExtended Property

Allows to get and set the value as Extended.

Class

TEntityAttribute

Syntax

```
property AsExtended: Extended;
```

Remarks

If the value cannot be converted to Extended, then an exception will be raised.

6.15.1.1.2.20 AsExtendedNullable Property

Allows to get and set the value as Extended or Null.

Class

TEntityAttribute

Syntax

```
property AsExtendedNullable: ExtendedNullable;
```

Remarks

If the value cannot be converted to Extended, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.15.1.1.2.21 AsGUID Property

Allows to get and set the value as TGUID.

Class

TEntityAttribute

Syntax

```
property AsGUID: TGUID;
```

Remarks

If the value cannot be converted to TGUID, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.15.1.1.2.22 AsGUIDNullable Property

Allows to get and set the value as TGUID or Null.

Class

TEntityAttribute

Syntax

```
property AsGUIDNullable: TGUIDNullable;
```

Remarks

If the value cannot be converted to TGUID, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.23 AsInt64 Property

Allows to get and set the value as Int64.

Class

TEntityAttribute

Syntax

```
property AsInt64: Int64;
```

Remarks

If the value cannot be converted to Int64, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.24 AsInt64Nullable Property

Allows to get and set the value as Int64 or Null.

Class

TEntityAttribute

Syntax

```
property AsInt64Nullable: Int64Nullable;
```

Remarks

If the value cannot be converted to Int64, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.25 AsInteger Property

Allows to get and set the value as Integer.

Class

TEntityAttribute

Syntax

```
property AsInteger: Integer;
```

Remarks

If the value cannot be converted to Integer, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.26 AsIntegerNullable Property

Allows to get and set the value as Integer or Null.

Class

TEntityAttribute

Syntax

```
property AsIntegerNullable: IntegerNullable;
```

Remarks

If the value cannot be converted to Integer, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.27 AsLongWord Property

Allows to get and set the value as LongWord.

Class

TEntityAttribute

Syntax

```
property AsLongWord: Cardinal;
```

Remarks

If the value cannot be converted to LongWord, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.28 AsLongWordNullable Property

Allows to get and set the value as LongWord or Null.

Class

TEntityAttribute

Syntax

```
property AsLongWordNullable: LongWordNullable;
```

Remarks

If the value cannot be converted to LongWord, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.29 AsShortInt Property

Allows to get and set the value as ShortInt.

Class

TEntityAttribute

Syntax

```
property AsShortInt: ShortInt;
```

Remarks

If the value cannot be converted to ShortInt, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.30 AsShortIntNullable Property

Allows to get and set the value as ShortInt or Null.

Class

TEntityAttribute

Syntax

```
property AsShortIntNullable: ShortIntNullable;
```

Remarks

If the value cannot be converted to ShortInt, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.31 AsSingle Property

Allows to get and set the value as Single.

Class

TEntityAttribute

Syntax

```
property AsSingle: Single;
```

Remarks

If the value cannot be converted to Single, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.32 AsSingleNullable Property

Allows to get and set the value as Single or Null.

Class

TEntityAttribute

Syntax

```
property AsSingleNullable: SingleNullable;
```

Remarks

If the value cannot be converted to Single, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.33 AsSmallInt Property

Allows to get and set the value as SmallInt.

Class

TEntityAttribute

Syntax

```
property AsSmallInt: SmallInt;
```

Remarks

If the value cannot be converted to SmallInt, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.34 AsSmallIntNullable Property

Allows to get and set the value as SmallInt or Null.

Class

TEntityAttribute

Syntax

```
property AsSmallIntNullable: SmallIntNullable;
```

Remarks

If the value cannot be converted to SmallInt, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.35 AsString Property

Allows to get and set the value as String.

Class

TEntityAttribute

Syntax

```
property AsString: string;
```

Remarks

If the value cannot be converted to String, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.36 AsStringNullable Property

Allows to get and set the value as String or Null.

Class

TEntityAttribute

Syntax

```
property AsStringNullable: StringNullable;
```

Remarks

If the value cannot be converted to String, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.37 AsTime Property

Allows to get and set the value as TTime.

Class

TEntityAttribute

Syntax

```
property AsTime: TTime;
```

Remarks

If the value cannot be converted to TTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.38 AsTimeNullable Property

Allows to get and set the value as TTime or Null.

Class

TEntityAttribute

Syntax

```
property AsTimeNullable: TTimeNullable;
```

Remarks

If the value cannot be converted to TTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.39 AsTimeStamp Property

Allows to get and set the value as TimeStamp.

Class

TEntityAttribute

Syntax

```
property AsTimeStamp: TSQLTimeStamp;
```

Remarks

If the value cannot be converted to TimeStamp, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.40 AsTimeStampNullable Property

Allows to get and set the value as TimeStamp or Null.

Class

TEntityAttribute

Syntax

```
property AsTimeStampNullable: TSQLTimeStampNullable;
```

Remarks

If the value cannot be converted to TimeStamp, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.41 AsUInt64 Property

Allows to get and set the value as UInt64.

Class

TEntityAttribute

Syntax

```
property AsUInt64: UInt64;
```

Remarks

If the value cannot be converted to UInt64, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.15.1.1.2.42 AsUInt64Nullable Property

Allows to get and set the value as UInt64 or Null.

Class

TEntityAttribute

Syntax

```
property AsUInt64Nullable: UInt64Nullable;
```

Remarks

If the value cannot be converted to UInt64, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.15.1.1.2.43 AsVariant Property

Allows to get and set the value as Variant.

Class

TEntityAttribute

Syntax

```
property AsVariant: Variant;
```

Remarks

If the value cannot be converted to Variant, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.44 AsWideString Property

Allows to get and set the value as WideString.

Class

TEntityAttribute

Syntax

```
property AsWideString: string;
```

Remarks

If the value cannot be converted to WideString, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.45 AsWideStringNullable Property

Allows to get and set the value as WideString or Null.

Class

TEntityAttribute

Syntax

```
property AsWideStringNullable: WideStringNullable;
```

Remarks

If the value cannot be converted to WideString, then an exception will be raised.

6.15.1.1.2.46 AsWord Property

Allows to get and set the value as Word.

Class

TEntityAttribute

Syntax

```
property AsWord: Word;
```

Remarks

If the value cannot be converted to Word, then an exception will be raised.

6.15.1.1.2.47 AsWordNullable Property

Allows to get and set the value as Word or Null.

Class

TEntityAttribute

Syntax

```
property AsWordNullable: WordNullable;
```

Remarks

If the value cannot be converted to Word, then an exception will be raised.

6.15.1.1.2.48 AsXML Property

Allows to get and set the value as XML.

Class

TEntityAttribute

Syntax

```
property AsXML: IXMLDocument;
```

Remarks

If the value cannot be converted to XML, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.15.1.1.2.49 IsModified Property

The property returns True if the attribute value was modified.

Class

TEntityAttribute

Syntax

```
property IsModified: boolean;
```


Request Support DAC Forum Provide Feedback

6.15.1.1.2.50 IsNull Property

The property returns True if the value is Null; otherwise, the property returns False.

Class

TEntityAttribute

Syntax

```
property IsNull: Boolean;
```

Remarks

The value can be set to Null by assigning the True value to IsNull. The False value can't be assigned to IsNull. IsNull can be set to False only by assigning a particular value to the Value property.

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.51 MetaAttribute Property

Meta-information about an attribute.

Class

TEntityAttribute

Syntax

```
property MetaAttribute: TMetaAttribute;
```

 Request Support DAC Forum Provide Feedback

6.15.1.1.2.52 Name Property

Attribute name.

Class

TEntityAttribute

Syntax

```
property Name: string;
```

 Request Support DAC Forum Provide Feedback

6.15.1.1.3 Methods

Methods of the **TEntityAttribute** class.

For a complete list of the **TEntityAttribute** class members, see the TEntityAttribute Members

topic.

Public

Name	Description
Compare	Compares its own value with AValue.
FromValue	Allows to copy data from TEDValue to an attribute.
ToString	The method converts a value to string representation, so that it is suitable for display.
ToValue	Allows to save data from an attribute to TEDValue.

See Also

- TEntityAttribute Class
- TEntityAttribute Class Members

Request Support DAC Forum Provide Feedback

6.15.1.1.3.1 Compare Method

Compares its own value with AValue.

Class

TEntityAttribute

Syntax

```
function Compare(AValue: TEDValue): Integer;
```

Parameters

AValue

Remarks

If Self = AValue - returns '0';

If Self > AValue - returns '1';

If Self < AValue - returns '-1';

6.15.1.1.3.2 FromValue Method

Allows to copy data from TEDValue to an attribute.

Class

TEntityAttribute

Syntax

```
procedure FromValue(AValue: TEDValue);
```

Parameters

AValue

6.15.1.1.3.3 ToString Method

The method converts a value to string representation, so that it is suitable for display.

Class

TEntityAttribute

Syntax

```
function ToString: string; override;
```


6.15.1.1.3.4 ToValue Method

Allows to save data from an attribute to TEDValue.

Class

TEntityAttribute

Syntax

```
procedure ToValue(AValue: TEDValue);
```

Parameters

AValue

 Request Support DAC Forum Provide Feedback

6.15.1.2 TEntityAttributes Class

List of TEntityAttribute.

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

Unit

EntityDAC.EntityAttributes

Syntax

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

 Request Support DAC Forum Provide Feedback

6.15.1.2.1 Members

TEntityAttributes class overview.

Properties

Name	Description
Count	Returns the number of values in the list.
Items	Lists the attributes in the list.

 Request Support DAC Forum Provide Feedback

6.15.1.2.2 Properties

Properties of the **TEntityAttributes** class.

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

Public

Name	Description
Count	Returns the number of values in the list.
Items	Lists the attributes in the list.

See Also

- TEntityAttributes Class
- TEntityAttributes Class Members

 Request Support DAC Forum Provide Feedback

6.15.1.2.2.1 Count Property

Returns the number of values in the list.

Class

TEntityAttributes

Syntax

```
property Count: Integer;
```

 Request Support DAC Forum Provide Feedback

6.15.1.2.2.2 Items Property(Indexer)

Lists the attributes in the list.

Class

TEntityAttributes

Syntax

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

Parameters

Index

6.16 EntityDAC.EntityConnection

The unit contains implementation of the connection component functionality.

Classes

Name	Description
TEntityConnection	Encapsulates a EntityDAC connection to a database.

6.16.1 Classes

Classes in the **EntityDAC.EntityConnection** unit.

Classes

Name	Description
TEntityConnection	Encapsulates a EntityDAC connection to a database.

6.16.1.1 TEntityConnection Class

Encapsulates a EntityDAC connection to a database.

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

Unit

EntityDAC.EntityConnection

Syntax

```
TEntityConnection = class(TComponent);
```

Remarks

The component is designed to represent a EntityDAC database connection. The connection provided by a single TEntityConnection component can be shared by multiple data context components through their Connection properties.

See Also

- TEntityContext
- TObjectContext
- TDataContext

Request Support DAC Forum Provide Feedback

6.16.1.1.1 Members

TEntityConnection class overview.

Properties

Name	Description
Connected	Determines whether a connection has been established.
ConnectionString	Specifies the connection parameters for the connection.
DefaultModel	Specifies the default meta model for all associated data contexts.
DefaultModelName	Specifies the name of the default meta model for all associated data contexts.

Dialect	Specifies the current SQL dialect used by the data provider.
DialectName	Specifies the name of the current SQL dialect used by the data provider.
InTransaction	Indicates whether a transaction is already in progress.
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Provider	Defines the current data provider used for establishing a connection.
ProviderName	Specifies the name of the current data provider used for establishing a connection.

Methods

Name	Description
CommitTransaction	Commits the current transaction.
Connect	Overloaded.Description is not available at the moment.
CreateDatabase	Create all database objects for the specified meta model.
Disconnect	Closes the connection.
DropDatabase	Drop all database objects used by the specified meta model.
ExecuteCursor	Overloaded.Description is not available at the moment.
ExecuteSQL	Overloaded.Description is not available at the moment.
RollbackTransaction	Discards all current data changes and ends transaction.

StartTransaction	Initiates a new transaction in the associated database.

Events

Name	Description
AfterConnect	Occurs immediately after establishing a connection.
AfterDisconnect	Occurs immediately after closing a connection.
BeforeConnect	Occurs just before establishing a connection.
BeforeDisconnect	Occurs just before closing a connection.

Request Support DAC Forum Provide Feedback

6.16.1.1.2 Properties

Properties of the **TEntityConnection** class.

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

Public

Name	Description
DefaultModel	Specifies the default meta model for all associated data contexts.
Dialect	Specifies the current SQL dialect used by the data provider.
InTransaction	Indicates whether a transaction is already in progress.
Provider	Defines the current data provider used for establishing a connection.

Published

Name	Description
Connected	Determines whether a connection has been established.
ConnectionString	Specifies the connection parameters for the connection.
DefaultModelName	Specifies the name of the default meta model for all associated data contexts.
DialectName	Specifies the name of the current SQL dialect used by the data provider.
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
ProviderName	Specifies the name of the current data provider used for establishing a connection.

See Also

- TEntityConnection Class
- TEntityConnection Class Members

6.16.1.1.2.1 Connected Property

Determines whether a connection has been established.

Class

TEntityConnection

Syntax

```
property Connected: boolean default False;
```

Remarks

Read the Connected property value to determine the current status of a database connection. If Connected is True, the database connection is active; if False, then the connection is inactive. Set Connected to True to open the connection. Set Connected to False to terminate the connection.

The database to which the connection is established, is specified by ProviderName and DialectName properties. The connection parameters are passed through the ConnectionString property.

Setting Connected to True generates a BeforeConnect event, establishes the connection, and generates an AfterConnect event. In addition, when setting Connected to True TEntityConnection may display a login dialog, depending on the value of LoginPrompt. Setting Connected to False generates a BeforeDisconnect event, drops the connection, and generates an AfterDisconnect event.

Also, the Connected property value is toggled when Connect and Disconnect methods are executed.

See Also

- ProviderName
- DialectName
- ConnectionString
- LoginPrompt
- Connect
- Disconnect
- BeforeConnect
- AfterConnect
- BeforeDisconnect
- AfterDisconnect

6.16.1.1.2.2 ConnectionString Property

Specifies the connection parameters for the connection.

Class

TEntityConnection

Syntax

```
property ConnectionString: string;
```

Remarks

Set ConnectionString to specify the information needed to connect the connection component to the database. The value used for ConnectionString consists of one or more parameters used to establish the connection. Specify multiple parameters as a list with individual parameters separated by semicolons.

Particular parameters set specified in the ConnectionString depends on the database to which the connection is established. The database is specified by ProviderName and DialectName properties.

Description of specific connection string parameters for all supported data providers you can find in the connectionstring.htm article.

See Also

- ProviderName
- DialectName

6.16.1.1.2.3 DefaultModel Property

Specifies the default meta model for all associated data contexts.

Class

TEntityConnection

Syntax

```
property DefaultModel: TMetaModel;
```

Remarks

The property defines the default meta model which is used by data contexts associated with the connection.

Read the DefaultModel property to access the instance of the default meta model. To set the default model, the DefaultModelName property is used.

A data context can override the default connection settings and use different meta model by setting the TCustomContext.ModelName property.

See Also

- DefaultModelName
- TCustomContext
- TCustomContext.ModelName

6.16.1.1.2.4 DefaultModelName Property

Specifies the name of the default meta model for all associated data contexts.

Class

TEntityConnection

Syntax

```
property DefaultModelName: string;
```

Remarks

The property defines the name of the default meta model which is used by data contexts associated with the connection.

Read the DefaultModelName property to determine the name of the current meta model. Set the name of the property to specify the current meta model. When the valid model name is set, the corresponding meta model instance can be accessed through the DefaultModel

property.

A data context can override the default connection settings and use different meta model by setting the TCustomContext.ModelName property.

See Also

- DefaultModel
- TCustomContext
- TCustomContext.ModelName

Request Support DAC Forum Provide Feedback

6.16.1.1.2.5 Dialect Property

Specifies the current SQL dialect used by the data provider.

Class

TEntityConnection

Syntax

```
property Dialect: TSQLDialect;
```

Remarks

A SQL dialect defines a database-specific behavior for the current data provider. Since some data providers are designed for interacting with universal sets of data access components, such as UniDAC, the SQLDialect property specifies, which exact database is used.

Read the Dialect property to access the instance of the current SQL dialect. To set the current dialect, the DialectName property is used. The TDataProvider.MultiDialect property is used to determine, whether the data provider supports working with several SQL dialects.

See Also

- DialectName
- TDataProvider.MultiDialect
- ProviderName

- Provider
- TDataProvider

6.16.1.1.2.6 DialectName Property

Specifies the name of the current SQL dialect used by the data provider.

Class

TEntityConnection

Syntax

```
property DialectName: string;
```

Remarks

A SQL dialect defines a database-specific behavior for the current data provider. Since some data providers are designed for interacting with universal sets of data access components, such as UniDAC, the DialectName property has to specify, which exact database is used.

Read the DialectName property to determine the name of the current SQL dialect. Set the name of the property to specify the current dialect name. When the data provider supports the only SQL dialect, the manual setting the DialectName property value has no effect. The TDataProvider.MultiDialect property is used to determine, whether the data provider supports working with several SQL dialects.

When the valid dialect name is set, the corresponding SQL dialect instance can be accessed through the Dialect property.

See Also

- Dialect
- TDataProvider.MultiDialect
- ProviderName
- Provider
- TDataProvider

 Request Support DAC Forum Provide Feedback

6.16.1.1.2.7 InTransaction Property

Indicates whether a transaction is already in progress.

Class

TEntityConnection

Syntax

```
property InTransaction: boolean;
```

Remarks

Read the InTransaction property to determine a transaction status. When the property value is True, it means that a transaction is already in progress. When an active transaction exists, a subsequent call to StartTransaction without first calling CommitTransaction or M:Devart.EntityDAC.TEntityConnection.RollbackTransaction to end the current transaction raises an exception.

See Also

- StartTransaction
- CommitTransaction
- RollbackTransaction

 Request Support DAC Forum Provide Feedback

6.16.1.1.2.8 LoginPrompt Property

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

Class

TEntityConnection

Syntax

```
property LoginPrompt: boolean default True;
```

Remarks

When the property value is set to True, a login dialog appears immediately before opening a new connection. EntityDAC does not have its own login dialog, therefore a default login dialog of the current data access components set is used.

Request Support DAC Forum Provide Feedback

6.16.1.1.2.9 Provider Property

Defines the current data provider used for establishing a connection.

Class

TEntityConnection

Syntax

```
property Provider: TDataProvider;
```

Remarks

Since EntityDAC is abstracted from the data access layer, direct interaction with the database is handled by specialized component packages such as Devart Data Access Components. A data provider is the specialized class that provides interaction between TEntityConnection and used data access components.

Read the DataProvider property value to access the current data provider instance. Setting the property value can be used to set the current provider at run-time. But in most cases, the ProviderName property is more suitable for setting the current provider because all data providers used in an application are registered by the data provider manager, and available through their names.

To define a database-specific behavior of the current data provider, DialectName and Dialect properties are used.

See Also

- ProviderName

- DialectName
- Dialect
- TDataProvider

 Request Support DAC Forum Provide Feedback

6.16.1.1.2.10 ProviderName Property

Specifies the name of the current data provider used for establishing a connection.

Class

TEntityConnection

Syntax

```
property ProviderName: string;
```

Remarks

Since EntityDAC is abstracted from the data access layer, direct interaction with the database is handled by specialized component packages such as Devart Data Access Components. A data provider is the specialized class that provides interaction between TEntityConnection and used data access components. All data providers used in an application are registered by the data provider manager, and available through their names.

Read the property value to determine the current provider name. Set the ProviderName property value to specify the current data provider name.

When a valid provider name is set, the corresponding data provider instance can be accessed through the Provider property. And vice versa, when the P:Devart.EntityDAC.TEntityConnection.DataProvider property is set, the corresponding provider name is assigned to the ProviderName property.

To define a database-specific behavior of the current data provider, DialectName and Dialect properties are used.

See Also

- Provider

- DialectName
- Dialect
- TDataProvider

Request Support DAC Forum Provide Feedback

6.16.1.1.3 Methods

Methods of the **TEntityConnection** class.

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

Public

Name	Description
CommitTransaction	Commits the current transaction.
Connect	Overloaded.Description is not available at the moment.
CreateDatabase	Create all database objects for the specified meta model.
Disconnect	Closes the connection.
DropDatabase	Drop all database objects used by the specified meta model.
ExecuteCursor	Overloaded.Description is not available at the moment.
ExecuteSQL	Overloaded.Description is not available at the moment.
RollbackTransaction	Discards all current data changes and ends transaction.
StartTransaction	Initiates a new transaction in the associated database.

See Also

- TEntityConnection Class

- TEntityConnection Class Members

 Request Support DAC Forum Provide Feedback

6.16.1.1.3.1 CommitTransaction Method

Commits the current transaction.

Class

TEntityConnection

Syntax

```
procedure CommitTransaction;
```

Remarks

Call the CommitTransaction 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. To cancel all changes made within the current transaction, the RollbackTransaction method is used. To check whether the current transaction is active, the InTransaction property is used. When no active transactions present, the CommitTransaction method raises an exception.

See Also

- StartTransaction
- RollbackTransaction
- InTransaction

 Request Support DAC Forum Provide Feedback

6.16.1.1.3.2 Connect Method

Class

TEntityConnection

Overload List

Name	Description
Connect	Initiates a connection to a database.
Connect(**const** AConnectionString: **string**)	Initiates a connection to a database using the specified connection parameters.
Connect(**const** AProviderName: **string**; **const** ADialectName: **string**; **const** AConnectionString: **string**)	Initiates a connection to the specified database using the specified connection parameters.

Initiates a connection to a database.

Class

TEntityConnection

Syntax

```
procedure Connect; overload;
```

Remarks

Call Connect to initiate a connection to the database.

The database to which the connection is established, is specified by TEntityConnection.ProviderName and TEntityConnection.DialectName properties. The connection parameters are passed through the TEntityConnection.ConnectionString property.

Calling Connect generates a TEntityConnection.BeforeConnect event, establishes the connection, sets the TEntityConnection.Connected property to True, and generates an TEntityConnection.AfterConnect event. In addition, when calling Connect TEntityConnection may display a login dialog, depending on the value of TEntityConnection.LoginPrompt.

See Also

- TEntityConnection.ProviderName
- TEntityConnection.DialectName
- TEntityConnection.ConnectionString

- TEntityConnection.LoginPrompt
- TEntityConnection.BeforeConnect
- TEntityConnection.AfterConnect

Request Support DAC Forum Provide Feedback

Initiates a connection to a database using the specified connection parameters.

Class

`TEntityConnection`

Syntax

```
procedure Connect(const AConnectionString: string); overload;
```

Parameters

AConnectionString
Specifies the connection parameters.

Remarks

Call Connect to initiate a connection to the database.

The database to which the connection is established, is specified by TEntityConnection.ProviderName and TEntityConnection.DialectName properties.

Calling Connect generates a TEntityConnection.BeforeConnect event, establishes the connection, sets the TEntityConnection.Connected property to True, and generates an TEntityConnection.AfterConnect event. In addition, when calling Connect TEntityConnection may display a login dialog, depending on the value of TEntityConnection.LoginPrompt.

See Also

- TEntityConnection.ProviderName
- TEntityConnection.DialectName
- TEntityConnection.ConnectionString
- TEntityConnection.LoginPrompt
- TEntityConnection.BeforeConnect

- TEntityConnection.AfterConnect

Initiates a connection to the specified database using the specified connection parameters.

Class

TEntityConnection

Syntax

```
procedure Connect(const AProviderName: string; const
ADialectName: string; const AConnectionString: string);
overload;
```

Parameters

AProviderName
Specifies the current data provider.

ADialectName
Specifies the current SQL dialect.

AConnectionString
Specifies the connection parameters.

Remarks

Call Connect to initiate a connection to the database.

Calling Connect generates a TEntityConnection.BeforeConnect event, establishes the connection, sets the TEntityConnection.Connected property to True, and generates an TEntityConnection.AfterConnect event. In addition, when calling Connect TEntityConnection may display a login dialog, depending on the value of TEntityConnection.LoginPrompt.

See Also

- TEntityConnection.ProviderName
- TEntityConnection.DialectName
- TEntityConnection.ConnectionString
- TEntityConnection.LoginPrompt

- TEntityConnection.BeforeConnect
- TEntityConnection.AfterConnect

6.16.1.1.3.3 CreateDatabase Method

Create all database objects for the specified meta model.

Class

TEntityConnection

Syntax

```
procedure CreateDatabase(const Model: TMetaModel = nil; const
Options: TDatabaseModifyOptions = []);
```

Parameters

Model

Options

Specifies the database creation options.

Remarks

Call CreateDatabase to automatically create all database objects needed for the specified meta model. When executing the method, the connection has to be already connected and no active transactions have to be present. Otherwise, the appropriate exceptions will be raised.

Calling CreateDatabase generates a set of DDL expressions depending on the specified meta model and executes the expressions one by one using the ExecuteSQL method.

The MetaModel parameter specifies the meta model for which the database need to be created. When MetaModel is set to Nil, the default connection meta model specified by the DefaultModelName property is used.

The Options parameter specifies additional options of the database creation process. When the moCommitEachStatement is specified in Options, each DDL statement executed within a separate transaction. Otherwise, the only transaction starts in the beginning of the method execution and commits after all statements executed. When the moIgnoreErrors is specified in options, then when error is occurred during the method execution, no error message will be

shown and the execution continued. Otherwise, the execution will be stopped and the appropriate exception raised.

To drop all database objects used by the meta model, the DropDatabase method is used.

See Also

- ExecuteSQL
- DefaultModelName
- DropDatabase

6.16.1.1.3.4 Disconnect Method

Closes the connection.

Class

TEntityConnection

Syntax

```
procedure Disconnect;
```

Remarks

Call Disconnect to close the database connection.

Calling Disconnect generates a BeforeDisconnect event, drops the connection, sets the Connected property to False, and generates an AfterDisconnect event.

See Also

- Connected
- BeforeDisconnect
- AfterDisconnect

6.16.1.1.3.5 DropDatabase Method

Drop all database objects used by the specified meta model.

Class

TEntityConnection

Syntax

```
procedure DropDatabase(const Model: TMetaModel = nil; const
Options: TDatabaseModifyOptions = []);
```

Parameters

Model

Options

Specifies the database deletion options.

Remarks

Call DropDatabase to automatically drop all database objects used by the specified meta model. When executing the method, the connection has to be already connected and no active transactions have to be present. Otherwise, the appropriate exceptions will be raised.

Calling DropDatabase generates a set of DDL expressions depending on the specified meta model and executes the expressions one by one using the ExecuteSQL method.

The MetaModel parameter specifies the meta model for which the database need to be dropped. When MetaModel is set to Nil, the default connection meta model specified by the DefaultModelName property is used.

The Options parameter specifies additional options of the database deletion process. When the moCommitEachStatement is specified in Options, each DDL statement executed within a separate transaction. Otherwise, the only transaction starts in the beginning of the method execution and commits after all statements executed. When the moIgnoreErrors is specified in options, then when error is occurred during the method execution, no error message is shown and the execution continued. Otherwise, the execution is stopped and the appropriate exception raised.

To create all database objects needed for the meta model, the CreateDatabase method is used.

See Also

- ExecuteSQL
- DefaultModelName
- CreateDatabase

 Request Support DAC Forum Provide Feedback

6.16.1.1.3.6 ExecuteCursor Method

Class

TEntityConnection

Overload List

Name	Description
ExecuteCursor(SQLStatement: ISQLStatement)	Executes a Cursor directly in the database.
ExecuteCursor(**const** SQL: **string**; Params: TEDParams)	Executes a Cursor directly in the database.

 Request Support DAC Forum Provide Feedback

Executes a Cursor directly in the database.

Class

TEntityConnection

Syntax

```
function ExecuteCursor(SQLStatement: ISQLStatement): IEDCursor;
overload;
```

Parameters

SQLStatement

Specifies the Cursor to be executed and its parameters.

Remarks

Call the ExecuteCursor method to execute a Cursor directly in the database. Supply the statement as a TSQLStatement class instance, which encapsulates the SQL statement text and its parameters. The TSQLStatement.Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the method.

See Also

-
- TSQLStatement
- TSQLStatement.Params

Request Support DAC Forum Provide Feedback

Executes a Cursor directly in the database.

Class

TEntityConnection

Syntax

```
function ExecuteCursor(const SQL: string; Params: TEDParams = nil): IEDCursor; overload;
```

Parameters

SQL

Specifies the Cursor to be executed.

Params

Specifies the collection of the statement parameters.

Remarks

Call the ExecuteCursor method to execute a Cursor directly in the database. Supply the Params collection with the parameters accordingly to the ones in the Cursor which itself is passed in the SQL string parameter. The TSQLStatement.Params array must contain all IN and OUT parameters defined in the Cursor. For OUT parameters provide any values of valid

types so that they are explicitly defined before call to the method.

See Also

- ExecuteCursor
- TSQLStatement
- TSQLStatement.Params

Request Support DAC Forum Provide Feedback

6.16.1.1.3.7 ExecuteSQL Method

Class

TEntityConnection

Overload List

Name	Description
ExecuteSQL(SQLStatement: ISQLStatement)	Executes a SQL statement directly in the database.
ExecuteSQL(**const** SQL: **string**; Params: TEDParams)	Executes a SQL statement directly in the database.

Request Support DAC Forum Provide Feedback

Executes a SQL statement directly in the database.

Class

TEntityConnection

Syntax

```
procedure ExecuteSQL(SQLStatement: ISQLStatement); overload;
```

Parameters

SQLStatement

Specifies the SQL statement to be executed and its parameters.

Remarks

Call the ExecuteSQL method to execute a SQL statement directly in the database. Supply the statement as a TSQLStatement class instance, which encapsulates the SQL statement text and its parameters. The TSQLStatement.Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the method.

See Also

- TSQLStatement
- TSQLStatement.Params

Request Support DAC Forum Provide Feedback

Executes a SQL statement directly in the database.

Class

TEntityConnection

Syntax

```
procedure ExecuteSQL(const SQL: string; Params: TEDParams =
nil); overload;
```

Parameters

SQL

Specifies the SQL statement to be executed.

Params

Specifies the collection of the statement parameters.

Remarks

Call the ExecuteSQL method to execute a SQL statement directly in the database. Supply the Params collection with the parameters accordingly to the ones in the SQL statement which itself is passed in the SQL string parameter. The TSQLStatement.Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the method.

See Also

- TSQLStatement
- TSQLStatement.Params

Request Support DAC Forum Provide Feedback

6.16.1.1.3.8 RollbackTransaction Method

Discards all current data changes and ends transaction.

Class

TEntityConnection

Syntax

```
procedure RollbackTransaction;
```

Remarks

Call the RollbackTransaction method to discard all data modifications 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. To save all changes made within the current transaction, the CommitTransaction method is used. To check whether the current transaction is active, the InTransaction property is used. When no active transactions present, the RollbackTransaction method raises an exception.

See Also

- StartTransaction
- CommitTransaction
- InTransaction

Request Support DAC Forum Provide Feedback

6.16.1.1.3.9 StartTransaction Method

Initiates a new transaction in the associated database.

Class

TEntityConnection

Syntax

```
procedure StartTransaction;
```

Remarks

Call the StartTransaction method to begin a new transaction at the server. Before calling StartTransaction, an application should check the value of the InTransaction property. If the result is True, it means that a transaction is already in progress, a subsequent call to StartTransaction without first calling CommitTransaction or RollbackTransaction to end the current transaction raises an exception. Calling StartTransaction when connection is closed also raises an exception.

All data modifications that take place after a call to StartTransaction are held by the server until the application calls CommitTransaction to save the changes or RollbackTransaction to cancel them.

If the transaction is successfully started, the InTransaction property is set to True.

EntityDAC does not support working with multiple transactions.

See Also

- CommitTransaction
- RollbackTransaction
- InTransaction

6.16.1.1.4 Events

Events of the **TEntityConnection** class.

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

Published

Name	Description
AfterConnect	Occurs immediately after establishing a connection.
AfterDisconnect	Occurs immediately after closing a connection.
BeforeConnect	Occurs just before establishing a connection.
BeforeDisconnect	Occurs just before closing a connection.

See Also

- TEntityConnection Class
- TEntityConnection Class Members

6.16.1.1.4.1 AfterConnect Event

Occurs immediately after establishing a connection.

Class

TEntityConnection

Syntax

```
property AfterConnect: TNotifyEvent;
```

Remarks

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a database connection.

When theConnect method is called or the Connected property is set to True, a BeforeConnect event is generated, the connection is established, and then an AfterConnect

event is generated. In the AfterConnect event handler, the Connected property value is True.

See Also

- Connect
- Connected
- BeforeConnect

Request Support DAC Forum Provide Feedback

6.16.1.1.4.2 AfterDisconnect Event

Occurs immediately after closing a connection.

Class

TEntityConnection

Syntax

```
property AfterDisconnect: TNotifyEvent;
```

Remarks

Write a AfterDisconnect event handler to take application-specific actions immediately after the connection component closes a database connection.

When the Disconnect method is called or the Connected property is set to False, a BeforeDisconnect event is generated, the connection is closed, and then an AfterDisconnect event is generated. In the AfterDisconnect event handler, the Connected property value is False.

See Also

- Disconnect
- Connected
- BeforeDisconnect

Request Support DAC Forum Provide Feedback

6.16.1.1.4.3 BeforeConnect Event

Occurs just before establishing a connection.

Class

TEntityConnection

Syntax

```
property BeforeConnect: TNotifyEvent;
```

Remarks

Write a BeforeConnect event handler to take application-specific actions just before the connection component opens a database connection.

When the Connect method is called or the Connected property is set to True, a BeforeConnect event is generated, the connection is established, and then an AfterConnect event is generated. In the BeforeConnect event handler, the Connected property value is still False.

See Also

- Connect
- Connected
- AfterConnect

6.16.1.1.4.4 BeforeDisconnect Event

Occurs just before closing a connection.

Class

TEntityConnection

Syntax

```
property BeforeDisconnect: TNotifyEvent;
```

Remarks

Write a BeforeDisconnect event handler to take application-specific actions just before the connection component closes a database connection.

When the Disconnect method is called or the Disconnect property is set to False, a BeforeDisconnect event is generated, the connection is closed, and then an AfterDisconnect event is generated. In the BeforeDisconnect event handler, the Connected property value is still True.

See Also

- Disconnect
- Connected
- AfterDisconnect

Request Support DAC Forum Provide Feedback

6.17 EntityDAC.EntityContext

This unit contains implementation of objects lifecycle management.

Classes

Name	Description
TCustomEntityContext	The base class that provides the data context functionality.
TEntityCollectionUpdater	The base class for representing a list of the entity collections.
TEntityContext	The class that provides the data context functionality.
TMappedCollections	Represents a list of the entity collections.
TMappedEntity	The class that represents a mapped entity instance and provides methods for managing it.
TMappedReference	Represents the entity reference.

TMappedReferences	Represents a list of the entity references.

Types

Name	Description
TMappedEntityClass	The class that represents a mapped entity instance and provides methods for managing its.

 Request Support DAC Forum Provide Feedback

6.17.1 Classes

Classes in the **EntityDAC.EntityContext** unit.

Classes

Name	Description
TCustomEntityContext	The base class that provides the data context functionality.
TEntityCollectionUpdater	The base class for representing a list of the entity collections.
TEntityContext	The class that provides the data context functionality.
TMappedCollections	Represents a list of the entity collections.
TMappedEntity	The class that represents a mapped entity instance and provides methods for managing it.
TMappedReference	Represents the entity reference.
TMappedReferences	Represents a list of the entity references.

 Request Support DAC Forum Provide Feedback

6.17.1.1 TCustomEntityContext Class

The base class that provides the data context functionality.

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

Unit

EntityDAC.EntityContext

Syntax

```
TCustomEntityContext = class(TDataContext);
```

Remarks

TCustomEntityContext class provides functionality for managing an entity life cycle in the application. It provides methods for creating and initializing new entity instances, retrieving and storing entities from/to the database, storing used entities in the cache for future use, destroying of unused entities.

Since TCustomEntityContext is the base class, it should not be used directly. Instead, TCustomEntityContext descendants such as TEntityContext have to be used.

Inheritance Hierarchy

TCustomContext
  TDataContext
    TCustomEntityContext

See Also

- TEntityContext

6.17.1.1.1 Members

TCustomEntityContext class overview.

Properties

Name	Description
Connection (inherited from **TCustomContext**)	Identifies the connection component with which the data context is associated.
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
ModelName (inherited from **TCustomContext**)	Specifies the name of the meta model used by the data context.
Types (inherited from **TDataContext**)	The property is designed to determine a meta-type by a meta-type name.

Methods

Name	Description
Attach	The method is designed for attaching an entity instance to the data context and storing it in the object cache
Cancel	The method is designed to cancel changes made in an entity instance.
Create (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
CreateAttachedEntity	Overloaded.Description is not available at the moment.
CreateAttachedEntity<T>	Overloaded.Description is not available at the moment.
CreateEntity	Overloaded.Description is not available at the moment.
CreateEntity<T>	Overloaded.Description is not available at the moment.
Delete	The method is designed for deleting an entity.
DeleteAndSave	The method is designed for permanent deleting an entity.
ExecuteQuery<T> (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
ExecuteSQL (inherited from **TCustomContext**)	Overloaded.Description is

	not available at the moment.
GetEntities	Overloaded.Description is not available at the moment.
GetEntities<T>	Overloaded.Description is not available at the moment.
GetEntity	Overloaded.Description is not available at the moment.
GetEntity<T>	Overloaded.Description is not available at the moment.
IsAttached	The method is designed to check whether the specified entity is attached to the data context.
RejectChanges (inherited from **TDataContext**)	The method is designed to cancel changes in all attached entities
Save	The method is designed for saving changes made in an entity instance.
SubmitChanges (inherited from **TDataContext**)	The method is designed for saving changes in all attached entities.

Events

Name	Description
OnGetGeneratorValue (inherited from **TCustomContext**)	Occurs when an entity attribute value generator of type "custom" needs to generate its value.

6.17.1.1.2 Methods

Methods of the **TCustomEntityContext** class.

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

Public

Name	Description
Attach	The method is designed for attaching an entity instance to the data context and storing it in the object cache
Cancel	The method is designed to cancel changes made in an entity instance.
Create (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
CreateAttachedEntity	Overloaded.Description is not available at the moment.
CreateAttachedEntity<T>	Overloaded.Description is not available at the moment.
CreateEntity	Overloaded.Description is not available at the moment.
CreateEntity<T>	Overloaded.Description is not available at the moment.
Delete	The method is designed for deleting an entity.
DeleteAndSave	The method is designed for permanent deleting an entity.
GetEntities	Overloaded.Description is not available at the moment.
GetEntities<T>	Overloaded.Description is not available at the moment.
GetEntity	Overloaded.Description is not available at the moment.
GetEntity<T>	Overloaded.Description is not available at the moment.
IsAttached	The method is designed to check whether the specified entity is attached to the data context.
RejectChanges (inherited from **TDataContext**)	The method is designed to cancel changes in all attached entities
Save	The method is designed for saving changes made in an entity instance.
SubmitChanges (inherited from **TDataContext**)	The method is designed for saving changes in all attached entities.

See Also

- TCustomEntityContext Class
- TCustomEntityContext Class Members

6.17.1.1.2.1 Attach Method

The method is designed for attaching an entity instance to the data context and storing it in the object cache

Class

TCustomEntityContext

Syntax

```
procedure Attach(Entity: TMappedEntity);
```

Parameters

Entity

The entity instance that has to be attached to the data context.

Remarks

The method attaches an entity instance created with TCustomEntityContext.CreateEntity to the data context and places it to the object cache. As a result of this method execution, several aims are reached:

- on subsequent calls to the same entity (e.g., using the TCustomEntityContext.GetEntity method), an instance saved in the cache will be returned, that eliminates re-accessing a database and significantly increases performance;
- there appears a possibility to perform modification operations for the entity instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel,that will project instance modifications in appropriate database structures;
- the data context takes control of entity instance destruction on application shutdown, that

eliminates the need to destroy objects manually.

Existence of separate TCustomEntityContext.CreateEntity and TCustomEntityContext.Attach methods is due to the check of uniqueness of the entity primary key value on saving it to the object cache, therefore it is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness).

If the primary key value of an entity is definitely known at the moment of its creation, then the TCustomEntityContext.CreateAttachedEntity method can be used.

See Also

- CreateEntity
- CreateAttachedEntity
- Delete
- Save
- Cancel
- TEntity

Request Support DAC Forum Provide Feedback

6.17.1.1.2.2 Cancel Method

The method is designed to cancel changes made in an entity instance.

Class

TCustomEntityContext

Syntax

```
procedure Cancel(Entity: TMappedEntity; Cascade: boolean = False);
```

Parameters

Entity

The entity instance whose changes have to be cancelled.

Cascade

The parameter defines whether to perform cascade cancel of modidications of entity

references and linked collections when canceling the entity modifications. The default value is False.

Remarks

The method cancels modifications made in an entity instance. If an instance was deleted with TCustomEntityContext.Delete, the entity is restored from the object cache on the method execution, database access doesn't occur. The method cancels only those changes, that were not saved with the TCustomEntityContext.Save or TDataContext.SubmitChanges methods.

To cancel changes for all entities attached to the data context, the TDataContext.RejectChanges method is used.

See Also

- CreateEntity
- CreateAttachedEntity
- Delete
- Save
- Attach
- TEntity

6.17.1.1.2.3 CreateAttachedEntity Method

Class

`TCustomEntityContext`

Overload List

Name	Description
CreateAttachedEntity(MetaExpr: IMetaType)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(MetaExpr: IMetaType; **const** KeyValue: Variant)	The method is designed for creating a new entity instance and simultaneous attaching

	it to the data context
CreateAttachedEntity(MetaExpr: IMetaType; **const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(EntityClass: TMappedEntityClass)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(EntityClass: TMappedEntityClass; **const** KeyValue: Variant)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(EntityClass: TMappedEntityClass; **const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(MetaType: TMappedMetaType)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(MetaType: TMappedMetaType; **const** KeyValue: Variant)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(MetaType: TMappedMetaType; **const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity(Key: TPrimaryKey)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

`TCustomEntityContext`

Syntax

```
function CreateAttachedEntity(MetaExpr: IMetaType): TMappedEntity;
overload;
```

Parameters

MetaExpr

The meta-type of the entity to be created.

Remarks

The method creates a new entity instance of the specified meta-type. The entity attributes forming the primary key will be initialized by the default values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Note, that when using this method, the entity attributes forming the primary key are initialized by the default values, therefore it may cause the exception when attaching the entity to the data context due to unique key violation.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.CreateAttachedEntity(EmpType);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(MetaExpr: IMetaType; const
KeyValue: Variant): TMappedEntity; overload;
```

Parameters

MetaExpr

The meta-type of the entity to be created.

KeyValue

The initial value of the primary key meta attribute.

Remarks

The method creates a new entity instance of the specified meta-type. The entity primary key attribute will be initialized by the specified value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.CreateAttachedEntity(EmpType, 1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel

- TEntity

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(MetaExpr: IMetaType; const KeyValues: array of Variant): TMappedEntity; overload;
```

Parameters

MetaExpr

The meta-type of the entity to be created.

KeyValues

The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified meta-type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, M:Devart.EntityDAC.TEntityContext.Save(Devart.EntityDAC.TMappedEntity,System.Boolean), TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.CreateAttachedEntity(EmpType, [1, 1]);
  // ...
end;
```

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(EntityClass: TMappedEntityClass): TMappedEntity; overload;
```

Parameters

EntityClass

The class type of the entity to be created

Remarks

The method creates a new entity instance of the specified class type. The entity attributes forming the primary key will be initialized by the default values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Note, that when using this method, the entity attributes forming the primary key are initialized by the default values, therefore it may cause the exception when attaching the entity to the data context due to unique key violation.

Example

```
var
  EmpEntity: TEntity;
begin
  EmpEntity := Context.CreateAttachedEntity(TEmp);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(EntityClass: TMappedEntityClass;
const KeyValue: Variant): TMappedEntity; overload;
```

Parameters

EntityClass

The class type of the entity to be created

KeyValue

The initial value of the primary key meta attribute.

Remarks

The method creates a new entity instance of the specified meta-type. The entity primary key attribute will be initialized by the specified value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpEntity: TEntity;
begin
  EmpEntity := Context.CreateAttachedEntity(TEmp, 1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(EntityClass: TMappedEntityClass;
const KeyValues: array of Variant): TMappedEntity; overload;
```

Parameters

EntityClass

The class type of the entity to be created

KeyValues

The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified class type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by

this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, M:Devart.EntityDAC.TEntityContext.Save(Devart.EntityDAC.TMappedEntity,System.Boolean), TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpEntity: TEntity;
begin
  EmpEntity := Context.CreateAttachedEntity(TEmp, [1, 1]);
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(MetaType: TMappedMetaType):
TMappedEntity; overload;
```

Parameters

MetaType

The meta-type of the entity to be created.

Remarks

The method creates a new entity instance of the specified meta-type. The entity attributes forming the primary key will be initialized by the default values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling

TCustomEntityContext.Attach.

Note, that when using this method, the entity attributes forming the primary key are initialized by the default values, therefore it may cause the exception when attaching the entity to the data context due to unique key violation.

Example

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.CreateAttachedEntity(EmpType);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(MetaType: TMappedMetaType; const
KeyValue: Variant): TMappedEntity; overload;
```

Parameters

MetaType
The meta-type of the entity to be created.
KeyValue
The initial value of the primary key meta attribute.

Remarks

The method creates a new entity instance of the specified meta-type. The entity primary key attribute will be initialized by the specified value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.CreateAttachedEntity(EmpType, 1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(MetaType: TMappedMetaType; const KeyValues: array of Variant): TMappedEntity; overload;
```

Parameters

MetaType

The meta-type of the entity to be created.

KeyValues

The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified meta-type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, M:Devart.EntityDAC.TEntityContext.Save(Devart.EntityDAC.TMappedEntity,System.Boolean), TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.CreateAttachedEntity(EmpType, [1, 1]);
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity(Key: TPrimaryKey): TMappedEntity;
overload;
```

Parameters

Key

The primary key for the entity to be created.

Remarks

The method creates a new entity instance of the specified meta-type with the specified primary key. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  PK: TPrimaryKey;
  EmpEntity: TEntity;
begin
  PK := TPrimaryKey.Create(Context['Emp']);
  PK.Values[0].AsInteger := 1;
  EmpEntity := Context.CreateAttachedEntity(PK);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

6.17.1.1.2.4 CreateAttachedEntity<T> Method

Class

TCustomEntityContext

Overload List

Name	Description
CreateAttachedEntity<T>	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity<T>(Key: TPrimaryKey)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity<T>(**const** KeyValue: Variant)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context
CreateAttachedEntity<T>(**const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity<T: TMappedEntity>: T; overload;
```

Type parameters

T

The class type of the entity to be created

Remarks

The method creates a new entity instance of the specified meta-type. The attributes forming

the entity primary key will be initialized by the specified values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, M:Devart.EntityDAC.TEntityContext.Save(Devart.EntityDAC.TMappedEntity,System.Boolean), TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpEntity: TEntity;
begin
EmpEntity := Context.CreateAttachedEntity<TEmp>;
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity<T: TMappedEntity>(Key: TPrimaryKey):
T; overload;
```

Type parameters

T

The class type of the entity to be created

Parameters

Key

The primary key for the entity to be created.

Remarks

The method creates a new entity instance of the specified class type with the specified primary key. The entity instance created by this method is initially attached to the data context

and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  PK: TPrimaryKey;
  EmpEntity: TEntity;
begin
  PK := TPrimaryKey.Create(Context['Emp']);
  PK.Values[0].AsInteger := 1;
EmpEntity := Context.CreateAttachedEntity<TEmp>(PK);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity<T: TMappedEntity>(const KeyValue:
Variant): T; overload;
```

Type parameters

T

The class type of the entity to be created

Parameters

KeyValue

Remarks

The method creates a new entity instance of the specified class type. The entity attributes forming the primary key will be initialized by the default values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Note, that when using this method, the entity attributes forming the primary key are initialized by the default values, therefore it may cause the exception when attaching the entity to the data context due to unique key violation.

Example

```
var
  EmpEntity: TEntity;
begin
EmpEntity := Context.CreateAttachedEntity<TEmp>(1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance and simultaneous attaching it to the data context

Class

TCustomEntityContext

Syntax

```
function CreateAttachedEntity<T: TMappedEntity>(const KeyValues:
array of Variant): T; overload;
```

Type parameters

T

The class type of the entity to be created

Parameters

KeyValues

The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified class type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, M:Devart.EntityDAC.TEntityContext.Save(Devart.EntityDAC.TMappedEntity,System.Boolean), TCustomEntityContext.Cancel, without pre-calling TCustomEntityContext.Attach.

Example

```
var
  EmpEntity: TEntity;
begin
EmpEntity := Context.CreateAttachedEntity<TEmp>([1, 1]);
  // ...
end;
```

6.17.1.1.2.5 CreateEntity Method

Class

TCustomEntityContext

Overload List

Name	Description
CreateEntity(MetaExpr: IMetaType)	The method is designed for creating a new entity instance.
CreateEntity(MetaExpr: IMetaType; **const** KeyValue: Variant)	The method is designed for creating a new entity instance.
CreateEntity(MetaExpr: IMetaType; **const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance.
CreateEntity(EntityClass: TMappedEntityClass)	The method is designed for creating a new entity instance.
CreateEntity(EntityClass: TMappedEntityClass; **const** KeyValue: Variant)	The method is designed for creating a new entity instance.
CreateEntity(EntityClass: TMappedEntityClass; **const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance.
CreateEntity(MetaType: TMappedMetaType)	The method is designed for creating a new entity instance.
CreateEntity(MetaType: TMappedMetaType; **const** KeyValue: Variant)	The method is designed for creating a new entity instance.
CreateEntity(MetaType: TMappedMetaType; **const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance.
CreateEntity(Key: TPrimaryKey)	The method is designed for creating a new entity instance.

Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(MetaExpr: IMetaType): TMappedEntity;
overload;
```

Parameters

MetaExpr

The meta-type of the entity to be created.

Remarks

The method creates a new entity instance of the specified meta-type. The entity attributes forming the primary key will be initialized by the default values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.CreateEntity(EmpType);
  EmpEntity.Attributes['EmpNo'].AsInteger := 1;
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach

- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(MetaExpr: IMetaType; const KeyValue:
Variant): TMappedEntity; overload;
```

Parameters

MetaExpr
The meta-type of the entity to be created.

KeyValue
The initial value of the primary key meta attribute.

Remarks

The method creates a new entity instance of the specified meta-type. The entity primary key attribute will be initialized by the specified value. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with

simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.CreateEntity(EmpType, 1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(MetaExpr: IMetaType; const KeyValues: array
 of Variant): TMappedEntity; overload;
```

Parameters

MetaExpr

The meta-type of the entity to be created.

KeyValues

The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified meta-type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.CreateEntity(EmpType, [1, 1]);
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(EntityClass: TMappedEntityClass):
TMappedEntity; overload;
```

Parameters

EntityClass
The class type of the entity to be created

Remarks

The method creates a new entity instance of the specified class type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpEntity: TEntity;
begin
  EmpEntity := Context.CreateEntity(TEmp);
  EmpEntity.Attributes['EmpNo'].AsInteger := 1;
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(EntityClass: TMappedEntityClass; const
KeyValue: Variant): TMappedEntity; overload;
```

Parameters

EntityClass

The class type of the entity to be created

KeyValue

The initial value of the primary key meta attribute.

Remarks

The method creates a new entity instance of the specified class type. The entity primary key attribute will be initialized by the specified value. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpEntity: TEntity;
begin
  EmpEntity := Context.CreateEntity(TEmp, 1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel

- TEntity

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(EntityClass: TMappedEntityClass; const KeyValues: array of Variant): TMappedEntity; overload;
```

Parameters

EntityClass
The class type of the entity to be created

KeyValues
The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified class type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpEntity: TEntity;
begin
  EmpEntity := Context.CreateEntity(TEmp, [1, 1]);
  // ...
end;
```


The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(MetaType: TMappedMetaType): TMappedEntity;
overload;
```

Parameters

MetaType

Remarks

The method creates a new entity instance of the specified meta-type. The entity attributes forming the primary key will be initialized by the default values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.CreateEntity(EmpType);
  EmpEntity.Attributes['EmpNo'].AsInteger := 1;
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(MetaType: TMappedMetaType; const KeyValue:
Variant): TMappedEntity; overload;
```

Parameters

MetaType

KeyValue

The initial value of the primary key meta attribute.

Remarks

The method creates a new entity instance of the specified meta-type. The entity primary key attribute will be initialized by the specified value. The entity instance created by this method is

not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.CreateEntity(EmpType, 1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(MetaType: TMappedMetaType; const KeyValues:
array of Variant): TMappedEntity; overload;
```

Parameters

MetaType

KeyValues

The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified meta-type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEntity;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.CreateEntity(EmpType, [1, 1]);
  // ...
end;
```

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity(Key: TPrimaryKey): TMappedEntity; overload;
```

Parameters

Key

The primary key for the entity to be created.

Remarks

The method creates a new entity instance of the specified meta-type with the specified primary key. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  PK: TPrimaryKey;
  EmpEntity: TEntity;
begin
  PK := TPrimaryKey.Create(Context['Emp']);
  PK.Values[0].AsInteger := 1;
  EmpEntity := Context.CreateEntity(PK);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

6.17.1.1.2.6 CreateEntity<T> Method

Class

TCustomEntityContext

Overload List

Name	Description
CreateEntity<T>	The method is designed for creating a new entity instance.
CreateEntity<T>(Key: TPrimaryKey)	The method is designed for creating a new entity instance.
CreateEntity<T>(**const** KeyValue: Variant)	The method is designed for creating a new entity instance.
CreateEntity<T>(**const** KeyValues: **array of** Variant)	The method is designed for creating a new entity instance.

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

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity<T: TMappedEntity>: T; overload;
```

Type parameters

T

The class type of the entity to be created

Remarks

The method creates a new entity instance of the specified class type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpEntity: TEntity;
begin
EmpEntity := Context.CreateEntity<TEmp>;
  EmpEntity.Attributes['EmpNo'].AsInteger := 1;
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity<T: TMappedEntity>(Key: TPrimaryKey): T;
```

```
overload;
```

Type parameters

T

The class type of the entity to be created

Parameters

Key

The primary key for the entity to be created.

Remarks

The method creates a new entity instance of the specified class type with the specified primary key. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  PK: TPrimaryKey;
  EmpEntity: TEntity;
begin
  PK := TPrimaryKey.Create(Context['Emp']);
  PK.Values[0].AsInteger := 1;
EmpEntity := Context.CreateEntity<TEmp>(PK);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach

- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity<T: TMappedEntity>(const KeyValue: Variant): T; overload;
```

Type parameters

T

The class type of the entity to be created

Parameters

KeyValue

The initial value of the primary key meta attribute.

Remarks

The method creates a new entity instance of the specified class type. The entity primary key attribute will be initialized by the specified value. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the

TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpEntity: TEntity;
begin
EmpEntity := Context.CreateAttachedEntity<TEmp>(1);
  // ...
end;
```

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for creating a new entity instance.

Class

TCustomEntityContext

Syntax

```
function CreateEntity<T: TMappedEntity>(const KeyValues: array of
Variant): T; overload;
```

Type parameters

T

The class type of the entity to be created

Parameters

KeyValues

The array of initial values of meta attributes forming the entity primary key.

Remarks

The method creates a new entity instance of the specified class type. The attributes forming the entity primary key will be initialized by the specified values. The entity instance created by this method is not attached to the data context (is not placed to the object cache), therefore it won't be automatically destroyed on application shutdown. It should be destroyed manually. It is expedient to use the method in case when the primary key value of an entity is unknown at the moment of its creation (or there is no confidence in its uniqueness), in order to avoid an exception on an attempt to place an object to the cache.

To attach an entity instance to the data context and place it to the object cache (in order to perform further modification operations: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel), the TCustomEntityContext.Attach method should be used. To create an entity instance with simultaneous attaching it to the data context, the TCustomEntityContext.CreateAttachedEntity method should be used.

Example

```
var
  EmpEntity: TEntity;
begin
EmpEntity := Context.CreateEntity<TEmp>([1, 1]);
  // ...
end;
```


6.17.1.1.2.7 Delete Method

The method is designed for deleting an entity.

Class

TCustomEntityContext

Syntax

```
procedure Delete(Entity: TMappedEntity; Cascade: boolean = False);
```

Parameters

Entity
The entity to be deleted.

Cascade
The parameter defines whether to perform cascade deletion of entity references and linked collections when deleting the entity. The default value is False.

Remarks

The method deletes the specified entity. The performed deletion is recoverable. When the method is performed, references to the entity are not deleted from linked entities, the entity is not deleted from collections of linked objects. Physical deletion of data from corresponding database structures doesn't occur as well.

To cancel entity deletion, the TCustomEntityContext.Cancel method is used.

To submit entity deletion, the TCustomEntityContext.Save method is used.

To permanently delete an entity, the TCustomEntityContext.DeleteAndSave method is used.

See Also

- Save
- Cancel
- DeleteAndSave
- TEntity

6.17.1.1.2.8 DeleteAndSave Method

The method is designed for permanent deleting an entity.

Class

TCustomEntityContext

Syntax

```
procedure DeleteAndSave(Entity: TMappedEntity; Cascade: boolean = False);
```

Parameters

Entity

The entity to be deleted.

Cascade

The parameter defines whether to perform cascade premanent deletion of entity references and linked collections when deleting the entity. The default value is False.

Remarks

The method deletes the specified entity. The performed deletion is irreversible. When the method is performed, references to the entity are deleted from linked entities, the entity is deleted from collections of linked entities, physical deletion of data from corresponding database structures occurs as well. The method execution is equivalent to consequent execution of the methods: TCustomEntityContext.Delete, TCustomEntityContext.Save

See Also

- Save
- Cancel
- Delete
- TEntity

6.17.1.1.2.9 GetEntities Method

Class

`TCustomEntityContext`

Overload List

Name	Description
GetEntities	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(MetaExpr: IMetaType)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(MetaExpr: IMetaType; **const** Condition: TExpression)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(MetaExpr: IMetaType; **const** Condition: **string**)	The method is designed for retrieving an entity collection by specified criteria.

GetEntities(Key: TCustomKey)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(EntityClass: TMappedEntityClass)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(EntityClass: TMappedEntityClass; **const** Condition: TExpression)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(EntityClass: TMappedEntityClass; **const** Condition: **string**)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(MetaType: TMappedMetaType)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(MetaType: TMappedMetaType; **const** Condition: TExpression)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities(MetaType: TMappedMetaType; **const** Condition: **string**)	The method is designed for retrieving an entity collection by specified criteria.

The method is designed for retrieving an entity collection by specified criteria.

Unit

Syntax

Remarks

The method returns an entity collection by the specified LINQ query. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
```

```
  EmpType: IMetaType;
  Expression: ILinqQueryable;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  Expression := Context.From(EmpType).Where(EmpType['Deptno'] = 20).Select;
  EmpEntities := Context.GetEntities(Expression);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(MetaExpr: IMetaType): IEntityEnumerable;
overload;
```

Parameters

MetaExpr

The meta-type of entities to be retrieved.

Remarks

The method returns a collection that contains all entities of the specified meta-type. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be

automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  EmpEntities := Context.GetEntities(EmpType);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(MetaExpr: IMetaType; const Condition:
TExpression): IEntityEnumerable; overload;
```

Parameters

MetaExpr

The meta-type of entities to be retrieved.

Condition

The logical expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified meta-type selected by the specified conditional expression. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  EmpEntities := Context.GetEntities(EmpType, EmpType['Sal'] > 1000);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(MetaExpr: IMetaType; const Condition:
string): IEntityEnumerable; overload;
```

Parameters

MetaExpr

The meta-type of entities to be retrieved.

Condition

The string expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified meta-type selected by the specified string condition. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  EmpEntities := Context.GetEntities(EmpType, 'Sal > 1000');
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

 Request Support DAC Forum Provide Feedback

Reserved.

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(Key: TCustomKey): IEntityEnumerable;
overload;
```

Parameters

Key

The custom entity key that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified meta-type selected by the specified key value. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  Key: TEntityKey;
  EmpEntities: IEntityEnumerable;
begin
  Key := TEntityKey.Create(EntityContext['Emp']['Deptno']);
  Key.Values[0].AsInteger := 10;
  EmpEntities := Context.GetEntities(Key);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach

- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(EntityClass: TMappedEntityClass):
IEntityEnumerable; overload;
```

Parameters

EntityClass
The class type of entities to be retrieved.

Remarks

The method returns a collection that contains all entities of the specified meta-type. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  EmpEntities := Context.GetEntities(EmpType);
```

```
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(EntityClass: TMappedEntityClass; const
Condition: TExpression): IEntityEnumerable; overload;
```

Parameters

EntityClass

The class type of entities to be retrieved.

Condition

The logical expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified meta-type selected by the specified conditional expression. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  EmpEntities := Context.GetEntities(EmpType, EmpType['Sal'] > 1000);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(EntityClass: TMappedEntityClass; const
Condition: string): IEntityEnumerable; overload;
```

Parameters

EntityClass

The class type of entities to be retrieved.

Condition

The string expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified meta-type selected by the specified string condition. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  EmpEntities := Context.GetEntities(EmpType, 'Sal > 1000');
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(MetaType: TMappedMetaType):
IEntityEnumerable; overload;
```

Parameters

MetaType

Remarks

The method returns a collection that contains all entities of the specified meta-type. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: TMappedMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntities := Context.GetEntities(EmpType);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(MetaType: TMappedMetaType; const Condition:
TExpression): IEntityEnumerable; overload;
```

Parameters

MetaType

Condition

The logical expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified meta-type selected by the specified conditional expression. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpMetaType: TMappedMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context['Emp'];
  EmpMetaType := Context.Model.MetaTypes.Get('Emp');
  EmpEntities := Context.GetEntities(EmpMetaType, EmpType['Sal'] > 1000);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel

- TEntity

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities(MetaType: TMappedMetaType; const Condition: string): IEntityEnumerable; overload;
```

Parameters

MetaType

Condition

The string expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified meta-type selected by the specified string condition. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: TMappedMetaType;
  EmpEntities: IEntityEnumerable;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntities := Context.GetEntities(EmpType, 'Sal > 1000');
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

 Request Support DAC Forum Provide Feedback

6.17.1.1.2.10 GetEntities<T> Method

Class

TCustomEntityContext

Overload List

Name	Description
GetEntities<T>	The method is designed for retrieving an entity collection by specified criteria.
GetEntities<T>(LinqExpression: ILinqBase)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities<T>(Key: TCustomKey)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities<T>(**const** Condition: TExpression)	The method is designed for retrieving an entity collection by specified criteria.
GetEntities<T>(**const** Condition: **string**)	The method is designed for retrieving an entity collection by specified criteria.

 Request Support DAC Forum Provide Feedback

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities<T: TMappedEntity>: IEntityEnumerable<T>;
overload;
```

Type parameters

T

The class type of entities to be retrieved

Remarks

The method returns a collection that contains all entities of the specified class type. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
EmpEntities: IEntityEnumerable<TEmp>;
begin
EmpEntities := Context.GetEntities<TEmp>;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities<T: TEntity>(LinqExpression: ILinqBase):
IEntityEnumerable<T>; overload;
```

Type parameters

T

The class type of entities to be retrieved

Parameters

LinqExpression

A complete LINQ query for selecting entities.

Remarks

The method returns a collection of entities of the specified class type by the specified LINQ query.. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  Expression: ILinqQueryable;
EmpEntities: IEntityEnumerable<TEmp>;
begin
  EmpType := Context['Emp'];
  Expression := Context.From(EmpType).Where(EmpType['Deptno'] = 20).Select;
  EmpEntities := Context.GetEntities<TEmp>(Expression);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities<T: TMappedEntity>(Key: TCustomKey):
IEntityEnumerable<T>; overload;
```

Type parameters

T

The class type of entities to be retrieved

Parameters

Key

The custom entity key that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified class type selected by the specified key value. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  Key: TEntityKey;
EmpEntities: IEntityEnumerable<TEmp>;
begin
  Key := TEntityKey.Create(EntityContext['Emp']['EmpNo']);
  Key.Values[0].AsInteger := 1;
EmpEntities := Context.GetEntities<TEmp%gt;(Key);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities<T: TMappedEntity>(const Condition:
TExpression): IEntityEnumerable<T>; overload;
```

Type parameters

T

The class type of entities to be retrieved

Parameters

Condition

The logical expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified class type selected by the specified conditional expression. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
EmpEntities: IEntityEnumerable<TEmp>;
begin
  EmpType := Context['Emp'];
EmpEntity := Context.GetEntities<TEmp>(EmpType['Sal'] > 1000);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an entity collection by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntities<T: TMappedEntity>(const Condition: string):
IEntityEnumerable<T>; overload;
```

Type parameters

T

The class type of entities to be retrieved

Parameters

Condition

The string expression that defines the condition which each of selected entities must conform.

Remarks

The method returns a collection of entities of the specified class type selected by the specified string condition. Collection members created by this method are initially attached to the data context and placed to the object cache, therefore you can already perform modification operations for them: TCustomEntityContext.Delete, TCustomEntityContext.Save, TCustomEntityContext.Cancel without pre-calling TCustomEntityContext.Attach. In addition, these entity instances will be automatically destroyed, and there will be no need to provide for their manual destruction.

Example

Examples of calling the method:

```
var
EmpEntities: IEntityEnumerable<TEmp>;
begin
EmpEntity := Context.GetEntities<TEmp>('Sal > 1000');
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

6.17.1.1.2.11 GetEntity Method

Class

TCustomEntityContext

Overload List

Name	Description
GetEntity	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaExpr: IMetaType; **const** Condition: TExpression)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaExpr: IMetaType; **const** KeyValue: Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaExpr: IMetaType; **const** KeyValues: **array of** Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaExpr: IMetaType; **const** Condition: **string**)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(Key: TCustomKey)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(EntityClass: TMappedEntityClass; **const** Condition: TExpression)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(EntityClass: TMappedEntityClass; **const** KeyValue: Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(EntityClass: TMappedEntityClass; **const** KeyValues: **array of** Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(EntityClass: TMappedEntityClass; **const** Condition: **string**)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaType: TMappedMetaType; **const** Condition: TExpression)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaType: TMappedMetaType; **const** KeyValue: Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaType: TMappedMetaType; **const** KeyValues: **array of** Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity(MetaType: TMappedMetaType; **const** Condition: **string**)	The method is designed for retrieving an existing entity instance by specified criteria.

The method is designed for retrieving an existing entity instance by specified criteria.

Unit

Syntax

Remarks

The method returns an entity instance by the specified LINQ query. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  Expression: ILinqQueryable;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
Expression := Context.From(EmpType).Where(EmpType['EmpNo'] = 1).Select;
 EmpEntity := Context.GetEntity(Expression) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaExpr: IMetaType; const Condition:
TExpression): TMappedEntity; overload;
```

Parameters

MetaExpr

The meta-type of the entity to be retrieved.

Condition

The logical expression that defines the condition which the selected entitiy must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified conditional expression. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, EmpType['EmpNo'] = 1) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaExpr: IMetaType; const KeyValue:
Variant): TMappedEntity; overload;
```

Parameters

MetaExpr

The meta-type of the entity to be retrieved.

KeyValue

The value of the entity primary key.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified primary key value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, 1) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaExpr: IMetaType; const KeyValues: array of
Variant): TMappedEntity; overload;
```

Parameters

MetaExpr
The meta-type of the entity to be retrieved.

KeyValues
The value of the entity primary key.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified complex primary key values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the

TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, [1, 1]) as TEmp;
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaExpr: IMetaType; const Condition: string):
TMappedEntity; overload;
```

Parameters

MetaExpr

The meta-type of the entity to be retrieved.

Condition

The string expression that defines the condition which the selected entity must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified string condition. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling

TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, 'EmpNo = 1') as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(Key: TCustomKey): TMappedEntity; overload;
```

Parameters

Key

The custom entity key that defines the condition which the selected entitiy must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified key value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  Key: TEntityKey;
  EmpEntity: TEmp;
begin
  Key := TEntityKey.Create(EntityContext['Emp']['EmpNo']);
  Key.Values[0].AsInteger := 1;
  EmpEntity := Context.GetEntity(Key) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(EntityClass: TMappedEntityClass; const
```

```
Condition: TExpression): TMappedEntity; overload;
```

Parameters

EntityClass

The class type of the entity to be retrieved.

Condition

The logical expression that defines the condition which the selected entitiy must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified conditional expression. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, EmpType['EmpNo'] = 1) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(EntityClass: TMappedEntityClass; const
KeyValue: Variant): TMappedEntity; overload;
```

Parameters

EntityClass

The class type of the entity to be retrieved.

KeyValue

The value of the entity primary key.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified primary key value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, 1) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete

- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(EntityClass: TMappedEntityClass; const KeyValues: array of Variant): TMappedEntity; overload;
```

Parameters

EntityClass
The class type of the entity to be retrieved.

KeyValues
The value of the entity primary key.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified complex primary key values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, [1, 1]) as TEmp;
```

```
  // ...
end;
```


The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(EntityClass: TMappedEntityClass; const
Condition: string): TMappedEntity; overload;
```

Parameters

EntityClass

The class type of the entity to be retrieved.

Condition

The string expression that defines the condition which the selected entity must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified string condition. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  EmpEntity := Context.GetEntity(EmpType, 'EmpNo = 1') as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaType: TMappedMetaType; const Condition:
TExpression): TMappedEntity; overload;
```

Parameters

MetaType

Condition

The logical expression that defines the condition which the selected entitiy must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified conditional expression. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.GetEntity(EmpType, EmpType['EmpNo'] = 1) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaType: TMappedMetaType; const KeyValue:
Variant): TMappedEntity; overload;
```

Parameters

MetaType

KeyValue

The value of the entity primary key.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified primary key value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity

method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEmp;
begin
EmpType := Context.Model.MetaTypes.Get('Emp');
 EmpEntity := Context.GetEntity(EmpType, 1) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaType: TMappedMetaType; const KeyValues:
array of Variant): TMappedEntity; overload;
```

Parameters

MetaType

KeyValues
The value of the entity primary key.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified complex primary key values. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEmp;
begin
EmpType := Context.Model.MetaTypes.Get('Emp');
 EmpEntity := Context.GetEntity(EmpType, [1, 1]) as TEmp;
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity(MetaType: TMappedMetaType; const Condition:
string): TMappedEntity; overload;
```

Parameters

MetaType

Condition
The string expression that defines the condition which the selected entity must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified string condition. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: TMappedMetaType;
  EmpEntity: TEmp;
begin
  EmpType := Context.Model.MetaTypes.Get('Emp');
  EmpEntity := Context.GetEntity(EmpType, 'EmpNo = 1') as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

6.17.1.1.2.12 GetEntity<T> Method

Class

TCustomEntityContext

Overload List

Name	Description

GetEntity<T>	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity<T>(Key: TCustomKey)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity<T>(**const** Condition: TExpression)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity<T>(**const** KeyValue: Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity<T>(**const** KeyValues: **array of** Variant)	The method is designed for retrieving an existing entity instance by specified criteria.
GetEntity<T>(**const** Condition: **string**)	The method is designed for retrieving an existing entity instance by specified criteria.

The method is designed for retrieving an existing entity instance by specified criteria.

Unit

Syntax

Remarks

The method returns an entity instance of the specified class type selected by the specified LINQ query. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpType: IMetaType;
  Expression: ILinqQueryable;
  EmpEntity: TEmp;
begin
  EmpType := Context['Emp'];
  Expression := Context.From(EmpType).Where(EmpType['EmpNo'] = 1).Select;
```

```
  EmpEntity := Context.GetEntity<TEmp>(Expression);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity<T: TMappedEntity>(Key: TCustomKey): T;
overload;
```

Type parameters

T

Parameters

Key

The custom entity key that defines the condition which the selected entitiy must conform.

Remarks

The method returns an entity instance of the specified meta-type selected by the specified key value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  Key: TEntityKey;
  EmpEntity: TEmp;
begin
  Key := TEntityKey.Create(EntityContext['Emp']['EmpNo']);
  Key.Values[0].AsInteger := 1;
  EmpEntity := Context.GetEntity(Key) as TEmp;
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity<T: TMappedEntity>(const Condition:
TExpression): T; overload;
```

Type parameters

T

The class type of the entity to be retrieved

Parameters

Condition

The logical expression that defines the condition which the selected entitiy must conform.

Remarks

The method returns an entity instance of the specified class type selected by the specified conditional expression. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpEntity: TEmp;
begin
EmpEntity := Context.GetEntity<TEmp>(EmpType, EmpType['EmpNo'] = 1);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity<T: TMappedEntity>(const KeyValue: Variant):
```

```
T; overload;
```

Type parameters

T

The class type of the entity to be retrieved

Parameters

KeyValue

The value of the entity primary key.

Remarks

The method returns an entity instance of the specified class type selected by the specified primary key value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpEntity: TEmp;
begin
EmpEntity := Context.GetEntity<TEmp>(EmpType, 1);
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel
- TEntity

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity<T: TMappedEntity>(const KeyValues: array of Variant): T; overload;
```

Type parameters

T

The class type of the entity to be retrieved

Parameters

KeyValues

The value of the entity primary key.

Remarks

The method returns an entity instance of the specified class type selected by the specified complex primary key value. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

```
var
  EmpEntity: TEmp;
begin
EmpEntity := Context.GetEntity<TEmp>(EmpType, [1, 1]);
  // ...
end;
```


Request Support DAC Forum Provide Feedback

The method is designed for retrieving an existing entity instance by specified criteria.

Class

TCustomEntityContext

Syntax

```
function GetEntity<T: TMappedEntity>(const Condition: string): T; overload;
```

Type parameters

T

The class type of the entity to be retrieved

Parameters

Condition

The string expression that defines the condition which the selected entity must conform.

Remarks

The method returns an entity instance of the specified class type selected by the specified string condition. The entity instance created by this method is initially attached to the data context and placed to the object cache (in contrast to the TCustomEntityContext.CreateEntity method), therefore you can already perform modification operations for the instance: TCustomEntityContext.Delete, TCustomEntityContext.Cancel) without pre-calling TCustomEntityContext.Attach. In addition, this entity instance will be automatically destroyed, and there will be no need to provide for its manual destruction.

Example

Examples of calling the method:

```
var
  EmpEntity: TEmp;
begin
EmpEntity := Context.GetEntity<TEmp>(EmpType, 'EmpNo = 1');
  // ...
end;
```

See Also

- TCustomEntityContext.Attach
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Cancel

- TEntity

6.17.1.1.2.13 IsAttached Method

The method is designed to check whether the specified entity is attached to the data context.

Class

TCustomEntityContext

Syntax

```
function IsAttached(Entity: TMappedEntity): boolean;
```

Parameters

Entity
The entity which has to be checked

Remarks

Use the method to determine whether the particular entity is attached to the data context. The method returns True if the entity is attached, False otherwise. "Attached" means that the entity is either created with the TCustomEntityContext.CreateAttachedEntity method, or created with the TCustomEntityContext.CreateEntity method and then attached using the Attach method.

See Also

- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.CreateEntity
- Attach

6.17.1.1.2.14 Save Method

The method is designed for saving changes made in an entity instance.

Class

TCustomEntityContext

Syntax

```
procedure Save(Entity: TMappedEntity; Cascade: boolean = False);
```

Parameters

Entity

The entity instance to be saved.

Cascade

The parameter defines whether to perform cascade saving of changes of entity references and linked collections when saving entity changes. The default value is False.

Remarks

The method performs permanent saving of modifications made in an entity instance. If the entity was deleted with TCustomEntityContext.Delete, then when executing this method, there occurs deletion of references to this entity in linked entities, deletion of the entity from linked entities collections, as well as deletion of data from corresponding database structures.

For irreversible saving of changes of all entities attached to the data context, the TDataContext.SubmitChanges method is used.

See Also

- Delete
- TDataContext.SubmitChanges
- TEntity

6.17.1.2 TEntityCollectionUpdater Class

The base class for representing a list of the entity collections.

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

Unit

EntityDAC.EntityContext

Syntax

```
TEntityCollectionUpdater = class(TEntityLinkUpdater,
ICollectionUpdater, INotifiableCollection);
```

Remarks

TEntityCollectionUpdater contains a list of TEntityCollection and provides methods for iterating and accessing the list elements. TEntityCollectionUpdater is the base class and should not be used directly. For operating entity collections in the code TMappedCollections class is used.

Inheritance Hierarchy

TEntityLinkUpdater

TEntityCollectionUpdater

See Also

- TEntityCollection
- TMappedCollections

6.17.1.2.1 Members

TEntityCollectionUpdater class overview.

Properties

Name	Description
Count	Indicates elements count in the list.
Items	Returns the collection by its index.

6.17.1.2.2 Properties

Properties of the **TEntityCollectionUpdater** class.

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

Public

Name	Description
Count	Indicates elements count in the list.
Items	Returns the collection by its index.

See Also

- TEntityCollectionUpdater Class
- TEntityCollectionUpdater Class Members

Request Support DAC Forum Provide Feedback

6.17.1.2.2.1 Count Property

Indicates elements count in the list.

Class

TEntityCollectionUpdater

Syntax

```
property Count: Integer;
```

Remarks

The property indicates the number of TEntityCollection elements contained in the list.

See Also

- TEntityCollection

Request Support DAC Forum Provide Feedback

6.17.1.2.2.2 Items Property(Indexer)

Returns the collection by its index.

Class

TEntityCollectionUpdater

Syntax

```
property Items[Index: integer]: TObject;
```

Parameters

Index
The index of the collection.

Remarks

The function returns the TEntityCollection element by its specified index.

See Also

- TEntityCollection

Request Support DAC Forum Provide Feedback

6.17.1.3 TEntityContext Class

The class that provides the data context functionality.

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

Unit

EntityDAC.EntityContext

Syntax

```
TEntityContext = class(TCustomEntityContext);
```

Remarks

TEntityContext class is derived from TCustomEntityContext and provides functionality for managing an entity life cycle in the application. It provides methods for creating and initializing new entity instances, retrieving and storing entities from/to the database, storing used entities in the cache for future use, destroying of unused entities.

Inheritance Hierarchy

TCustomContext
  TDataContext
    TCustomEntityContext
      TEntityContext

See Also

- TCustomEntityContext

6.17.1.3.1 Members

TEntityContext class overview.

Properties

Name	Description
Connection	Identifies the connection component with which the data context is associated.
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
ModelName	Identifies the meta model with which the data context is associated.
Options	The class allows setting up the behavior of the TEntityContext class.

Types (inherited from **TDataContext**)	The property is designed to determine a meta-type by a meta-type name.

Methods

Name	Description
Attach (inherited from **TCustomEntityContext**)	The method is designed for attaching an entity instance to the data context and storing it in the object cache
Cancel (inherited from **TCustomEntityContext**)	The method is designed to cancel changes made in an entity instance.
Create (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
CreateAttachedEntity (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.
CreateAttachedEntity<T> (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.
CreateEntity (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.
CreateEntity<T> (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.
Delete (inherited from **TCustomEntityContext**)	The method is designed for deleting an entity.
DeleteAndSave (inherited from **TCustomEntityContext**)	The method is designed for permanent deleting an entity.
ExecuteQuery<T> (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
ExecuteSQL (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
GetEntities (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.
GetEntities<T> (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.
GetEntity (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.
GetEntity<T> (inherited from **TCustomEntityContext**)	Overloaded.Description is not available at the moment.

IsAttached (inherited from **TCustomEntityContext**)	The method is designed to check whether the specified entity is attached to the data context.
RejectChanges (inherited from **TDataContext**)	The method is designed to cancel changes in all attached entities
Save (inherited from **TCustomEntityContext**)	The method is designed for saving changes made in an entity instance.
SubmitChanges (inherited from **TDataContext**)	The method is designed for saving changes in all attached entities.

Events

Name	Description
OnGetGeneratorValue (inherited from **TCustomContext**)	Occurs when an entity attribute value generator of type "custom" needs to generate its value.

Request Support DAC Forum Provide Feedback

6.17.1.3.2 Properties

Properties of the **TEntityContext** class.

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

Public

Name	Description
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
Types (inherited from **TDataContext**)	The property is designed to determine a meta-type by a meta-type name.

Published

Name	Description
Connection	Identifies the connection component with which the data context is associated.
ModelName	Identifies the meta model with which the data context is associated.
Options	The class allows setting up the behavior of the TEntityContext class.

See Also

- TEntityContext Class
- TEntityContext Class Members

Request Support DAC Forum Provide Feedback

6.17.1.3.2.1 Connection Property

Identifies the connection component with which the data context is associated.

Class

TEntityContext

Syntax

```
property Connection: TEntityConnection;
```

Remarks

Set the property to associate the data context with the TEntityConnection component. Use the property to access properties, events and methods of the connection associated with the data context.

See Also

- TEntityConnection

6.17.1.3.2.2 ModelName Property

Identifies the meta model with which the data context is associated.

Class

TEntityContext

Syntax

```
property ModelName: string;
```

Remarks

Set the property to associate the data context with the meta model that contains meta-data of all entity types which the data context has to operate.

When the property is not set then the data context uses the default meta model that specified in the TEntityConnection.DefaultModelName property.

See Also

- TEntityConnection
- TEntityConnection.DefaultModelName

6.17.1.3.2.3 Options Property

The class allows setting up the behavior of the TEntityContext class.

Class

TEntityContext

Syntax

```
property Options: TContextOptions;
```

See Also

- TContextOptions

 Request Support DAC Forum Provide Feedback

6.17.1.4 TMappedCollections Class

Represents a list of the entity collections.

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

Unit

EntityDAC.EntityContext

Syntax

```
TMappedCollections = class(TEntityEnumerables);
```

Remarks

TMappedCollections contains a list of TMappedCollection and provides methods for iterating and accessing the list elements.

Inheritance Hierarchy

TEntityEnumerables

TMappedCollections

See Also

- TMappedCollection

 Request Support DAC Forum Provide Feedback

6.17.1.4.1 Members

TMappedCollections class overview.

 Request Support DAC Forum Provide Feedback

6.17.1.5 TMappedEntity Class

The class that represents a mapped entity instance and provides methods for managing it.

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

Unit

EntityDAC.EntityContext

Syntax

```
TMappedEntity = class(TEntity);
```

Remarks

TMappedEntity intended to hold the instance of an entity that is mapped to the particular database table. TMappedEntity is updatable, it can be saved to the database or deleted. Those entities that are the result of a query execution and can not be mapped to the particular table, are represented with the TUnmappedEntity class instances.

Inheritance Hierarchy

TEntity

TMappedEntity

See Also

- TEntity
- TUnmappedEntity

6.17.1.5.1 Members

TMappedEntity class overview.

Properties

Name	Description
Attributes (inherited from **TEntity**)	The property represents the entity attributes collection.

Collections	The property represents a list of the entity collections.
EntityState (inherited from **TEntity**)	The property indicates the entity state.
MetaType	The property is designed for indicating the entity meta-type.
References	The property represents a list of the entity references.
UpdateState (inherited from **TEntity**)	The property indicates the entity update state.

Methods

Name	Description
AttributeByName (inherited from **TEntity**)	Returns an entity attribute by its name.
Cancel	Overloaded. The method is designed to cancel changes made in an entity instance.
Compare (inherited from **TEntity**)	The method is designed for comparing the entity key with the specified key.
Create (inherited from **TEntity**)	Overloaded. The constructor is designed for creating a new entity instance.
Delete	Overloaded. The method is designed for deleting an entity.
DeleteAndSave	Overloaded. The method is designed for permanently deleting an entity.
FromKey (inherited from **TEntity**)	The method is designed for setting the entity key value.
IsAttached (inherited from **TEntity**)	The method is designed to determine whether the entity is attached.
Save	Overloaded. The method is designed for saving changes made in an entity instance.
ToKey (inherited from **TEntity**)	The method is designed for filling the specified key with the entity key value.

6.17.1.5.2 Properties

Properties of the **TMappedEntity** class.

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

Public

Name	Description
Attributes (inherited from **TEntity**)	The property represents the entity attributes collection.
Collections	The property represents a list of the entity collections.
EntityState (inherited from **TEntity**)	The property indicates the entity state.
MetaType	The property is designed for indicating the entity meta-type.
References	The property represents a list of the entity references.
UpdateState (inherited from **TEntity**)	The property indicates the entity update state.

See Also

- TMappedEntity Class
- TMappedEntity Class Members

6.17.1.5.2.1 Collections Property

The property represents a list of the entity collections.

Class

`TMappedEntity`

Syntax

```
property Collections: TMappedCollections;
```

Remarks

The TMappedCollections class represents a list of the entity collections and provides methods for iterating and accessing the list elements.

See Also

- TMappedCollections

Request Support DAC Forum Provide Feedback

6.17.1.5.2.2 MetaType Property

The property is designed for indicating the entity meta-type.

Class

TMappedEntity

Syntax

```
property MetaType: TMappedMetaType;
```

Remarks

The property indicates the entity meta-type. The property is read-only.

See Also

- TMappedMetaType

Request Support DAC Forum Provide Feedback

6.17.1.5.2.3 References Property

The property represents a list of the entity references.

Class

TMappedEntity

Syntax

```
property References: TMappedReferences;
```

Remarks

The TMappedReferences class represents a list of the entity references and provides methods for iterating and accessing the list elements.

See Also

- TMappedReferences

6.17.1.5.3 Methods

Methods of the **TMappedEntity** class.

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

Public

Name	Description
AttributeByName (inherited from **TEntity**)	Returns an entity attribute by its name.
Cancel	Overloaded. The method is designed to cancel changes made in an entity instance.
Compare (inherited from **TEntity**)	The method is designed for comparing the entity key with the specified key.
Create (inherited from **TEntity**)	Overloaded. The constructor is designed for creating a new entity instance.
Delete	Overloaded. The method is designed for deleting an entity.
DeleteAndSave	Overloaded. The method is designed for permanently

	deleting an entity.
FromKey (inherited from **TEntity**)	The method is designed for setting the entity key value.
IsAttached (inherited from **TEntity**)	The method is designed to determine whether the entity is attached.
Save	Overloaded. The method is designed for saving changes made in an entity instance.
ToKey (inherited from **TEntity**)	The method is designed for filling the specified key with the entity key value.

See Also

- TMappedEntity Class
- TMappedEntity Class Members

Request Support DAC Forum Provide Feedback

6.17.1.5.3.1 Cancel Method

The method is designed to cancel changes made in an entity instance.

Class

`TMappedEntity`

Overload List

Name	Description
Cancel	The method is designed to cancel changes made in an entity instance.
Cancel(Cascade: boolean)	The method is designed to cancel changes made in an entity instance.

Request Support DAC Forum Provide Feedback

The method is designed to cancel changes made in an entity instance.

Class

TMappedEntity

Syntax

```
procedure Cancel; overload;
```

Remarks

The method cancels modifications made in an entity instance. If an instance was deleted with TCustomEntityContext.Delete, the entity is restored from the object cache on the method execution, database access doesn't occur. The method cancels only those changes, that were not saved with the TCustomEntityContext.Save or TDataContext.SubmitChanges methods.

The method performs a non-cascade cancelling. For cancel the entity changes cascade use the overloaded Cancel method instead.

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Attach
- TEntity

The method is designed to cancel changes made in an entity instance.

Class

TMappedEntity

Syntax

```
procedure Cancel(Cascade: boolean); overload;
```

Parameters

Cascade

The parameter defines whether to perform cascade cancel of modidications of entity references and linked collections when canceling the entity modifications. The default value is False.

Remarks

The method cancels modifications made in an entity instance. If an instance was deleted with TCustomEntityContext.Delete, the entity is restored from the object cache on the method execution, database access doesn't occur. The method cancels only those changes, that were not saved with the TCustomEntityContext.Save or TDataContext.SubmitChanges methods.

To cancel changes for all entities attached to the data context, the TDataContext.RejectChanges method is used.

See Also

- TCustomEntityContext.CreateEntity
- TCustomEntityContext.CreateAttachedEntity
- TCustomEntityContext.Delete
- TCustomEntityContext.Save
- TCustomEntityContext.Attach
- TEntity

6.17.1.5.3.2 Delete Method

The method is designed for deleting an entity.

Class

TMappedEntity

Overload List

Name	Description
Delete	The method is designed for deleting an entity.
Delete(Cascade: boolean)	The method is designed for deleting an entity.

Request Support DAC Forum Provide Feedback

The method is designed for deleting an entity.

Class

TMappedEntity

Syntax

```
procedure Delete; overload;
```

Remarks

The method deletes the specified entity. The performed deletion is recoverable. When the method is performed, references to the entity are not deleted from linked entities, the entity is not deleted from collections of linked objects. Physical deletion of data from corresponding database structures doesn't occur as well.

To cancel entity deletion, the TMappedEntity.Cancel method is used.

To submit entity deletion, the TMappedEntity.Save method is used.

To permanently delete an entity, the TMappedEntity.DeleteAndSave method is used.

The method performs a non-cascade deletion. To delete the entity cascade, use the overloaded Delete method instead.

See Also

- Delete
- TMappedEntity.Cancel
- TMappedEntity.Save

- TMappedEntity.DeleteAndSave
- TCustomEntityContext.Delete

The method is designed for deleting an entity.

Class

TMappedEntity

Syntax

```
procedure Delete(Cascade: boolean); overload; deprecated;
```

Parameters

Cascade

The parameter defines whether to perform cascade deletion of entity references and linked collections when deleting the entity.

Remarks

The method deletes the specified entity. The performed deletion is recoverable. When the method is performed, references to the entity are not deleted from linked entities, the entity is not deleted from collections of linked objects. Physical deletion of data from corresponding database structures doesn't occur as well.

To cancel entity deletion, the TMappedEntity.Cancel method is used.

To submit entity deletion, the TMappedEntity.Save method is used.

To permanently delete an entity, the TMappedEntity.DeleteAndSave method is used.

The method is an analogue of the TCustomEntityContext.Delete method.

See Also

- TMappedEntity.Cancel
- TMappedEntity.Save
- TMappedEntity.DeleteAndSave
- TCustomEntityContext.Delete

6.17.1.5.3.3 DeleteAndSave Method

The method is designed for permanently deleting an entity.

Class

TMappedEntity

Overload List

Name	Description
DeleteAndSave	The method is designed for permanently deleting an entity.
DeleteAndSave(Cascade: boolean)	The method is designed for permanent deleting an entity.

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

The method is designed for permanently deleting an entity.

Class

TMappedEntity

Syntax

```
procedure DeleteAndSave; overload;
```

Remarks

The method deletes the specified entity. The performed deletion is irreversible. When the method is performed, references to the entity are deleted from linked entities, the entity is deleted from collections of linked entities, physical deletion of data from corresponding database structures occurs as well. The method execution is equivalent to consequent execution of the methods: TMappedEntity.Delete, TMappedEntity.Save.

The method performs a non-cascade deletion. For delete the entity cascade use the overloaded DeleteAndSave method instead.

See Also

- TMappedEntity.Save
- TMappedEntity.Delete
- DeleteAndSave

Request Support DAC Forum Provide Feedback

The method is designed for permanent deleting an entity.

Class

TMappedEntity

Syntax

```
procedure DeleteAndSave(Cascade: boolean); overload;
```

Parameters

Cascade

The parameter defines whether to perform cascade permanent deletion of entity references and linked collections when deleting the entity.

Remarks

The method deletes the specified entity. The performed deletion is irreversible. When the method is performed, references to the entity are deleted from linked entities, the entity is deleted from collections of linked entities, physical deletion of data from corresponding database structures occurs as well. The method execution is equivalent to consequent execution of the methods: TMappedEntity.Delete, TMappedEntity.Save.

The method is the analogue of the TCustomEntityContext.DeleteAndSave method.

See Also

- TMappedEntity.Save
- TMappedEntity.Delete
- TCustomEntityContext.DeleteAndSave

Request Support DAC Forum Provide Feedback

6.17.1.5.3.4 Save Method

The method is designed for saving changes made in an entity instance.

Class

TMappedEntity

Overload List

Name	Description
Save	The method is designed for saving changes made in an entity instance.
Save(Cascade: boolean)	The method is designed for saving changes made in an entity instance.

Request Support DAC Forum Provide Feedback

The method is designed for saving changes made in an entity instance.

Class

TMappedEntity

Syntax

```
procedure Save; overload;
```

Remarks

The method performs permanent saving of modifications made in an entity instance. If the entity was deleted with TMappedEntity.Delete, then when executing this method, there occurs deletion of references to this entity in linked entities, deletion of the entity from linked entities collections, as well as deletion of data from corresponding database structures. The method performs a non-cascade saving. For save the entity cascade use the overloaded TCustomEntityContext.Save method instead.

See Also

- TMappedEntity.Delete
- Save
- TCustomEntityContext.Save

 Request Support DAC Forum Provide Feedback

The method is designed for saving changes made in an entity instance.

Class

TMappedEntity

Syntax

```
procedure Save(Cascade: boolean); overload;
```

Parameters

Cascade

The parameter defines whether to perform cascade saving of changes of entity references and linked collections when saving entity changes.

Remarks

The method performs permanent saving of modifications made in an entity instance. If the entity was deleted with TMappedEntity.Delete, then when executing this method, there occurs deletion of references to this entity in linked entities, deletion of the entity from linked entities collections, as well as deletion of data from corresponding database structures.

The method is the analogue of the TCustomEntityContext.Save method.

See Also

- TMappedEntity.Delete
- TCustomEntityContext.Save

 Request Support DAC Forum Provide Feedback

6.17.1.6 TMappedReference Class

Represents the entity reference.

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

Unit

EntityDAC.EntityContext

Syntax

```
TMappedReference = class(TEntityReference);
```

Remarks

The class represents the entity reference. A reference is a link from the entity to another entity in one-to-one associations, or a link to the parent entity in one-to-many associations.

Inheritance Hierarchy

TEntityReference

TMappedReference

Request Support DAC Forum Provide Feedback

6.17.1.6.1 Members

TMappedReference class overview.

Properties

Name	Description
IsModified (inherited from **TEntityReference**)	Indicates whether the reference is modified.
MetaReference (inherited from **TEntityReference**)	Contains the meta description of the reference.
Value	Provides access to the referenced entity instance.

Methods

Name	Description
MetaType (inherited from **TEntityReference**)	Indicates the meta-type of the entity.

Request Support DAC Forum Provide Feedback

6.17.1.6.2 Properties

Properties of the **TMappedReference** class.

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

Public

Name	Description
IsModified (inherited from **TEntityReference**)	Indicates whether the reference is modified.
MetaReference (inherited from **TEntityReference**)	Contains the meta description of the reference.
Value	Provides access to the referenced entity instance.

See Also

- TMappedReference Class
- TMappedReference Class Members

Request Support DAC Forum Provide Feedback

6.17.1.6.2.1 Value Property

Provides access to the referenced entity instance.

Class

`TMappedReference`

Syntax

```
property Value: TMappedEntity;
```

Remarks

The property provides access to the referenced entity instance.

See Also

- TMappedEntity

 Request Support DAC Forum Provide Feedback

6.17.1.7 TMappedReferences Class

Represents a list of the entity references.

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

Unit

EntityDAC.EntityContext

Syntax

```
TMappedReferences = class(TEntityReferences);
```

Remarks

TMappedReferences contains a list of TMappedReference and provides methods for iterating and accessing the list elements.

Inheritance Hierarchy

TEntityReferences

TMappedReferences

See Also

- TMappedReference

 Request Support DAC Forum Provide Feedback

6.17.1.7.1 Members

TMappedReferences class overview.

Properties

Name	Description
Count (inherited from **TEntityReferences**)	Indicates elements count in the list.
Items	Returns the reference by its index.

Methods

Name	Description
Find	Returns the reference by its name.

Request Support DAC Forum Provide Feedback

6.17.1.7.2 Properties

Properties of the **TMappedReferences** class.

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

Public

Name	Description
Count (inherited from **TEntityReferences**)	Indicates elements count in the list.
Items	Returns the reference by its index.

See Also

- TMappedReferences Class
- TMappedReferences Class Members

Request Support DAC Forum Provide Feedback

Reserved.

6.17.1.7.2.1 Items Property(Indexer)

Returns the reference by its index.

Class

TMappedReferences

Syntax

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

Parameters

Index
The index of the reference.

Remarks

The function returns the TMappedReference element by its specified index.

See Also

- TMappedReference

Request Support DAC Forum Provide Feedback

6.17.1.7.3 Methods

Methods of the **TMappedReferences** class.

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

Public

Name	Description
Find	Returns the reference by its name.

See Also

- TMappedReferences Class

- TMappedReferences Class Members

6.17.1.7.3.1 Find Method

Returns the reference by its name.

Class

TMappedReferences

Syntax

```
function Find(const Name: string): TMappedReference;
```

Parameters

Name
The name of the reference.

Remarks

The function returns the TMappedReference element by its specified name. The name is case sensitive.

See Also

- TMappedReference

6.17.2 Types

Types in the **EntityDAC.EntityContext** unit.

Types

Name	Description
TMappedEntityClass	The class that represents a mapped entity instance and provides methods for managing its.

 Request Support DAC Forum Provide Feedback

6.17.2.1 TMappedEntityClass Class Reference

The class that represents a mapped entity instance and provides methods for managing its.

Unit

EntityDAC.EntityContext

Syntax

```
TMappedEntityClass = class of TMappedEntity;
```

Remarks

TMappedEntity intended to hold the instance of an entity that is mapped to the particular database table. TMappedEntity is updatable, it can be saved to the database or deleted. Those entities that are the result of a query execution and can not be mapped to the particular table, are represented with the TUnmappedEntity class instances.

 Request Support DAC Forum Provide Feedback

6.18 EntityDAC.EntityDataSet

The unit contains implementation of the entity dataset functionality.

Classes

Name	Description
TCustomEntityDataSet	The base class for all datasets that represents entity data.
TCustomEntityTable	The base class that provides table component functionality.
TEntityDataSet	Used to represent entity data from different sources.
TEntityDataSetOptions	Used for setting TEntityDataSet options

TEntityDataSource	Provides an interface for connecting data-aware controls on a form and EntityDAC dataset components.
TEntityQuery	Used to represent a collection of entities that is the result of a query execution.
TEntityTable	Used to represent entities of the particular type.

Request Support DAC Forum Provide Feedback

6.18.1 Classes

Classes in the **EntityDAC.EntityDataSet** unit.

Classes

Name	Description
TCustomEntityDataSet	The base class for all datasets that represents entity data.
TCustomEntityTable	The base class that provides table component functionality.
TEntityDataSet	Used to represent entity data from different sources.
TEntityDataSetOptions	Used for setting TEntityDataSet options
TEntityDataSource	Provides an interface for connecting data-aware controls on a form and EntityDAC dataset components.
TEntityQuery	Used to represent a collection of entities that is the result of a query execution.
TEntityTable	Used to represent entities of the particular type.

 Request Support DAC Forum Provide Feedback

6.18.1.1 TCustomEntityDataSet Class

The base class for all datasets that represents entity data.

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

Unit

EntityDAC.EntityDataSet

Syntax

```
TCustomEntityDataSet = class(TEDCustomVirtualDataSet);
```

Remarks

TCustomEntityDataSet is a base class for all datasets that represents entity data. It defines basic functionality for a dataset. Since TCustomEntityDataSet is a base class, it should not be used directly. Instead, TCustomEntityDataSet descendants such as TEntityDataSet, TEntityTable and TEntityQuery have to be used.

See Also

- TEntityDataSet
- TEntityTable
- TEntityQuery

 Request Support DAC Forum Provide Feedback

6.18.1.1.1 Members

TCustomEntityDataSet class overview.

Properties

Name	Description
Context	Identifies the data context for which the dataset represents

	entity data.
FieldExpressions	Lists all field expressions of the dataset.
Options	The class allows setting up the behavior of the TCustomEntityDataSet class.

Methods

Name	Description
AddFieldExpression	Overloaded.Description is not available at the moment.
ClearFieldExpressions	Deletes all field expressions in the dataset.
Current<T>	Returns the current entity in the dataset.
CurrentEntity	Returns the current entity in the dataset.
CurrentObject	Returns the current entity in the dataset.
DeleteFieldExpression	Overloaded.Description is not available at the moment.

Events

Name	Description
OnDelete	Occurs before an application deletes the current entity from the database.
OnPost	Occurs before an application saves changes for the current entity to the database.

6.18.1.1.2 Properties

Properties of the **TCustomEntityDataSet** class.

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

Published

Name	Description
Context	Identifies the data context for which the dataset represents entity data.
FieldExpressions	Lists all field expressions of the dataset.
Options	The class allows setting up the behavior of the TCustomEntityDataSet class.

See Also

- TCustomEntityDataSet Class
- TCustomEntityDataSet Class Members

6.18.1.1.2.1 Context Property

Identifies the data context for which the dataset represents entity data.

Class

TCustomEntityDataSet

Syntax

```
property Context: TDataContext;
```

Remarks

Set the property to associate the dataset with a data context. Use the property to access

properties, events and methods of the data context associated with the dataset.

See Also

- TDataContext

 Request Support DAC Forum Provide Feedback

6.18.1.1.2.2 FieldExpressions Property

Lists all field expressions of the dataset.

Class

TCustomEntityDataSet

Syntax

```
property FieldExpressions: TFieldExpressions;
```

Remarks

The property is used to access field expressions of the dataset. The field expressions mechanism allows to compose complex expressions using operations with entity attributes, and create corresponding dataset fields based on these expressions. Field expressions by something are similar to calculated fields. The main difference is that the logic of computation of a calculated field needs to be written in the handler of the TDataSet.OnCalcFields event, and a field expression can be defined both at run-time, and at design-time.

To manage field expressions at run-time, AddFieldExpression, DeleteFieldExpression and ClearFieldExpressions methods are used.

See Also

- AddFieldExpression
- DeleteFieldExpression
- ClearFieldExpressions

 Request Support DAC Forum Provide Feedback

6.18.1.1.2.3 Options Property

The class allows setting up the behavior of the TCustomEntityDataSet class.

Class

TCustomEntityDataSet

Syntax

```
property Options: TEntityDataSetOptions;
```

See Also

- TEntityDataSetOptions

Request Support DAC Forum Provide Feedback

6.18.1.1.3 Methods

Methods of the **TCustomEntityDataSet** class.

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

Public

Name	Description
AddFieldExpression	Overloaded.Description is not available at the moment.
ClearFieldExpressions	Deletes all field expressions in the dataset.
Current<T>	Returns the current entity in the dataset.
CurrentEntity	Returns the current entity in the dataset.
CurrentObject	Returns the current entity in the dataset.
DeleteFieldExpression	Overloaded.Description is not available at the moment.

See Also

- TCustomEntityDataSet Class
- TCustomEntityDataSet Class Members

 Request Support DAC Forum Provide Feedback

6.18.1.1.3.1 AddFieldExpression Method

Class

TCustomEntityDataSet

Overload List

Name	Description
AddFieldExpression(**const** Expression: TExpression; StrictReferences: boolean)	Adds new field expression to the dataset.
AddFieldExpression(**const** FieldName: **string**; **const** Expression: TExpression; StrictReferences: boolean)	Adds new field expression to the dataset.
AddFieldExpression(**const** Expression: **string**; StrictReferences: boolean)	Adds new field expression to the dataset.
AddFieldExpression(**const** FieldName: **string**; **const** Expression: **string**; StrictReferences: boolean)	Adds new field expression to the dataset.

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

Adds new field expression to the dataset.

Class

TCustomEntityDataSet

Syntax

```
function AddFieldExpression(const Expression: TExpression;
StrictReferences: boolean = False): TFieldExpression; overload;
```

Parameters

Expression

Defines the field expression.

StrictReferences

Return Value

Added field expression.

Remarks

The method adds new field expression to the dataset. The name of the dataset field being created will be set automatically, according to the specified expression. To explicitly specify the name for the resulting field, overloaded AddFieldExpression methods can be used. To delete the field expression, TCustomEntityDataSet.DeleteFieldExpression method is used. To clear all dataset field expressions, the TCustomEntityDataSet.ClearFieldExpressions method is used.

See Also

- AddFieldExpression
- TCustomEntityDataSet.DeleteFieldExpression
- TCustomEntityDataSet.ClearFieldExpressions

Request Support DAC Forum Provide Feedback

Adds new field expression to the dataset.

Class

TCustomEntityDataSet

Syntax

```
function AddFieldExpression(const FieldName: string; const
Expression: TExpression; StrictReferences: boolean = False):
TFieldExpression; overload;
```

Parameters

FieldName

Defines the name of the dataset field.

Expression

Defines the field expression.

StrictReferences

Return Value

Added field expression.

Remarks

The method adds new field expression to the dataset.

To delete the field expression, TCustomEntityDataSet.DeleteFieldExpression method is used. To clear all dataset field expressions, the TCustomEntityDataSet.ClearFieldExpressions method is used.

See Also

- TCustomEntityDataSet.DeleteFieldExpression
- TCustomEntityDataSet.ClearFieldExpressions

Request Support DAC Forum Provide Feedback

Adds new field expression to the dataset.

Class

TCustomEntityDataSet

Syntax

```
function AddFieldExpression(const Expression: string;
StrictReferences: boolean = False): TFieldExpression; overload;
```

Parameters

Expression

Defines the field expression in the string format.

StrictReferences

Return Value

Added field expression.

Remarks

The method adds new field expression to the dataset. The expression has to be specified in the string format using rules described in the article. The name of the dataset field being created will be set automatically, according to the specified expression. To explicitly specify

the name for the resulting field, the overloaded AddFieldExpression method can be used.

To delete the field expression, TCustomEntityDataSet.DeleteFieldExpression method is used. To clear all dataset field expressions, the TCustomEntityDataSet.ClearFieldExpressions methods is used.

See Also

- AddFieldExpression
- TCustomEntityDataSet.DeleteFieldExpression
- TCustomEntityDataSet.ClearFieldExpressions

Adds new field expression to the dataset.

Class

TCustomEntityDataSet

Syntax

```
function AddFieldExpression(const FieldName: string; const
Expression: string; StrictReferences: boolean = False):
TFieldExpression; overload;
```

Parameters

FieldName

Defines the name of the dataset field.

Expression

Defines the field expression in the string format.

StrictReferences

Return Value

Added field expression.

Remarks

The method adds new field expression to the dataset. The expression has to be specified in the string format using rules described in the article. To delete the field expression, TCustomEntityDataSet.DeleteFieldExpression method is used. To clear all dataset field

expressions, the TCustomEntityDataSet.ClearFieldExpressions method is used.

See Also

- TCustomEntityDataSet.DeleteFieldExpression
- TCustomEntityDataSet.ClearFieldExpressions

Request Support DAC Forum Provide Feedback

6.18.1.1.3.2 ClearFieldExpressions Method

Deletes all field expressions in the dataset.

Class

TCustomEntityDataSet

Syntax

```
procedure ClearFieldExpressions;
```

Remarks

The method deletes all field expressions in the dataset and removes all corresponding dataset fields.

To add new field expression, AddFieldExpression method is used. To delete the field expression, DeleteFieldExpression method is used.

Request Support DAC Forum Provide Feedback

6.18.1.1.3.3 Current<T> Method

Returns the current entity in the dataset.

Class

TCustomEntityDataSet

Syntax

```
function Current<T: class>: T;
```

Type parameters

T

Specifies the class type of the entity to be returned.

Remarks

Read the property to access the current entity in the dataset as a specified class instance.

When the dataset is TEntityDataSet and its source is defined using TEntityDataSet.SourceObject or TEntityDataSet.SourceEntity then the Current<T> property returns the source entity.

When the dataset is TEntityDataSet and its source is defined using the TEntityDataSet.SourceCollection property then the Current<T> property returns the current entity in the source collection.

For TEntityTable and TEntityQuery, the Current<T> property returns the current entity in the dataset.

See Also

- TEntityDataSet
- TEntityDataSet.SourceObject
- TEntityDataSet.SourceEntity
- TEntityDataSet.SourceCollection
- TEntityTable
- TEntityQuery

Request Support DAC Forum Provide Feedback

6.18.1.1.3.4 CurrentEntity Method

Returns the current entity in the dataset.

Class

`TCustomEntityDataSet`

Syntax

```
function CurrentEntity: TEntity;
```

Remarks

Read the property to access the current entity in the dataset as TEntity instance.

When the dataset is TEntityDataSet and its source is defined using TEntityDataSet.SourceObject or TEntityDataSet.SourceEntity then the CurrentEntity property returns the source entity.

When the dataset is TEntityDataSet and its source is defined using the TEntityDataSet.SourceCollection property then the CurrentEntity property returns the current entity in the source collection.

For TEntityTable and TEntityQuery, the CurrentEntity property returns the current entity in the dataset.

See Also

- TEntityDataSet
- TEntityDataSet.SourceObject
- TEntityDataSet.SourceEntity
- TEntityDataSet.SourceCollection
- TEntityTable
- TEntityQuery

6.18.1.1.3.5 CurrentObject Method

Returns the current entity in the dataset.

Class

TCustomEntityDataSet

Syntax

```
function CurrentObject: TObject;
```

Remarks

Read the property to access the current entity in the dataset as TObject instance.

When the dataset is TEntityDataSet and its source is defined using TEntityDataSet.SourceObject or TEntityDataSet.SourceEntity then the CurrentObject property returns the source entity.

When the dataset is TEntityDataSet and its source is defined using the TEntityDataSet.SourceCollection property then the CurrentObject property returns the current entity in the source collection.

For TEntityTable and TEntityQuery, the CurrentObject property returns the current entity in the dataset.

See Also

- TEntityDataSet
- TEntityDataSet.SourceObject
- TEntityDataSet.SourceEntity
- TEntityDataSet.SourceCollection
- TEntityTable
- TEntityQuery

6.18.1.1.3.6 DeleteFieldExpression Method

Class

`TCustomEntityDataSet`

Overload List

Name	Description
DeleteFieldExpression(Index: integer)	Deletes a field expression by index.
DeleteFieldExpression(**const** FieldName: **string**)	Deletes a field expression by the field name.

Deletes a field expression by index.

Class

TCustomEntityDataSet

Syntax

```
procedure DeleteFieldExpression(Index: integer); overload;
```

Parameters

Index

Defines the index of the field expression.

Remarks

The method deletes a field expression by its index in the TCustomEntityDataSet.FieldExpressions list. Also, a field expression can be deleted by its field name using the overloaded DeleteFieldExpression method.

To add new field expression, TCustomEntityDataSet.AddFieldExpression method is used. To clear all dataset field expressions, the TCustomEntityDataSet.ClearFieldExpressions method is used.

See Also

- DeleteFieldExpression
- TCustomEntityDataSet.AddFieldExpression
- TCustomEntityDataSet.ClearFieldExpressions

Deletes a field expression by the field name.

Class

TCustomEntityDataSet

Syntax

```
procedure DeleteFieldExpression(const FieldName: string);
overload;
```

Parameters

FieldName

Defines the name of the dataset field.

Remarks

The method deletes a field expression by its field name. Also, a field expression can be deleted by its index using the overloaded DeleteFieldExpression method.

To add new field expression, TCustomEntityDataSet.AddFieldExpression method is used. To clear all dataset field expressions, the TCustomEntityDataSet.ClearFieldExpressions method is used.

See Also

- DeleteFieldExpression
- TCustomEntityDataSet.AddFieldExpression
- TCustomEntityDataSet.ClearFieldExpressions

6.18.1.1.4 Events

Events of the **TCustomEntityDataSet** class.

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

Published

Name	Description
OnDelete	Occurs before an application deletes the current entity from the database.

OnPost	Occurs before an application saves changes for the current entity to the database.

See Also

- TCustomEntityDataSet Class
- TCustomEntityDataSet Class Members

Request Support DAC Forum Provide Feedback

6.18.1.1.4.1 OnDelete Event

Occurs before an application deletes the current entity from the database.

Class

TCustomEntityDataSet

Syntax

```
property OnDelete: TOnModifyEvent;
```

Remarks

Write a OnDelete event handler to take specific action when the dataset deletes the current entity from the database. OnDelete is triggered when an application calls the Delete method. Delete checks whether the entity corresponding to the currect record exists, calls the BeforeDelete event, then retrieves the current entity and calls the OnDelete event.

In the OnDelete event handler, the DataSetAction parameter has the daDelete value.

The ApplyAction parameter defines which actions have to be performed to the current entity after the event is called:

- aaSave - the entity will be deleted from the database;
- aaUpdateCollection - the entity will be removed from the internal dataset collection.

Request Support DAC Forum Provide Feedback

6.18.1.1.4.2 OnPost Event

Occurs before an application saves changes for the current entity to the database.

Class

TCustomEntityDataSet

Syntax

```
property OnPost: TOnModifyEvent;
```

Remarks

Write a OnPost event handler to take specific action when the dataset applies changes for the current record to the current entity and before the application saves the entity changes. OnPost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, calls the BeforePost event, then applies data changes to the current entity and calls the OnPost event.

In the OnPost event handler, the DataSetAction parameter indicates the current dataset operation:

- daInsert - a new record is inserted;
- daEdit - an existing record is modified.

The ApplyAction parameter defines which actions have to be performed to the current entity after the event is called:

- aaSave - the entity will be saved to the database;
- aaUpdateCollection - the entity will be added/modified in the internal dataset collection.

Request Support DAC Forum Provide Feedback

6.18.1.2 TCustomEntityTable Class

The base class that provides table component functionality.

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

Unit

EntityDAC.EntityDataSet

Syntax

```
TCustomEntityTable = class(TCustomEntityDataSet);
```

Remarks

TCustomEntityTable is a base class that provides table component functionality.

Since TCustomEntityTable is the base class, it should not be used directly. Instead, TCustomEntityTable descendants such as TEntityTable have to be used.

Inheritance Hierarchy

TCustomEntityDataSet

TCustomEntityTable

See Also

- TEntityTable

6.18.1.2.1 Members

TCustomEntityTable class overview.

Properties

Name	Description
Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.
TypeName	Specifies the meta-type of

	entities with which the dataset will be filled.

Methods

Name	Description
AddFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.
ClearFieldExpressions (inherited from **TCustomEntityDataSet**)	Deletes all field expressions in the dataset.
Current<T> (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
CurrentEntity (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
CurrentObject (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
DeleteFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.

Events

Name	Description
OnDelete (inherited from **TCustomEntityDataSet**)	Occurs before an application deletes the current entity from the database.
OnPost (inherited from **TCustomEntityDataSet**)	Occurs before an application saves changes for the current entity to the database.

6.18.1.2.2 Properties

Properties of the **TCustomEntityTable** class.

For a complete list of the **TCustomEntityTable** class members, see the

TCustomEntityTable Members topic.

Public

Name	Description
TypeName	Specifies the meta-type of entities with which the dataset will be filled.

Published

Name	Description
Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.

See Also

- TCustomEntityTable Class
- TCustomEntityTable Class Members

6.18.1.2.2.1 TypeName Property

Specifies the meta-type of entities with which the dataset will be filled.

Class

TCustomEntityTable

Syntax

```
property TypeName: string;
```

Remarks

The property is designed to set the meta-type of entities with which the dataset will be filled. To set MetaTypeName to a meaningful value, the TEntityContext.ModelName property should already be set. If MetaTypeName is set at design time, then a meta-type name can be selected from the drop-down list in the Object Inspector.

See Also

- TEntityContext.ModelName

Request Support DAC Forum Provide Feedback

6.18.1.3 TEntityDataSet Class

Used to represent entity data from different sources.

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

Unit

EntityDAC.EntityDataSet

Syntax

```
TEntityDataSet = class(TCustomEntityDataSet);
```

Remarks

The class is used to represent entity data from different sources such as a single entity or an entity collection. For represent all entities of the particular type, the TEntityTable component can be used. For represent a collection of entities that is the result of a query execution, TEntityQuery can be used.

Inheritance Hierarchy

TCustomEntityDataSet
  TEntityDataSet

See Also

- TEntityTable

- TEntityQuery

6.18.1.3.1 Members

TEntityDataSet class overview.

Properties

Name	Description
Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.
SourceCollection	Sets an entity collection as the dataset source.
SourceEntity	Sets a single entity as the dataset source.
SourceObject	Sets a single entity as the dataset source.

Methods

Name	Description
AddFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.
ClearFieldExpressions (inherited from **TCustomEntityDataSet**)	Deletes all field expressions in the dataset.
Current<T> (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
CurrentEntity (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.

CurrentObject (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
DeleteFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.

Events

Name	Description
OnDelete (inherited from **TCustomEntityDataSet**)	Occurs before an application deletes the current entity from the database.
OnPost (inherited from **TCustomEntityDataSet**)	Occurs before an application saves changes for the current entity to the database.

Request Support DAC Forum Provide Feedback

6.18.1.3.2 Properties

Properties of the **TEntityDataSet** class.

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

Public

Name	Description
SourceCollection	Sets an entity collection as the dataset source.
SourceEntity	Sets a single entity as the dataset source.
SourceObject	Sets a single entity as the dataset source.

Published

Name	Description

Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.

See Also

- TEntityDataSet Class
- TEntityDataSet Class Members

Request Support DAC Forum Provide Feedback

6.18.1.3.2.1 SourceCollection Property

Sets an entity collection as the dataset source.

Class

`TEntityDataSet`

Syntax

```
property SourceCollection: IEnumerable;
```

Remarks

The property is designed to set a collection of entities as the dataset source. To set a single entity as the dataset source, SourceObject and SourceEntity methods are used.

See Also

- SourceObject
- SourceEntity

Request Support DAC Forum Provide Feedback

Reserved.

6.18.1.3.2.2 SourceEntity Property

Sets a single entity as the dataset source.

Class

TEntityDataSet

Syntax

```
property SourceEntity: TEntity;
```

Remarks

The property is designed to set a single TEntity descendant as the dataset source. When the entity is not the TEntity descendant, the SourceObject method can be used. To set an entity collection as the dataset source, the SourceCollection method is used.

See Also

- SourceCollection
- SourceObject

Request Support DAC Forum Provide Feedback

6.18.1.3.2.3 SourceObject Property

Sets a single entity as the dataset source.

Class

TEntityDataSet

Syntax

```
property SourceObject: TObject;
```

Remarks

The property is designed to set a single entity as the dataset source. The property is useful when attribute-mapped objects are used, and the entity is a trivial class instance. When the

entity is TEntity descendant, the SourceEntity method can be used. To set an entity collection as the dataset source, the SourceCollection method is used.

See Also

- SourceEntity
- SourceCollection

 Request Support DAC Forum Provide Feedback

6.18.1.4 TEntityDataSetOptions Class

Used for setting TEntityDataSet options

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

Unit

EntityDAC.EntityDataSet

Syntax

```
TEntityDataSetOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

6.18.1.4.1 Members

TEntityDataSetOptions class overview.

Properties

Name	Description
SaveOnPost	Determines, whether the modified entity is saved automatically when the Post method is executed.
SyncFieldValues	Determines, where the field modification is immediately reflected in the corresponding entity

	attribute.

Request Support DAC Forum Provide Feedback

6.18.1.4.2 Properties

Properties of the **TEntityDataSetOptions** class.

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

Published

Name	Description
SaveOnPost	Determines, whether the modified entity is saved automatically when the Post method is executed.
SyncFieldValues	Determines, where the field modification is immediately reflected in the corresponding entity attribute.

See Also

- TEntityDataSetOptions Class
- TEntityDataSetOptions Class Members

Request Support DAC Forum Provide Feedback

6.18.1.4.2.1 SaveOnPost Property

Determines, whether the modified entity is saved automatically when the Post method is executed.

Class

`TEntityDataSetOptions`

Syntax

```
property SaveOnPost: Boolean default True;
```

Remarks

If the property is set to True then the TEntity.Save method is called for the current entity in the dataset when the Post method is executed. When the property is False then modified entity is not saved automatically on Post, and has to be saved manually. The default value is True.

See Also

- TEntity

Request Support DAC Forum Provide Feedback

6.18.1.4.2.2 SyncFieldValues Property

Determines, where the field modification is immediately reflected in the corresponding entity attribute.

Class

TEntityDataSetOptions

Syntax

```
property SyncFieldValues: boolean default True;
```

Remarks

If the property is set to True then when modifying a field, the changed value immediately assigned to the corresponding entity attribute. If the property is False then the record changes reflected to the corresponding entity at once on the Post execution. The default value is True.

See Also

- TEntity

Request Support DAC Forum Provide Feedback

6.18.1.5 TEntityDataSource Class

Provides an interface for connecting data-aware controls on a form and EntityDAC dataset components.

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

Unit

EntityDAC.EntityDataSet

Syntax

```
TEntityDataSource = class(TDataSource);
```

 Request Support DAC Forum Provide Feedback

6.18.1.5.1 Members

TEntityDataSource class overview.

 Request Support DAC Forum Provide Feedback

6.18.1.6 TEntityQuery Class

Used to represent a collection of entities that is the result of a query execution.

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

Unit

EntityDAC.EntityDataSet

Syntax

```
TEntityQuery = class(TCustomEntityDataSet);
```

Remarks

The class is used to represent a collection of entities that is the result of a query execution. For represent entity data from different sources such as a single entity or an entity collection, the TEntityDataSet component can be used. For represent entities of the particular type,

TEntityTable can be used.

Inheritance Hierarchy

TCustomEntityDataSet

TEntityQuery

See Also

- TEntityDataSet
- TEntityTable

6.18.1.6.1 Members

TEntityQuery class overview.

Properties

Name	Description
Active	Specifies whether or not the dataset is open.
Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
LINQ	Contains the text of the LINQ statement to execute for the query.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.

Methods

Name	Description

AddFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.
ClearFieldExpressions (inherited from **TCustomEntityDataSet**)	Deletes all field expressions in the dataset.
Current<T> (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
CurrentEntity (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
CurrentObject (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
DeleteFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.

Events

Name	Description
OnDelete (inherited from **TCustomEntityDataSet**)	Occurs before an application deletes the current entity from the database.
OnPost (inherited from **TCustomEntityDataSet**)	Occurs before an application saves changes for the current entity to the database.

6.18.1.6.2 Properties

Properties of the **TEntityQuery** class.

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

Published

Name	Description

Active	Specifies whether or not the dataset is open.
Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
LINQ	Contains the text of the LINQ statement to execute for the query.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.

See Also

- TEntityQuery Class
- TEntityQuery Class Members

Request Support DAC Forum Provide Feedback

6.18.1.6.2.1 Active Property

Specifies whether or not the dataset is open.

Class

TEntityQuery

Syntax

```
property Active;
```

Remarks

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed, the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data.

Request Support DAC Forum Provide Feedback

Reserved.

6.18.1.6.2.2 LINQ Property

Contains the text of the LINQ statement to execute for the query.

Class

TEntityQuery

Syntax

```
property LINQ: TStrings;
```

Remarks

The property is used to provide the LINQ statement that the query component executes when its Open method is called. At design time the LINQ property can be edited by invoking the String List editor in the Object Inspector.

The LINQ property may contain only one complete LINQ statement at a time.

6.18.1.7 TEntityTable Class

Used to represent entities of the particular type.

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

Unit

EntityDAC.EntityDataSet

Syntax

```
TEntityTable = class(TCustomEntityTable);
```

Remarks

The class is used to represent entities of the particular type. To represent entity data from different sources, such as a single entity or an entity collection, the TEntityDataSet component can be used. To represent a collection of entities, that is the result of query

execution, TEntityQuery can be used.

Inheritance Hierarchy

TCustomEntityDataSet
  TCustomEntityTable
    TEntityTable

See Also

- TEntityDataSet
- TEntityQuery

Request Support DAC Forum Provide Feedback

6.18.1.7.1 Members

TEntityTable class overview.

Properties

Name	Description
Active	Specifies whether or not the dataset is open.
Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.
TypeName	Specifies the meta-type of entities with which the dataset will be filled.

Methods

Name	Description

AddFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.
ClearFieldExpressions (inherited from **TCustomEntityDataSet**)	Deletes all field expressions in the dataset.
Current<T> (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
CurrentEntity (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
CurrentObject (inherited from **TCustomEntityDataSet**)	Returns the current entity in the dataset.
DeleteFieldExpression (inherited from **TCustomEntityDataSet**)	Overloaded.Description is not available at the moment.

Events

Name	Description
OnDelete (inherited from **TCustomEntityDataSet**)	Occurs before an application deletes the current entity from the database.
OnPost (inherited from **TCustomEntityDataSet**)	Occurs before an application saves changes for the current entity to the database.

6.18.1.7.2 Properties

Properties of the **TEntityTable** class.

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

Published

Name	Description
Active	Specifies whether or not the dataset is open.

Context (inherited from **TCustomEntityDataSet**)	Identifies the data context for which the dataset represents entity data.
FieldExpressions (inherited from **TCustomEntityDataSet**)	Lists all field expressions of the dataset.
Options (inherited from **TCustomEntityDataSet**)	The class allows setting up the behavior of the TCustomEntityDataSet class.
TypeName	Specifies the meta-type of entities with which the dataset will be filled.

See Also

- TEntityTable Class
- TEntityTable Class Members

Request Support DAC Forum Provide Feedback

6.18.1.7.2.1 Active Property

Specifies whether or not the dataset is open.

Class

TEntityTable

Syntax

```
property Active;
```

Remarks

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed, the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data.

Request Support DAC Forum Provide Feedback

6.18.1.7.2.2 TypeName Property

Specifies the meta-type of entities with which the dataset will be filled.

Class

TEntityTable

Syntax

```
property TypeName: string;
```

Remarks

The property is designed to set the meta-type of entities with which the dataset will be filled. To set MetaTypeName to a meaningful value, the TEntityContext.ModelName property should already be set. If MetaTypeName is set at design time, then a meta-type name can be selected from the drop-down list in the Object Inspector.

See Also

- TEntityContext.ModelName

6.19 EntityDAC.EntityXMLModel

The unit contains implementation of the XML mapping.

Classes

Name	Description
TCustomEntityModel	Implements the XML mapping and is used for configure EntityDAC components at design-time.
TEntityModelOptions	Specifies the additional options for the model.
TEntityXMLModel	Implements the XML mapping and is used for configure EntityDAC components at design-time.

 Request Support DAC Forum Provide Feedback

6.19.1 Classes

Classes in the **EntityDAC.EntityXMLModel** unit.

Classes

Name	Description
TCustomEntityModel	Implements the XML mapping and is used for configure EntityDAC components at design-time.
TEntityModelOptions	Specifies the additional options for the model.
TEntityXMLModel	Implements the XML mapping and is used for configure EntityDAC components at design-time.

 Request Support DAC Forum Provide Feedback

6.19.1.1 TCustomEntityModel Class

Implements the XML mapping and is used for configure EntityDAC components at design-time.

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

Unit

EntityDAC.EntityXMLModel

Syntax

```
TCustomEntityModel = class(TComponent);
```

Remarks

The component is designed to implement the XML mapping. Also, it is used for configure other EntityDAC components such as TEntityConnection, TEntityContext, TEntityTable,

TEntityQuery at design-time.

See Also

- TEntityConnection
- TEntityContext
- TEntityTable
- TEntityQuery

 Request Support DAC Forum Provide Feedback

6.19.1.1.1 Members

TCustomEntityModel class overview.

Properties

Name	Description
FileName	Specifies the path to the XML-mapping file.
Options	Specifies the additional options for the model.

 Request Support DAC Forum Provide Feedback

6.19.1.1.2 Properties

Properties of the **TCustomEntityModel** class.

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

Public

Name	Description
FileName	Specifies the path to the XML-mapping file.
Options	Specifies the additional options for the model.

See Also

- TCustomEntityModel Class
- TCustomEntityModel Class Members

6.19.1.1.2.1 FileName Property

Specifies the path to the XML-mapping file.

Class

TCustomEntityModel

Syntax

```
property FileName: string;
```

Remarks

Set the property value to specify a file in which the XML mapping is defined. The file specified in FileName is loaded automatically, and the meta-model defined in the file is created and registered in the meta model manager.

If the umDesignTime is specified in TEntityModelOptions.Usage, then the meta model becomes accessible at design-time, so properties such as TEntityConnection.DefaultModelName, TEntityContext.ModelName and TEntityTable.TypeName can be configured.

If the umRunTime is specified in TEntityModelOptions.Usage, then the meta model can be used at run-time. For mapping meta model to the entity classes, the special [XmlMapped] class attribute is used. See the A:xml-mapped-entities.htm article for details.

TEntityModel supports loading either of EntityDeveloper project files (*.enml) or generated XML-mapping files (*.xml). When the file specified is not a valid XML-mapping file, the exception is raised.

See Also

- Options

- TEntityConnection.DefaultModelName
- TEntityContext.ModelName
- TEntityTable.TypeName

 Request Support DAC Forum Provide Feedback

6.19.1.1.2.2 Options Property

Specifies the additional options for the model.

Class

TCustomEntityModel

Syntax

```
property Options: TEntityModelOptions;
```

 Request Support DAC Forum Provide Feedback

6.19.1.2 TEntityModelOptions Class

Specifies the additional options for the model.

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

Unit

EntityDAC.EntityXMLModel

Syntax

```
TEntityModelOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

6.19.1.2.1 Members

TEntityModelOptions class overview.

Properties

Name	Description
Usage	Specifies when the model is used.

Request Support DAC Forum Provide Feedback

6.19.1.2.2 Properties

Properties of the **TEntityModelOptions** class.

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

Published

Name	Description
Usage	Specifies when the model is used.

See Also

- TEntityModelOptions Class
- TEntityModelOptions Class Members

Request Support DAC Forum Provide Feedback

6.19.1.2.2.1 Usage Property

Specifies when the model is used.

Class

TEntityModelOptions

Syntax

```
property Usage: TUsageMode default [umDesignTime, umRunTime];
```

Remarks

The property specifies when the XML model is used.

If the umDesignTime is specified in TEntityModelOptions.Usage, then the meta model becomes accessible at design-time, so properties such as TEntityConnection.DefaultModelName, TEntityContext.ModelName and TEntityTable.TypeName can be configured.

If the umRunTime is specified in TEntityModelOptions.Usage, then the meta model can be used at run-time. For mapping meta model to the entity classes, the special [XmlMapped] class attribute is used. See the XML-mapped entities article for details.

See Also

- TEntityConnection.DefaultModelName
- TEntityContext.ModelName
- TEntityTable.TypeName

Request Support DAC Forum Provide Feedback

6.19.1.3 TEntityXMLModel Class

Implements the XML mapping and is used for configure EntityDAC components at design-time.

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

Unit

EntityDAC.EntityXMLModel

Syntax

```
TEntityXMLModel = class(TCustomEntityModel);
```

Remarks

The component is designed to implement the XML mapping. Also, it is used for configure other EntityDAC components such as TEntityConnection, TEntityContext, TEntityTable,

TEntityQuery at design-time.

Inheritance Hierarchy

TCustomEntityModel

TEntityXMLModel

See Also

- TEntityConnection
- TEntityContext
- TEntityTable
- TEntityQuery

Request Support DAC Forum Provide Feedback

6.19.1.3.1 Members

TEntityXMLModel class overview.

Properties

Name	Description
FileName	Specifies the path to the XML-mapping file.
Options	Specifies the additional options for the model.

Request Support DAC Forum Provide Feedback

6.19.1.3.2 Properties

Properties of the **TEntityXMLModel** class.

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

Published

Name	Description
FileName	Specifies the path to the XML-mapping file.
Options	Specifies the additional options for the model.

See Also

- TEntityXMLModel Class
- TEntityXMLModel Class Members

Request Support DAC Forum Provide Feedback

6.19.1.3.2.1 FileName Property

Specifies the path to the XML-mapping file.

Class

TEntityXMLModel

Syntax

```
property FileName: string;
```

Remarks

Set the property value to specify a file in which the XML mapping is defined. The file specified in FileName is loaded automatically, and the meta-model defined in the file is created and registered in the meta model manager.

If the umDesignTime is specified in TEntityModelOptions.Usage, then the meta model becomes accessible at design-time, so properties such as TEntityConnection.DefaultModelName, TEntityContext.ModelName and TEntityTable.TypeName can be configured.

If the umRunTime is specified in TEntityModelOptions.Usage, then the meta model can be used at run-time. For mapping meta model to the entity classes, the special [XmlMapped] class attribute is used. See the A:xml-mapped-entities.htm article for details.

TEntityModel supports loading either of EntityDeveloper project files (*.enml) or generated

XML-mapping files (*.xml). When the file specified is not a valid XML-mapping file, the exception is raised.

See Also

- TCustomEntityModel.Options
- TEntityConnection.DefaultModelName
- TEntityContext.ModelName
- TEntityTable.TypeName

Request Support DAC Forum Provide Feedback

6.19.1.3.2.2 Options Property

Specifies the additional options for the model.

Class

TEntityXMLModel

Syntax

```
property Options: TEntityModelOptions;
```


Request Support DAC Forum Provide Feedback

6.20 EntityDAC.Enumerable

The unit contains implementation of the entity enumeration.

Classes

Name	Description
TObjectEnumerable<T>	The base class for representing an entities enumeration.

Interfaces

Name	Description
IObjectEnumerable<T>	The base interface which declares functionality for entity enumerations.

Request Support DAC Forum Provide Feedback

6.20.1 Classes

Classes in the **EntityDAC.Enumerable** unit.

Classes

Name	Description
TObjectEnumerable<T>	The base class for representing an entities enumeration.

Request Support DAC Forum Provide Feedback

6.20.1.1 TObjectEnumerable<T> Class

The base class for representing an entities enumeration.

For a list of all members of this type, see TObjectEnumerable<T> members.

Unit

EntityDAC.Enumerable

Syntax

```
TObjectEnumerable<T: class> = class(TCustomEnumerable,
IObjectEnumerable, IEnumerable);
```

Remarks

TObjectEnumerable is the base class for representing an enumeration of entities. TObjectEnumerable supports the IObjectEnumerable<T> interface and implements methods for iterating through the enumeration and access its elements. TObjectEnumerable is the abstract class and should not be used directly. TObjectCollection<T> and TEntityCollection

classes which are inherited from TObjectEnumerable, are used as a base to handling entities collections in the code.

See Also

- IObjectEnumerable<T>
- TObjectCollection<T>
- TEntityCollection

 Request Support DAC Forum Provide Feedback

6.20.1.1.1 Members

TObjectEnumerable<T> class overview.

Properties

Name	Description
Items	Provides indexed access to elements in the enumeration.

Methods

Name	Description
Contains	Checks whether the enumeration contains the specified entity instance.
Count	Indicates the number of elements in the enumeration.
First	Returns the first element of the enumeration.
Last	Returns the last element of the enumeration.
MetaType	Indicates the meta-type of the enumeration elements.
Single	Returns the only element in the enumeration.
ToList	Returns the enumeration as a TList instance.

 Request Support DAC Forum Provide Feedback

6.20.1.1.2 Properties

Properties of the **TObjectEnumerable<T>** class.

For a complete list of the **TObjectEnumerable<T>** class members, see the TObjectEnumerable<T> Members topic.

Public

Name	Description
Items	Provides indexed access to elements in the enumeration.

See Also

- TObjectEnumerable<T> Class
- TObjectEnumerable<T> Class Members

Request Support DAC Forum Provide Feedback

6.20.1.1.2.1 Items Property(Indexer)

Provides indexed access to elements in the enumeration.

Class

TObjectEnumerable<T>

Syntax

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

Parameters

Index
The zero-based index of the element.

Remarks

The property returns the enumeration element by its specified index. The property is read-only.

6.20.1.1.3 Methods

Methods of the **TObjectEnumerable<T>** class.

For a complete list of the **TObjectEnumerable<T>** class members, see the TObjectEnumerable<T> Members topic.

Public

Name	Description
Contains	Checks whether the enumeration contains the specified entity instance.
Count	Indicates the number of elements in the enumeration.
First	Returns the first element of the enumeration.
Last	Returns the last element of the enumeration.
MetaType	Indicates the meta-type of the enumeration elements.
Single	Returns the only element in the enumeration.
ToList	Returns the enumeration as a TList instance.

See Also

- TObjectEnumerable<T> Class
- TObjectEnumerable<T> Class Members

6.20.1.1.3.1 Contains Method

Checks whether the enumeration contains the specified entity instance.

Class

TObjectEnumerable<T>

Syntax

```
function Contains(Value: T): boolean;
```

Parameters

Value

Instance of the entity to be checked.

Remarks

The function returns True if the enumeration contains the specified entity instance, False otherwise.

Request Support DAC Forum Provide Feedback

6.20.1.1.3.2 Count Method

Indicates the number of elements in the enumeration.

Class

TObjectEnumerable<T>

Syntax

```
function Count: Integer; virtual; abstract;
```

Remarks

The property indicates the number of elements contained in the enumeration.

Request Support DAC Forum Provide Feedback

6.20.1.1.3.3 First Method

Returns the first element of the enumeration.

Class

TObjectEnumerable<T>

Syntax

```
function First: T;
```

Remarks

The function returns the first element of the enumeration. If the enumeration does not contain elements, the exception is raised.

 Request Support DAC Forum Provide Feedback

6.20.1.1.3.4 Last Method

Returns the last element of the enumeration.

Class

TObjectEnumerable<T>

Syntax

```
function Last: T;
```

Remarks

The function returns the last element of the enumeration. If the enumeration does not contain elements, the exception is raised.

 Request Support DAC Forum Provide Feedback

6.20.1.1.3.5 MetaType Method

Indicates the meta-type of the enumeration elements.

Class

TObjectEnumerable<T>

Syntax

```
function MetaType: TMetaType; virtual; abstract;
```

Remarks

The function indicates the meta-type of elements contained in the enumeration.

Request Support DAC Forum Provide Feedback

6.20.1.1.3.6 Single Method

Returns the only element in the enumeration.

Class

TObjectEnumerable<T>

Syntax

```
function Single: T;
```

Remarks

The function returns the only element in the enumeration. If the enumeration does not contain elements, or it contains more than one element, the appropriate exception is raised.

Request Support DAC Forum Provide Feedback

6.20.1.1.3.7 ToList Method

Returns the enumeration as a TList instance.

Class

TObjectEnumerable<T>

Syntax

```
function ToList: TList<T>;
```

Remarks

The function returns the TList instance filled with the enumeration elements.

Request Support DAC Forum Provide Feedback

Reserved.

6.20.2 Interfaces

Interfaces in the **EntityDAC.Enumerable** unit.

Interfaces

Name	Description
IObjectEnumerable<T>	The base interface which declares functionality for entity enumerations.

Request Support DAC Forum Provide Feedback

6.20.2.1 IObjectEnumerable<T> Interface

The base interface which declares functionality for entity enumerations.

Unit

EntityDAC.Enumerable

Syntax

```
IObjectEnumerable<T: class> = interface(IEnumerable)[ '<0FDFB7AD-B121-4875-A9E0-93BE718583C9>' ];
```

Remarks

The IObjectEnumerable<T> interface declares properties and methods for implementing entity enumerations. All enumeration and collection classes used in EntityDAC, such as TEntityEnumerable or TEntityCollection implement IObjectEnumerable<T> interface.

The IObjectEnumerable<T> interface is IEnumerable descendant.

See Also

- TEntityEnumerable
- TEntityCollection

Request Support DAC Forum Provide Feedback

Reserved.

6.20.2.1.1 Members

IObjectEnumerable<T> class overview.

Properties

Name	Description
Elements	Provides indexed access to an enumeration elements.

Methods

Name	Description
Contains	Determines whether an enumeration contains a specified element.
Count	Returns the number of elements in an enumeration.
ElementAt	Returns the element at a specified index in an enumeration.
First	Returns the first element of an enumeration.
FirstOrDefault	Returns the first element of an enumeration, or a nil value if no element is found.
Last	Returns the last element of an enumeration.
LastOrDefault	Returns the last element of an enumeration, or a nil value if no element is found.
MetaType	Indicates the meta-type of elements in an enumeration.
Single	Returns the only element of an enumeration, and throws an exception if there is not exactly one element in the enumeration.
SingleOrDefault	Returns the only element of an enumeration, or a nil value if there is not exactly one element in the

	enumeration.
ToList	Returns the whole elements collection of an enumeration as TList.
Where	Overloaded.Description is not available at the moment.

Request Support DAC Forum Provide Feedback

6.20.2.1.2 Properties

Properties of the **IObjectEnumerable<T>** class.

For a complete list of the **IObjectEnumerable<T>** class members, see the IObjectEnumerable<T> Members topic.

Public

Name	Description
Elements	Provides indexed access to an enumeration elements.

See Also

- IObjectEnumerable<T> Interface
- IObjectEnumerable<T> Interface Members

Request Support DAC Forum Provide Feedback

6.20.2.1.2.1 Elements Property(Indexer)

Provides indexed access to an enumeration elements.

Class

IObjectEnumerable<T>

Syntax

```
property Elements[Index: Integer]: T; default;
```

Parameters

Index

The zero-based index of the element to access.

Remarks

Use the property to access an element in the enumeration by its index. If the specified index is less than zero or greater than or equal to the number of elements in the enumeration, the exception is raised.

Request Support DAC Forum Provide Feedback

6.20.2.1.3 Methods

Methods of the **IObjectEnumerable<T>** class.

For a complete list of the **IObjectEnumerable<T>** class members, see the IObjectEnumerable<T> Members topic.

Public

Name	Description
Contains	Determines whether an enumeration contains a specified element.
Count	Returns the number of elements in an enumeration.
ElementAt	Returns the element at a specified index in an enumeration.
First	Returns the first element of an enumeration.
FirstOrDefault	Returns the first element of an enumeration, or a nil value if no element is found.
Last	Returns the last element of an enumeration.
LastOrDefault	Returns the last element of an enumeration, or a nil value if no element is found.
MetaType	Indicates the meta-type of elements in an enumeration.

Single	Returns the only element of an enumeration, and throws an exception if there is not exactly one element in the enumeration.
SingleOrDefault	Returns the only element of an enumeration, or a nil value if there is not exactly one element in the enumeration.
ToList	Returns the whole elements collection of an enumeration as TList.
Where	Overloaded.Description is not available at the moment.

See Also

- IObjectEnumerable<T> Interface
- IObjectEnumerable<T> Interface Members

6.20.2.1.3.1 Contains Method

Determines whether an enumeration contains a specified element.

Class

IObjectEnumerable<T>

Syntax

```
function Contains(Value: T): boolean;
```

Parameters

Value

The value to locate in the enumeration.

Remarks

Use the method to determine whether an enumeration contains a specified element. The method returns True if the element exists in the enumeration, False otherwise.

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.2 Count Method

Returns the number of elements in an enumeration.

Class

IObjectEnumerable<T>

Syntax

```
function Count: Integer;
```

Remarks

Use the method to determine the number of elements in an enumeration.

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.3 ElementAt Method

Returns the element at a specified index in an enumeration.

Class

IObjectEnumerable<T>

Syntax

```
function ElementAt(Index: Integer): T;
```

Parameters

Index

The zero-based index of the element to retrieve.

Remarks

Use the method to obtain the element at a specified index in an enumeration. If the specified index is less than zero or greater than or equal to the number of elements in the enumeration, the exception is raised.

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.4 First Method

Returns the first element of an enumeration.

Class

IObjectEnumerable<T>

Syntax

```
function First: T;
```

Remarks

Use the method to obtain the first element of an enumeration. The method throws an exception if the enumeration contains no elements. To instead return nil when the enumeration is empty, use the FirstOrDefault method.

See Also

- FirstOrDefault

Request Support DAC Forum Provide Feedback

6.20.2.1.3.5 FirstOrDefault Method

Returns the first element of an enumeration, or a nil value if no element is found.

Class

IObjectEnumerable<T>

Syntax

```
function FirstOrDefault: T;
```

Remarks

Use the method to obtain the first element of an enumeration. If the enumeration contains no elements, the method returns nil.

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.6 Last Method

Returns the last element of an enumeration.

Class

IObjectEnumerable<T>

Syntax

```
function Last: T;
```

Remarks

Use the method to obtain the last element of an enumeration. The method throws an exception if the enumeration contains no elements. To instead return nil when the enumeration is empty, use the LastOrDefault method.

See Also

- LastOrDefault

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.7 LastOrDefault Method

Returns the last element of an enumeration, or a nil value if no element is found.

Class

IObjectEnumerable<T>

Syntax

```
function LastOrDefault: T;
```

Remarks

Use the method to obtain the last element of an enumeration. If the enumeration contains no elements, the method returns nil.

6.20.2.1.3.8 MetaType Method

Indicates the meta-type of elements in an enumeration.

Class

IObjectEnumerable<T>

Syntax

```
function MetaType: TMetaType;
```

Remarks

Use the method to find out the meta-type of elements in an enumeration.

6.20.2.1.3.9 Single Method

Returns the only element of an enumeration, and throws an exception if there is not exactly one element in the enumeration.

Class

IObjectEnumerable<T>

Syntax

```
function Single: T;
```

Remarks

Use the method to obtain the only element of an enumeration. If the enumeration contains more than one element or the enumeration is empty, the method throws an exception. To instead return nil when the enumeration is empty or contains more that one element, use the M:Devart.EntityDAC.IObjectEnumerable{T}.SingleOrDefault() method.

See Also

- SingleOrDefault

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.10 SingleOrDefault Method

Returns the only element of an enumeration, or a nil value if there is not exactly one element in the enumeration.

Class

IObjectEnumerable<T>

Syntax

```
function SingleOrDefault: T;
```

Remarks

Use the method to obtain the only element of an enumeration. If the enumeration contains more than one element or the enumeration is empty, the method returns nil.

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.11 ToList Method

Returns the whole elements collection of an enumeration as TList.

Class

IObjectEnumerable<T>

Syntax

```
function ToList: TList<T>;
```

Remarks

Use the method to obtain the whole elements collection of an enumeration as TList.

 Request Support DAC Forum Provide Feedback

6.20.2.1.3.12 Where Method

Class

IObjectEnumerable<T>

Overload List

Name	Description
Where(**const** Filter: ICompiledExpressionStatement)	Filters an enumeration based on a specified condition.
Where(**const** Filter: TExpression)	Filters an enumeration based on a specified condition.
Where(**const** Filter: **string**)	Filters an enumeration based on a specified condition.

Filters an enumeration based on a specified condition.

Class

IObjectEnumerable<T>

Syntax

```
function Where(const Filter: ICompiledExpressionStatement): IObjectEnumerable<T>; overload;
```

Parameters

Filter

A condition to which the result elements must satisfy.

Return Value

An IObjectEnumerable that contains elements from the input enumeration that satisfy the condition.

Remarks

Use the method to obtain the filtered set of an enumeration elements based on a specified condition. The condition has to be specified as a precompiled expression.

See Also

- ICompiledExpressionStatement

 Request Support DAC Forum Provide Feedback

Filters an enumeration based on a specified condition.

Class

IObjectEnumerable<T>

Syntax

```
function Where(const Filter: TExpression): IObjectEnumerable<T>;
overload;
```

Parameters

Filter

A condition to which the result elements must satisfy.

Return Value

An IObjectEnumerable that contains elements from the input enumeration that satisfy the condition.

Remarks

Use the method to obtain the filtered set of an enumeration elements based on a specified condition. The condition has to be specified using the query syntax described in the A:linq_query_syntax.htm article.

 Request Support DAC Forum Provide Feedback

Filters an enumeration based on a specified condition.

Class

IObjectEnumerable<T>

Syntax

```
function Where(const Filter: string): IObjectEnumerable<T>;
overload;
```

Parameters

Filter

A condition to which the result elements must satisfy.

Return Value

An IObjectEnumerable that contains elements from the input enumeration that satisfy the condition.

Remarks

Use the method to obtain the filtered set of an enumeration elements based on a specified condition. The condition has to be specified as a string using the syntax described in the A:specify_LINQ_query_arguments_as_string.htm article.

Request Support DAC Forum Provide Feedback

6.21 EntityDAC.MetaData

The unit contains implementation of the metadata management.

Classes

Name	Description
TMappedMetaType	A meta-type for Entities, that can be saved to the database.
TMetaColumnList	A list of all the columns in the table.
TMetaColumns	A custom list of meta columns.
TMetaData	TMetaData is a base class, from which any metadata class is derived.
TMetaModel	The class stores meta-descriptions of all the entity classes of the model and relationships between them.
TMetaReference	Contains the meta description of the reference.
TMetaTable	This class contains metainformation about the table.

TMetaTableList	A list of all the tables in the model.
TMetaType	Meta-description of all the Entities used in the ORM.
TUnmappedMetaTable	Description of a virtual table, that doesn't actually exist in the database.

Request Support DAC Forum Provide Feedback

6.21.1 Classes

Classes in the **EntityDAC.MetaData** unit.

Classes

Name	Description
TMappedMetaType	A meta-type for Entities, that can be saved to the database.
TMetaColumnList	A list of all the columns in the table.
TMetaColumns	A custom list of meta columns.
TMetaData	TMetaData is a base class, from which any metadata class is derived.
TMetaModel	The class stores meta-descriptions of all the entity classes of the model and relationships between them.
TMetaReference	Contains the meta description of the reference.
TMetaTable	This class contains metainformation about the table.
TMetaTableList	A list of all the tables in the model.
TMetaType	Meta-description of all the Entities used in the ORM.
TUnmappedMetaTable	Description of a virtual table, that doesn't actually exist in

	the database.

Request Support DAC Forum Provide Feedback

6.21.1.1 TMappedMetaType Class

A meta-type for Entities, that can be saved to the database.

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

Unit

EntityDAC.MetaData

Syntax

```
TMappedMetaType = class(TMetaType);
```

Inheritance Hierarchy

TMetaData
  TMetaType
    TMappedMetaType

Request Support DAC Forum Provide Feedback

6.21.1.1.1 Members

TMappedMetaType class overview.

Properties

Name	Description
AllowCaching (inherited from **TMetaType**)	Allows to enable or disable entity caching.
ComplexMetaAttributes (inherited from **TMetaType**)	A list of all the complex attributes in the meta-type.
EntityClass	Class in Delphi, instances of which will be created for every new Entity with this meta-type.

Index	Used for fast access by index.
Inheritance	Describes inheritance if the given meta-type has it.
KeyGenerators	Value generators for key.
MetaAttributes (inherited from **TMetaType**)	A list of all the attributes in the meta-type.
MetaCollections (inherited from **TMetaType**)	A list of all the collections in the meta-type.
MetaKey	Primary key or any other key, that allows to uniquely identify an Entity.
MetaReferences (inherited from **TMetaType**)	A list of all the references in the meta-type.
MetaTable (inherited from **TMetaType**)	Specifies the table in the database, which the meta-type corresponds to.
Model (inherited from **TMetaType**)	Specifies the meta-model, which the meta-type belongs to.
Name (inherited from **TMetaData**)	Specifies the name of metadata.

Request Support DAC Forum Provide Feedback

6.21.1.1.2 Properties

Properties of the **TMappedMetaType** class.

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

Public

Name	Description
AllowCaching (inherited from **TMetaType**)	Allows to enable or disable entity caching.
ComplexMetaAttributes (inherited from **TMetaType**)	A list of all the complex attributes in the meta-type.
EntityClass	Class in Delphi, instances of which will be created for every new Entity with this

	meta-type.
Index	Used for fast access by index.
Inheritance	Describes inheritance if the given meta-type has it.
KeyGenerators	Value generators for key.
MetaAttributes (inherited from **TMetaType**)	A list of all the attributes in the meta-type.
MetaCollections (inherited from **TMetaType**)	A list of all the collections in the meta-type.
MetaKey	Primary key or any other key, that allows to uniquely identify an Entity.
MetaReferences (inherited from **TMetaType**)	A list of all the references in the meta-type.
MetaTable (inherited from **TMetaType**)	Specifies the table in the database, which the meta-type corresponds to.
Model (inherited from **TMetaType**)	Specifies the meta-model, which the meta-type belongs to.
Name (inherited from **TMetaData**)	Specifies the name of metadata.

See Also

- TMappedMetaType Class
- TMappedMetaType Class Members

6.21.1.1.2.1 EntityClass Property

Class in Delphi, instances of which will be created for every new Entity with this meta-type.

Class

TMappedMetaType

Syntax

```
property EntityClass: TClass;
```

 Request Support DAC Forum Provide Feedback

6.21.1.1.2.2 Index Property

Used for fast access by index.

Class

TMappedMetaType

Syntax

```
property Index: Integer;
```

 Request Support DAC Forum Provide Feedback

6.21.1.1.2.3 Inheritance Property

Describes inheritance if the given meta-type has it.

Class

TMappedMetaType

Syntax

```
property Inheritance: TMetaTypeInheritance;
```

 Request Support DAC Forum Provide Feedback

6.21.1.1.2.4 KeyGenerators Property

Value generators for key.

Class

TMappedMetaType

Syntax

```
property KeyGenerators: TMetaKeyGeneratorList;
```


6.21.1.1.2.5 MetaKey Property

Primary key or any other key, that allows to uniquely identify an Entity.

Class

TMappedMetaType

Syntax

```
property MetaKey: TMetaKey;
```


6.21.1.2 TMetaColumnList Class

A list of all the columns in the table.

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

Unit

EntityDAC.MetaData

Syntax

```
TMetaColumnList = class(TMetaList);
```

Inheritance Hierarchy

TMetaList

TMetaColumnList

6.21.1.2.1 Members

TMetaColumnList class overview.

Methods

Name	Description
Find	Searches for columns by name. If cannot find, returns nil.
Get	Searches for columns by name. If cannot find - raises an exception.

 Request Support DAC Forum Provide Feedback

6.21.1.2.2 Methods

Methods of the **TMetaColumnList** class.

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

Public

Name	Description
Find	Searches for columns by name. If cannot find, returns nil.
Get	Searches for columns by name. If cannot find - raises an exception.

See Also

- TMetaColumnList Class
- TMetaColumnList Class Members

 Request Support DAC Forum Provide Feedback

6.21.1.2.2.1 Find Method

Searches for columns by name. If cannot find, returns nil.

Class

TMetaColumnList

Syntax

```
function Find(const Name: string): TMetaColumn;
```

Parameters

Name

 Request Support DAC Forum Provide Feedback

6.21.1.2.2.2 Get Method

Searches for columns by name. If cannot find - raises an exception.

Class

TMetaColumnList

Syntax

```
function Get(const Name: string): TMetaColumn;
```

Parameters

Name

 Request Support DAC Forum Provide Feedback

6.21.1.3 TMetaColumns Class

A custom list of meta columns.

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

Unit

EntityDAC.MetaData

Syntax

```
TMetaColumns = class(TMetaColumnList);
```

Inheritance Hierarchy

TMetaList

TMetaColumnList

TMetaColumns

 Request Support DAC Forum Provide Feedback

6.21.1.3.1 Members

TMetaColumns class overview.

Methods

Name	Description
Add	Adds a meta-column
Find (inherited from **TMetaColumnList**)	Searches for columns by name. If cannot find, returns nil.
Get (inherited from **TMetaColumnList**)	Searches for columns by name. If cannot find - raises an exception.
Remove	Removes a meta-column.

 Request Support DAC Forum Provide Feedback

6.21.1.3.2 Methods

Methods of the **TMetaColumns** class.

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

Public

Name	Description
Add	Adds a meta-column
Find (inherited from **TMetaColumnList**)	Searches for columns by name. If cannot find, returns nil.
Get (inherited from **TMetaColumnList**)	Searches for columns by name. If cannot find - raises an exception.
Remove	Removes a meta-column.

See Also

- TMetaColumns Class
- TMetaColumns Class Members

6.21.1.3.2.1 Add Method

Adds a meta-column

Class

TMetaColumns

Syntax

```
procedure Add(Item: TMetaColumn);
```

Parameters

Item

6.21.1.3.2.2 Remove Method

Removes a meta-column.

Class

TMetaColumns

Syntax

```
procedure Remove(Item: TMetaColumn);
```

Parameters

Item

 Request Support DAC Forum Provide Feedback

6.21.1.4 TMetaData Class

TMetaData is a base class, from which any metadata class is derived.

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

Unit

EntityDAC.MetaData

Syntax

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

 Request Support DAC Forum Provide Feedback

6.21.1.4.1 Members

TMetaData class overview.

Properties

Name	Description
Name	Specifies the name of metadata.

 Request Support DAC Forum Provide Feedback

6.21.1.4.2 Properties

Properties of the **TMetaData** class.

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

Public

Name	Description
Name	Specifies the name of metadata.

See Also

- TMetaData Class
- TMetaData Class Members

 Request Support DAC Forum Provide Feedback

6.21.1.4.2.1 Name Property

Specifies the name of metadata.

Class

TMetaData

Syntax

```
property Name: string;
```

 Request Support DAC Forum Provide Feedback

6.21.1.5 TMetaModel Class

The class stores meta-descriptions of all the entity classes of the model and relationships between them.

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

Unit

EntityDAC.MetaData

Syntax

```
TMetaModel = class(TMetaData);
```

Inheritance Hierarchy

TMetaData

TMetaModel

 Request Support DAC Forum Provide Feedback

6.21.1.5.1 Members

TMetaModel class overview.

Properties

Name	Description
Index	The property is designed to indicate the index of the meta model in the global meta model list.
MetaAssociations	The property is designed to store a collection of meta-type relations.
MetaTables	The property is designed to store a collection of meta-tables.
MetaType	The property is designed to return a mapped meta-type by its name.
MetaTypes	The property is designed to store a collection of mapped meta-types.
Name (inherited from **TMetaData**)	Specifies the name of metadata.
UnmappedMetaTypes	The property is designed to store a collection of un-mapped meta-types.

 Request Support DAC Forum Provide Feedback

6.21.1.5.2 Properties

Properties of the **TMetaModel** class.

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

Public

Name	Description
Index	The property is designed to indicate the index of the meta model in the global meta model list.
MetaAssociations	The property is designed to store a collection of meta-type relations.
MetaTables	The property is designed to store a collection of meta-tables.
MetaType	The property is designed to return a mapped meta-type by its name.
MetaTypes	The property is designed to store a collection of mapped meta-types.
Name (inherited from **TMetaData**)	Specifies the name of metadata.
UnmappedMetaTypes	The property is designed to store a collection of un-mapped meta-types.

See Also

- TMetaModel Class
- TMetaModel Class Members

6.21.1.5.2.1 Index Property

The property is designed to indicate the index of the meta model in the global meta model list.

Class

TMetaModel

Syntax

```
property Index: Integer;
```

Remarks

The property indicates the index of the meta-model in the global meta-model list. It can be used as an alternative to accessing the model by its name.

 Request Support DAC Forum Provide Feedback

6.21.1.5.2.2 MetaAssociations Property

The property is designed to store a collection of meta-type relations.

Class

TMetaModel

Syntax

```
property MetaAssociations: TMetaAssociationList;
```

Remarks

The property stores a collection of meta-type relations, such as one-to-one, one-to-many and many-to-many associations.

 Request Support DAC Forum Provide Feedback

6.21.1.5.2.3 MetaTables Property

The property is designed to store a collection of meta-tables.

Class

TMetaModel

Syntax

```
property MetaTables: TMetaTableList;
```

Remarks

The property stores a collection of meta-tables. Each meta-table describes a database table where corresponding mapped meta-types are stored.

Request Support DAC Forum Provide Feedback

6.21.1.5.2.4 MetaType Property(Indexer)

The property is designed to return a mapped meta-type by its name.

Class

TMetaModel

Syntax

```
property MetaType[Name: string]: TMappedMetaType; default;
```

Parameters

Name

The unique name of the mapped meta-type to return.

Remarks

The property returns a particular meta-type by its unique name. The property can return a mapped meta-type only, because unmapped metatypes have auto-generated names and can not be accessed in this way.

Request Support DAC Forum Provide Feedback

6.21.1.5.2.5 MetaTypes Property

The property is designed to store a collection of mapped meta-types.

Class

TMetaModel

Syntax

```
property MetaTypes: TMappedMetaTypeList;
```

Remarks

The property stores a collection of mapped meta-types. Each mapped meta-type describes an entity class in the model, which is mapped to a database table (can read and write its properties to the table).

Request Support DAC Forum Provide Feedback

6.21.1.5.2.6 UnmappedMetaTypes Property

The property is designed to store a collection of un-mapped meta-types.

Class

TMetaModel

Syntax

```
property UnmappedMetaTypes: TUnmappedMetaTypeList;
```

Remarks

The property stores a collection of un-mapped meta-types. Unmapped meta-type means that the corresponding entity class is not mapped to a certain database table (for example, an entity that was returned as a result of a LINQ query).

Request Support DAC Forum Provide Feedback

6.21.1.5.3 Methods

Methods of the **TMetaModel** class.

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

See Also

- TMetaModel Class
- TMetaModel Class Members

Request Support DAC Forum Provide Feedback

6.21.1.5.3.1 Create Constructor

The constructor is designed to create a new instance of the class.

Class

TMetaModel

Syntax

```
constructor Create(Name: string);
```

Remarks

The constructor creates a new instance of the class. Since an application can use several models, each model description has to be stored in a separate TMetaModel class instance. All the instances are stored in a global meta-model list, and a particular one can be accessed via its name. Therefore, the name of the model has to be unique. An attempt to create a meta-model with an existing name will cause an exception.

Request Support DAC Forum Provide Feedback

6.21.1.6 TMetaReference Class

Contains the meta description of the reference.

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

Unit

EntityDAC.MetaData

Syntax

```
TMetaReference = class(TMetaLink);
```

Inheritance Hierarchy

TMetaData

TMetaDataMember

TMetaLink

TMetaReference

 Request Support DAC Forum Provide Feedback

6.21.1.6.1 Members

TMetaReference class overview.

 Request Support DAC Forum Provide Feedback

6.21.1.7 TMetaTable Class

This class contains metainformation about the table.

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

Unit

EntityDAC.MetaData

Syntax

```
TMetaTable = class(TMetaData);
```

Inheritance Hierarchy

TMetaData

TMetaTable

 Request Support DAC Forum Provide Feedback

6.21.1.7.1 Members

TMetaTable class overview.

Properties

Name	Description
Index	Allows to get fast access by serial number.
MetaColumns	A list of columns in the table.
Model	Specifies the model, which the given table belongs to.
Name (inherited from **TMetaData**)	Specifies the name of metadata.

 Request Support DAC Forum Provide Feedback

6.21.1.7.2 Properties

Properties of the **TMetaTable** class.

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

Public

Name	Description
Index	Allows to get fast access by serial number.
MetaColumns	A list of columns in the table.
Model	Specifies the model, which the given table belongs to.
Name (inherited from **TMetaData**)	Specifies the name of metadata.

See Also

- TMetaTable Class

- TMetaTable Class Members

 Request Support DAC Forum Provide Feedback

6.21.1.7.2.1 Index Property

Allows to get fast access by serial number.

Class

TMetaTable

Syntax

```
property Index: Integer;
```

 Request Support DAC Forum Provide Feedback

6.21.1.7.2.2 MetaColumns Property

A list of columns in the table.

Class

TMetaTable

Syntax

```
property MetaColumns: TMetaColumnList;
```

 Request Support DAC Forum Provide Feedback

6.21.1.7.2.3 Model Property

Specifies the model, which the given table belongs to.

Class

TMetaTable

Syntax

```
property Model: TMetaModel;
```


Request Support DAC Forum Provide Feedback

6.21.1.8 TMetaTableList Class

A list of all the tables in the model.

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

Unit

EntityDAC.MetaData

Syntax

```
TMetaTableList = class(TMetaList);
```

Inheritance Hierarchy

TMetaList

TMetaTableList

Request Support DAC Forum Provide Feedback

6.21.1.8.1 Members

TMetaTableList class overview.

Properties

Name	Description
Items	Lists the meta tables.

Request Support DAC Forum Provide Feedback

6.21.1.8.2 Properties

Properties of the **TMetaTableList** class.

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

Public

Name	Description
Items	Lists the meta tables.

See Also

- TMetaTableList Class
- TMetaTableList Class Members

 Request Support DAC Forum Provide Feedback

6.21.1.8.2.1 Items Property(Indexer)

Lists the meta tables.

Class

TMetaTableList

Syntax

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

Parameters

Index

 Request Support DAC Forum Provide Feedback

6.21.1.9 TMetaType Class

Meta-description of all the Entities used in the ORM.

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

Unit

EntityDAC.MetaData

Syntax

```
TMetaType = class(TMetaData);
```

Inheritance Hierarchy

TMetaData

TMetaType

Request Support DAC Forum Provide Feedback

6.21.1.9.1 Members

TMetaType class overview.

Properties

Name	Description
AllowCaching	Allows to enable or disable entity caching.
ComplexMetaAttributes	A list of all the complex attributes in the meta-type.
MetaAttributes	A list of all the attributes in the meta-type.
MetaCollections	A list of all the collections in the meta-type.
MetaReferences	A list of all the references in the meta-type.
MetaTable	Specifies the table in the database, which the meta-type corresponds to.
Model	Specifies the meta-model, which the meta-type belongs to.
Name (inherited from **TMetaData**)	Specifies the name of metadata.

Request Support DAC Forum Provide Feedback

6.21.1.9.2 Properties

Properties of the **TMetaType** class.

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

Public

Name	Description
AllowCaching	Allows to enable or disable entity caching.
ComplexMetaAttributes	A list of all the complex attributes in the meta-type.
MetaAttributes	A list of all the attributes in the meta-type.
MetaCollections	A list of all the collections in the meta-type.
MetaReferences	A list of all the references in the meta-type.
MetaTable	Specifies the table in the database, which the meta-type corresponds to.
Model	Specifies the meta-model, which the meta-type belongs to.
Name (inherited from **TMetaData**)	Specifies the name of metadata.

See Also

- TMetaType Class
- TMetaType Class Members

6.21.1.9.2.1 Allow Caching Property

Allows to enable or disable entity caching.

Class

`TMetaType`

Syntax

```
property AllowCaching: boolean;
```

Remarks

If Enabled is set to True, then caching of entities having this meta-type is enabled, and on attempt to retrieve an entity (entity list) for the second time using GetEntity or GetEntities methods, the corresponding entities will be taken from the cache.

If set to False, then caching of entities having this meta-type is disabled, and using GetEntity or GetEntities methods will return entities from the database every time.

True by default.

Request Support DAC Forum Provide Feedback

6.21.1.9.2.2 ComplexMetaAttributes Property

A list of all the complex attributes in the meta-type.

Class

TMetaType

Syntax

```
property ComplexMetaAttributes: TComplexMetaAttributeList;
```


Request Support DAC Forum Provide Feedback

6.21.1.9.2.3 MetaAttributes Property

A list of all the attributes in the meta-type.

Class

TMetaType

Syntax

```
property MetaAttributes: TMetaAttributeList;
```

 Request Support DAC Forum Provide Feedback

6.21.1.9.2.4 MetaCollections Property

A list of all the collections in the meta-type.

Class

TMetaType

Syntax

```
property MetaCollections: TMetaCollectionList;
```

 Request Support DAC Forum Provide Feedback

6.21.1.9.2.5 MetaReferences Property

A list of all the references in the meta-type.

Class

TMetaType

Syntax

```
property MetaReferences: TMetaReferenceList;
```

 Request Support DAC Forum Provide Feedback

6.21.1.9.2.6 MetaTable Property

Specifies the table in the database, which the meta-type corresponds to.

Class

TMetaType

Syntax

```
property MetaTable: TMetaTable;
```


6.21.1.9.2.7 Model Property

Specifies the meta-model, which the meta-type belongs to.

Class

TMetaType

Syntax

```
property Model: TMetaModel;
```


6.21.1.10 TUnmappedMetaTable Class

Description of a virtual table, that doesn't actually exist in the database.

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

Unit

EntityDAC.MetaData

Syntax

```
TUnmappedMetaTable = class(TMetaTable);
```

Inheritance Hierarchy

TMetaData

TMetaTable

TUnmappedMetaTable

6.21.1.10.1 Members

TUnmappedMetaTable class overview.

Properties

Name	Description
Index (inherited from **TMetaTable**)	Allows to get fast access by serial number.
MetaColumns (inherited from **TMetaTable**)	A list of columns in the table.
Model (inherited from **TMetaTable**)	Specifies the model, which the given table belongs to.
Name (inherited from **TMetaData**)	Specifies the name of metadata.

Request Support DAC Forum Provide Feedback

6.22 EntityDAC.NullableTypes

This unit contains implementation of nullable types management.

Structs

Name	Description
AnsiStringNullable	Nullable AnsiString type
BooleanNullable	Nullable Boolean type
ByteNullable	Nullable Byte type
CurrencyNullable	Nullable Currency type
DoubleNullable	Nullable Double type
ExtendedNullable	Nullable Extended type
Int64Nullable	Nullable Int64 type
IntegerNullable	Nullable Integer type
LongWordNullable	Nullable LongWord type
ShortIntNullable	Nullable ShortInt type
SingleNullable	Nullable Single type
SmallIntNullable	Nullable SmallInt type
TBcdNullable	Nullable TBcd type

TBytesNullable	Nullable TBytes type
TDateNullable	Nullable TDate type
TDateTimeNullable	Nullable TDateTime type
TGUIDNullable	Nullable TGUID type
TSQLTimeStampNullable	Nullable TSQLTimeStamp type
TTimeNullable	Nullable TTime type
WideStringNullable	Nullable WideString type
WordNullable	Nullable Word type

Request Support DAC Forum Provide Feedback

6.22.1 Structs

Structs in the **EntityDAC.NullableTypes** unit.

Structs

Name	Description
AnsiStringNullable	Nullable AnsiString type
BooleanNullable	Nullable Boolean type
ByteNullable	Nullable Byte type
CurrencyNullable	Nullable Currency type
DoubleNullable	Nullable Double type
ExtendedNullable	Nullable Extended type
Int64Nullable	Nullable Int64 type
IntegerNullable	Nullable Integer type
LongWordNullable	Nullable LongWord type
ShortIntNullable	Nullable ShortInt type
SingleNullable	Nullable Single type
SmallIntNullable	Nullable SmallInt type

TBcdNullable	Nullable TBcd type
TBytesNullable	Nullable TBytes type
TDateNullable	Nullable TDate type
TDateTimeNullable	Nullable TDateTime type
TGUIDNullable	Nullable TGUID type
TSQLTimeStampNullable	Nullable TSQLTimeStamp type
TTimeNullable	Nullable TTime type
WideStringNullable	Nullable WideString type
WordNullable	Nullable Word type

Request Support DAC Forum Provide Feedback

6.22.1.1 AnsiStringNullable Record

Nullable AnsiString type

Unit

EntityDAC.NullableTypes

Syntax

```
AnsiStringNullable = record;
```

Fields

Clear

Sets the IsNull property to True.

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.2 BooleanNullable Record

Nullable Boolean type

Unit

EntityDAC.NullableTypes

Syntax

```
BooleanNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.3 ByteNullable Record

Nullable Byte type

Unit

EntityDAC.NullableTypes

Syntax

```
ByteNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.4 CurrencyNullable Record

Nullable Currency type

Unit

EntityDAC.NullableTypes

Syntax

```
CurrencyNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.5 DoubleNullable Record

Nullable Double type

Unit

EntityDAC.NullableTypes

Syntax

```
DoubleNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.6 ExtendedNullable Record

Nullable Extended type

Unit

EntityDAC.NullableTypes

Syntax

```
ExtendedNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.7 Int64Nullable Record

Nullable Int64 type

Unit

EntityDAC.NullableTypes

Syntax

```
Int64Nullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.8 IntegerNullable Record

Nullable Integer type

Unit

EntityDAC.NullableTypes

Syntax

```
IntegerNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.9 LongWordNullable Record

Nullable LongWord type

Unit

EntityDAC.NullableTypes

Syntax

```
LongWordNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.10 ShortIntNullable Record

Nullable ShortInt type

Unit

EntityDAC.NullableTypes

Syntax

```
ShortIntNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.11 SingleNullable Record

Nullable Single type

Unit

EntityDAC.NullableTypes

Syntax

```
SingleNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.12 SmallIntNullable Record

Nullable SmallInt type

Unit

EntityDAC.NullableTypes

Syntax

```
SmallIntNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.13 TBcdNullable Record

Nullable TBcd type

Unit

EntityDAC.NullableTypes

Syntax

```
TBcdNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

6.22.1.14 TBytesNullable Record

Nullable TBytes type

Unit

EntityDAC.NullableTypes

Syntax

```
TBytesNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.15 TDateNullable Record

Nullable TDate type

Unit

EntityDAC.NullableTypes

Syntax

```
TDateNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.16 TDateTimeNullable Record

Nullable TDateTime type

Unit

EntityDAC.NullableTypes

Syntax

```
TDateTimeNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.17 TGUIDNullable Record

Nullable TGUID type

Unit

EntityDAC.NullableTypes

Syntax

```
TGUIDNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

6.22.1.18 TSQLTimeStampNullable Record

Nullable TSQLTimeStamp type

Unit

EntityDAC.NullableTypes

Syntax

```
TSQLTimeStampNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.19 TTimeNullable Record

Nullable TTime type

Unit

EntityDAC.NullableTypes

Syntax

```
TTimeNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

Request Support DAC Forum Provide Feedback

6.22.1.20 WideStringNullable Record

Nullable WideString type

Unit

EntityDAC.NullableTypes

Syntax

```
WideStringNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

6.22.1.21 WordNullable Record

Nullable Word type

Unit

EntityDAC.NullableTypes

Syntax

```
WordNullable = record;
```

Fields

Clear

Sets the IsNull property to True

HasValue

The property returns False if the value is Null; otherwise, the property returns True.

IsNull

The property returns True if the value is Null; otherwise, the property returns False.

ToString

The method converts a value to string representation, so that it is suitable for display.

Value

The property returns a value if it is not Null.

 Request Support DAC Forum Provide Feedback

6.23 EntityDAC.ObjectContext

The unit contains implementation of the data context functionality.

Classes

Name	Description
TCustomObjectContext	A basic class that provides the data context functionality.
TObjectContext	The class provides the data context functionality for handling entities which are not TEntity descendants.

 Request Support DAC Forum Provide Feedback

6.23.1 Classes

Classes in the **EntityDAC.ObjectContext** unit.

Classes

Name	Description

TCustomObjectContext	A basic class that provides the data context functionality.
TObjectContext	The class provides the data context functionality for handling entities which are not TEntity descendants.

Request Support DAC Forum Provide Feedback

6.23.1.1 TCustomObjectContext Class

A basic class that provides the data context functionality.

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

Unit

EntityDAC.ObjectContext

Syntax

```
TCustomObjectContext = class(TDataContext);
```

Remarks

The TCustomObjectContext class provides functionality for managing an entity life cycle in the application. It provides methods for creating and initializing new entity instances, retrieving and storing entities from/to the database, storing used entities in the cache for future use, destroying of unused entities.

The TCustomObjectContext functionality is identical to the TCustomEntityContext class. The main difference is that TCustomEntityContext is designed to manage entities which are the TEntity class descendants, and it is used when working with code-mapped entities, Attribute-mapped entities or XML-mapped entities. TCustomObjectContext is designed to manage entities which are the TObject descendants, and it is used when working with A:attribute-mapped-objects.

Since TCustomObjectContext is a basic class, it should not be used directly. Instead, TCustomObjectContext descendants such as TObjectContext have to be used.

Inheritance Hierarchy

TCustomContext

TDataContext

TCustomObjectContext

See Also

- TCustomEntityContext
- TObjectContext
- Model Mapping

6.23.1.1.1 Members

TCustomObjectContext class overview.

Properties

Name	Description
Connection (inherited from **TCustomContext**)	Identifies the connection component with which the data context is associated.
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
ModelName (inherited from **TCustomContext**)	Specifies the name of the meta model used by the data context.
Types (inherited from **TDataContext**)	The property is designed to determine a meta-type by a meta-type name.

Methods

Name	Description
Create (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.

ExecuteQuery<T> (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
ExecuteSQL (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
RejectChanges (inherited from **TDataContext**)	The method is designed to cancel changes in all attached entities
SubmitChanges (inherited from **TDataContext**)	The method is designed for saving changes in all attached entities.

Events

Name	Description
OnGetGeneratorValue (inherited from **TCustomContext**)	Occurs when an entity attribute value generator of type "custom" needs to generate its value.

6.23.1.2 TObjectContext Class

The class provides the data context functionality for handling entities which are not TEntity descendants.

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

Unit

EntityDAC.ObjectContext

Syntax

```
TObjectContext = class(TCustomObjectContext);
```

Remarks

TObjectContext class provides functionality for handling entities which are not TEntity descendants.

For operating entities which are TEntity descendants, TEntityContext class is used.

Inheritance Hierarchy

TCustomContext
  TDataContext
    TCustomObjectContext
      TObjectContext

See Also

- TEntity
- TEntityContext

6.23.1.2.1 Members

TObjectContext class overview.

Properties

Name	Description
Connection	Identifies the connection component with which the data context is associated.
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
ModelName	Specifies the name of the meta model used by the data context.
Options	The property allows setting up the behavior of the TObjectContext class.
Types (inherited from **TDataContext**)	The property is designed to determine a meta-type by a meta-type name.

Methods

Name	Description
Create (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
ExecuteQuery<T> (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
ExecuteSQL (inherited from **TCustomContext**)	Overloaded.Description is not available at the moment.
RejectChanges (inherited from **TDataContext**)	The method is designed to cancel changes in all attached entities
SubmitChanges (inherited from **TDataContext**)	The method is designed for saving changes in all attached entities.

Events

Name	Description
OnGetGeneratorValue (inherited from **TCustomContext**)	Occurs when an entity attribute value generator of type "custom" needs to generate its value.

6.23.1.2.2 Properties

Properties of the **TObjectContext** class.

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

Public

Name	Description
Dialect (inherited from **TCustomContext**)	Indicates the current SQL dialect used by the connection data provider.
Model (inherited from **TCustomContext**)	Specifies the meta model used by the data context.
Types (inherited from **TDataContext**)	The property is designed to determine a meta-type by a meta-type name.

Published

Name	Description
Connection	Identifies the connection component with which the data context is associated.
ModelName	Specifies the name of the meta model used by the data context.
Options	The property allows setting up the behavior of the TObjectContext class.

See Also

- TObjectContext Class
- TObjectContext Class Members

Request Support DAC Forum Provide Feedback

6.23.1.2.2.1 Connection Property

Identifies the connection component with which the data context is associated.

Class

TObjectContext

Syntax

```
property Connection: TEntityConnection;
```

Remarks

Use the property to access properties, events and methods of the connection associated with the data context. Set the property to associate the data context with the TEntityConnection component.

See Also

- TEntityConnection

6.23.1.2.2.2 ModelName Property

Specifies the name of the meta model used by the data context.

Class

TObjectContext

Syntax

```
property ModelName: string;
```

Remarks

The property specifies the name of the meta model which is used by the data context.

Read the ModelName property to determine the name of used meta model. Set the name of the property to specify the used meta model. When the valid model name is set, the corresponding meta model instance can be accessed through the TCustomContext.Model property. Unless the meta model name is not specified using ModelName, the default connection meta model specified by the TEntityConnection.DefaultModelName property is used.

See Also

- TCustomContext.Model
- TEntityConnection.DefaultModelName

6.23.1.2.2.3 Options Property

The property allows setting up the behavior of the TObjectContext class.

Class

TObjectContext

Syntax

```
property Options: TContextOptions;
```

See Also

- TContextOptions

Request Support DAC Forum Provide Feedback

6.24 EntityDAC.SQLDialect

The base unit that contains information about SQL implementations for various DBMS.

Classes

Name	Description
TSQLStatement	Encapsulates the SQL statement text and its parameters.

Interfaces

Name	Description
ICompiledExpressionStatement	The interface which declares functionality for compiled expression statements.
ICompiledLinqStatement	The base interface which declares functionality for compiled LINQ statements.

Request Support DAC Forum Provide Feedback

6.24.1 Classes

Classes in the **EntityDAC.SQLDialect** unit.

Classes

Name	Description

TSQLStatement	Encapsulates the SQL statement text and its parameters.

 Request Support DAC Forum Provide Feedback

6.24.1.1 TSQLStatement Class

Encapsulates the SQL statement text and its parameters.

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

Unit

EntityDAC.SQLDialect

Syntax

```
TSQLStatement = class(TCustomStatement, ISQLStatement);
```

Inheritance Hierarchy

TCustomStatement

TSQLStatement

 Request Support DAC Forum Provide Feedback

6.24.1.1.1 Members

TSQLStatement class overview.

 Request Support DAC Forum Provide Feedback

6.24.2 Interfaces

Interfaces in the **EntityDAC.SQLDialect** unit.

Interfaces

Name	Description

ICompiledExpressionStatement	The interface which declares functionality for compiled expression statements.
ICompiledLinqStatement	The base interface which declares functionality for compiled LINQ statements.

 Request Support DAC Forum Provide Feedback

6.24.2.1 ICompiledExpressionStatement Interface

The interface which declares functionality for compiled expression statements.

Unit

EntityDAC.SQLDialect

Syntax

```
ICompiledExpressionStatement = interface(ICompiledLinqStatement)
[ '<0B25F4CF-BDD4-4BCA-90D2-C36275410CEE>' ];
```

Remarks

The ICompiledExpressionStatement interface declares properties and methods for implementing compiled expression statements. ICompiledExpressionStatement is a ICompiledLinqStatement descendant.

See Also

- ICompiledLinqStatement

 Request Support DAC Forum Provide Feedback

6.24.2.1.1 Members

ICompiledExpressionStatement class overview.

 Request Support DAC Forum Provide Feedback

6.24.2.2 ICompiledLinqStatement Interface

The base interface which declares functionality for compiled LINQ statements.

Unit

EntityDAC.SQLDialect

Syntax

```
ICompiledLinqStatement = interface(ICustomStatement)[ '<0268F917-EE9C-476A-958D-BFDC0E85801C>' ];
```

Remarks

The base interface which declares functionality for compiled LINQ statements.

Request Support DAC Forum Provide Feedback

6.24.2.2.1 Members

ICompiledLinqStatement class overview.

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

6.25 EntityDAC.Values

The unit contains implementation of classes that allow storing of any data.

Classes

Name	Description
TEDValue	A basic class that allows to store any data.
TEDValues	List of TEDValue.

Types

Name	Description
TEDValueClass	Class implementing the TEDValue basic class

Request Support DAC Forum Provide Feedback

6.25.1 Classes

Classes in the **EntityDAC.Values** unit.

Classes

Name	Description
TEDValue	A basic class that allows to store any data.
TEDValues	List of TEDValue.

Request Support DAC Forum Provide Feedback

6.25.1.1 TEDValue Class

A basic class that allows to store any data.

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

Unit

EntityDAC.Values

Syntax

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


Request Support DAC Forum Provide Feedback

6.25.1.1.1 Members

TEDValue class overview.

Properties

Name	Description
AsAnsiString	Allows to get and set the value as AnsiString.

AsAnsiStringNullable	Allows to get and set the value as AnsiString or Null.
AsBCD	Allows to get and set the value as TBCD.
AsBcdNullable	Allows to get and set the value as TBCD or Null.
AsBoolean	Allows to get and set the value as Boolean.
AsBooleanNullable	Allows to get and set the value as Boolean or Null.
AsByte	Allows to get and set the value as Byte.
AsByteNullable	Allows to get and set the value as Byte or Null.
AsBytes	Allows to get and set the value as Bytes.
AsBytesNullable	Allows to get and set the value as Bytes or Null.
AsCurrency	Allows to get and set the value as Currency.
AsCurrencyNullable	Allows to get and set the value as Currency or Null.
AsDate	Allows to get and set the value as TDate.
AsDateNullable	Allows to get and set the value as TDate or Null.
AsDateTime	Allows to get and set the value as TDateTime.
AsDateTimeNullable	Allows to get and set the value as TDateTime or Null.
AsDouble	Allows to get and set the value as Double.
AsDoubleNullable	Allows to get and set the value as Double or Null.
AsExtended	Allows to get and set the value as Extended.
AsExtendedNullable	Allows to get and set the value as Extended or Null.
AsGUID	Allows to get and set the value as TGUID.
AsGUIDNullable	Allows to get and set the value as TGUID or Null.
AsInt64	Allows to get and set the

	value as Int64.
AsInt64Nullable	Allows to get and set the value as Int64 or Null.
AsInteger	Allows to get and set the value as Integer.
AsIntegerNullable	Allows to get and set the value as Integer or Null.
AsInterface	Allows to get and set the value as Interface.
AsLongWord	Allows to get and set the value as LongWord.
AsLongWordNullable	Allows to get and set the value as LongWord or Null.
AsObject	Allows to get and set the value as Object.
AsShortInt	Allows to get and set the value as ShortInt.
AsShortIntNullable	Allows to get and set the value as ShortInt or Null.
AsSingle	Allows to get and set the value as Single.
AsSingleNullable	Allows to get and set the value as Single or Null.
AsSmallInt	Allows to get and set the value as SmallInt.
AsSmallIntNullable	Allows to get and set the value as SmallInt or Null.
AsString	Allows to get and set the value as String.
AsStringNullable	Allows to get and set the value as String or Null.
AsTime	Allows to get and set the value as TTime.
AsTimeNullable	Allows to get and set the value as TTime or Null.
AsTimeStamp	Allows to get and set the value as TimeStamp.
AsTimeStampNullable	Allows to get and set the value as TimeStamp or Null.
AsUInt64	Allows to get and set the value as UInt64.
AsUInt64Nullable	Allows to get and set the value as UInt64 or Null.

AsVariant	Allows to get and set the value as Variant.
AsWideString	Allows to get and set the value as WideString.
AsWideStringNullable	Allows to get and set the value as WideString or Null.
AsWord	Allows to get and set the value as Word.
AsWordNullable	Allows to get and set the value as Word or Null.
AsXML	Allows to get and set the value as XML.

Methods

Name	Description
Assign	Copies the value from Source.
CanBeNull	Returns True if the value can be set to Null; otherwise - False.
Clear	Sets the IsNull property to True
Clone	Creates a copy of TEDValue with the same data type and value.
CreateValue	Creates a new TEDValue instance with the specified data type.
DataType	Returns the type of the stored Value.
GetHashCode	Returns an integer containing the hash code.
IsNull	The property returns True if the value is Null; otherwise, the property returns False.
ToString	The method converts a value to string representation, so that it is suitable for display.

Request Support DAC Forum Provide Feedback

6.25.1.1.2 Properties

Properties of the **TEDValue** class.

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

Public

Name	Description
AsAnsiString	Allows to get and set the value as AnsiString.
AsAnsiStringNullable	Allows to get and set the value as AnsiString or Null.
AsBCD	Allows to get and set the value as TBCD.
AsBcdNullable	Allows to get and set the value as TBCD or Null.
AsBoolean	Allows to get and set the value as Boolean.
AsBooleanNullable	Allows to get and set the value as Boolean or Null.
AsByte	Allows to get and set the value as Byte.
AsByteNullable	Allows to get and set the value as Byte or Null.
AsBytes	Allows to get and set the value as Bytes.
AsBytesNullable	Allows to get and set the value as Bytes or Null.
AsCurrency	Allows to get and set the value as Currency.
AsCurrencyNullable	Allows to get and set the value as Currency or Null.
AsDate	Allows to get and set the value as TDate.
AsDateNullable	Allows to get and set the value as TDate or Null.
AsDateTime	Allows to get and set the value as TDateTime.
AsDateTimeNullable	Allows to get and set the value as TDateTime or Null.
AsDouble	Allows to get and set the value as Double.

AsDoubleNullable	Allows to get and set the value as Double or Null.
AsExtended	Allows to get and set the value as Extended.
AsExtendedNullable	Allows to get and set the value as Extended or Null.
AsGUID	Allows to get and set the value as TGUID.
AsGUIDNullable	Allows to get and set the value as TGUID or Null.
AsInt64	Allows to get and set the value as Int64.
AsInt64Nullable	Allows to get and set the value as Int64 or Null.
AsInteger	Allows to get and set the value as Integer.
AsIntegerNullable	Allows to get and set the value as Integer or Null.
AsInterface	Allows to get and set the value as Interface.
AsLongWord	Allows to get and set the value as LongWord.
AsLongWordNullable	Allows to get and set the value as LongWord or Null.
AsObject	Allows to get and set the value as Object.
AsShortInt	Allows to get and set the value as ShortInt.
AsShortIntNullable	Allows to get and set the value as ShortInt or Null.
AsSingle	Allows to get and set the value as Single.
AsSingleNullable	Allows to get and set the value as Single or Null.
AsSmallInt	Allows to get and set the value as SmallInt.
AsSmallIntNullable	Allows to get and set the value as SmallInt or Null.
AsString	Allows to get and set the value as String.
AsStringNullable	Allows to get and set the value as String or Null.
AsTime	Allows to get and set the

	value as TTime.
AsTimeNullable	Allows to get and set the value as TTime or Null.
AsTimeStamp	Allows to get and set the value as TimeStamp.
AsTimeStampNullable	Allows to get and set the value as TimeStamp or Null.
AsUInt64	Allows to get and set the value as UInt64.
AsUInt64Nullable	Allows to get and set the value as UInt64 or Null.
AsVariant	Allows to get and set the value as Variant.
AsWideString	Allows to get and set the value as WideString.
AsWideStringNullable	Allows to get and set the value as WideString or Null.
AsWord	Allows to get and set the value as Word.
AsWordNullable	Allows to get and set the value as Word or Null.
AsXML	Allows to get and set the value as XML.

See Also

- TEDValue Class
- TEDValue Class Members

6.25.1.1.2.1 AsAnsiString Property

Allows to get and set the value as AnsiString.

Class

TEDValue

Syntax

```
property AsAnsiString: AnsiString;
```

Remarks

If the value cannot be converted to AnsiString, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.2 AsAnsiStringNullable Property

Allows to get and set the value as AnsiString or Null.

Class

TEDValue

Syntax

```
property AsAnsiStringNullable: AnsiStringNullable;
```

Remarks

If the value cannot be converted to AnsiString, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.3 AsBCD Property

Allows to get and set the value as TBCD.

Class

TEDValue

Syntax

```
property AsBCD: TBcd;
```

Remarks

If the value cannot be converted to TBCD, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.4 AsBcdNullable Property

Allows to get and set the value as TBCD or Null.

Class

TEDValue

Syntax

```
property AsBcdNullable: TBcdNullable;
```

Remarks

If the value cannot be converted to TBCD, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.5 AsBoolean Property

Allows to get and set the value as Boolean.

Class

TEDValue

Syntax

```
property AsBoolean: Boolean;
```

Remarks

If the value cannot be converted to Boolean, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.6 AsBooleanNullable Property

Allows to get and set the value as Boolean or Null.

Class

TEDValue

Syntax

```
property AsBooleanNullable: BooleanNullable;
```

Remarks

If the value cannot be converted to Boolean, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.7 AsByte Property

Allows to get and set the value as Byte.

Class

TEDValue

Syntax

```
property AsByte: Byte;
```

Remarks

If the value cannot be converted to Byte, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.8 AsByteNullable Property

Allows to get and set the value as Byte or Null.

Class

TEDValue

Syntax

```
property AsByteNullable: ByteNullable;
```

Remarks

If the value cannot be converted to Byte, then an exception will be raised.

6.25.1.1.2.9 AsBytes Property

Allows to get and set the value as Bytes.

Class

TEDValue

Syntax

```
property AsBytes: TBytes;
```

Remarks

If the value cannot be converted to Bytes, then an exception will be raised.

6.25.1.1.2.10 AsBytesNullable Property

Allows to get and set the value as Bytes or Null.

Class

TEDValue

Syntax

```
property AsBytesNullable: TBytesNullable;
```

Remarks

If the value cannot be converted to Bytes, then an exception will be raised.

6.25.1.1.2.11 AsCurrency Property

Allows to get and set the value as Currency.

Class

TEDValue

Syntax

```
property AsCurrency: Currency;
```

Remarks

If the value cannot be converted to Currency, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.12 AsCurrencyNullable Property

Allows to get and set the value as Currency or Null.

Class

TEDValue

Syntax

```
property AsCurrencyNullable: CurrencyNullable;
```

Remarks

If the value cannot be converted to Currency, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.13 AsDate Property

Allows to get and set the value as TDate.

Class

TEDValue

Syntax

```
property AsDate: TDate;
```

Remarks

If the value cannot be converted to TDate, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.14 AsDateNullable Property

Allows to get and set the value as TDate or Null.

Class

TEDValue

Syntax

```
property AsDateNullable: TDateNullable;
```

Remarks

If the value cannot be converted to TDate, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.15 AsDateTime Property

Allows to get and set the value as TDateTime.

Class

TEDValue

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

If the value cannot be converted to TDateTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.16 AsDateTimeNullable Property

Allows to get and set the value as TDateTime or Null.

Class

TEDValue

Syntax

```
property AsDateTimeNullable: TDateTimeNullable;
```

Remarks

If the value cannot be converted to TDateTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.17 AsDouble Property

Allows to get and set the value as Double.

Class

TEDValue

Syntax

```
property AsDouble: Double;
```

Remarks

If the value cannot be converted to Double, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.18 AsDoubleNullable Property

Allows to get and set the value as Double or Null.

Class

TEDValue

Syntax

```
property AsDoubleNullable: DoubleNullable;
```

Remarks

If the value cannot be converted to Double, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.19 AsExtended Property

Allows to get and set the value as Extended.

Class

TEDValue

Syntax

```
property AsExtended: Extended;
```

Remarks

If the value cannot be converted to Extended, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.20 AsExtendedNullable Property

Allows to get and set the value as Extended or Null.

Class

TEDValue

Syntax

```
property AsExtendedNullable: ExtendedNullable;
```

Remarks

If the value cannot be converted to Extended, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.21 AsGUID Property

Allows to get and set the value as TGUID.

Class

TEDValue

Syntax

```
property AsGUID: TGUID;
```

Remarks

If the value cannot be converted to TGUID, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.22 AsGUIDNullable Property

Allows to get and set the value as TGUID or Null.

Class

TEDValue

Syntax

```
property AsGUIDNullable: TGUIDNullable;
```

Remarks

If the value cannot be converted to TGUID, then an exception will be raised.

6.25.1.1.2.23 AsInt64 Property

Allows to get and set the value as Int64.

Class

TEDValue

Syntax

```
property AsInt64: Int64;
```

Remarks

If the value cannot be converted to Int64, then an exception will be raised.

6.25.1.1.2.24 AsInt64Nullable Property

Allows to get and set the value as Int64 or Null.

Class

TEDValue

Syntax

```
property AsInt64Nullable: Int64Nullable;
```

Remarks

If the value cannot be converted to Int64, then an exception will be raised.

6.25.1.1.2.25 AsInteger Property

Allows to get and set the value as Integer.

Class

TEDValue

Syntax

```
property AsInteger: Integer;
```

Remarks

If the value cannot be converted to Integer, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.26 AsIntegerNullable Property

Allows to get and set the value as Integer or Null.

Class

TEDValue

Syntax

```
property AsIntegerNullable: IntegerNullable;
```

Remarks

If the value cannot be converted to Integer, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.27 AsInterface Property

Allows to get and set the value as Interface.

Class

TEDValue

Syntax

```
property AsInterface: IUnknown;
```

Remarks

If the value cannot be converted to Interface, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.28 AsLongWord Property

Allows to get and set the value as LongWord.

Class

TEDValue

Syntax

```
property AsLongWord: Cardinal;
```

Remarks

If the value cannot be converted to LongWord, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.29 AsLongWordNullable Property

Allows to get and set the value as LongWord or Null.

Class

TEDValue

Syntax

```
property AsLongWordNullable: LongWordNullable;
```

Remarks

If the value cannot be converted to LongWord, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.30 AsObject Property

Allows to get and set the value as Object.

Class

TEDValue

Syntax

```
property AsObject: TObject;
```

Remarks

If the value cannot be converted to Object, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.31 AsShortInt Property

Allows to get and set the value as ShortInt.

Class

TEDValue

Syntax

```
property AsShortInt: ShortInt;
```

Remarks

If the value cannot be converted to ShortInt, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.32 AsShortIntNullable Property

Allows to get and set the value as ShortInt or Null.

Class

TEDValue

Syntax

```
property AsShortIntNullable: ShortIntNullable;
```

Remarks

If the value cannot be converted to ShortInt, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.33 AsSingle Property

Allows to get and set the value as Single.

Class

TEDValue

Syntax

```
property AsSingle: Single;
```

Remarks

If the value cannot be converted to Single, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.34 AsSingleNullable Property

Allows to get and set the value as Single or Null.

Class

TEDValue

Syntax

```
property AsSingleNullable: SingleNullable;
```

Remarks

If the value cannot be converted to Single, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.35 AsSmallInt Property

Allows to get and set the value as SmallInt.

Class

TEDValue

Syntax

```
property AsSmallInt: SmallInt;
```

Remarks

If the value cannot be converted to SmallInt, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.36 AsSmallIntNullable Property

Allows to get and set the value as SmallInt or Null.

Class

TEDValue

Syntax

```
property AsSmallIntNullable: SmallIntNullable;
```

Remarks

If the value cannot be converted to SmallInt, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.37 AsString Property

Allows to get and set the value as String.

Class

TEDValue

Syntax

```
property AsString: string;
```

Remarks

If the value cannot be converted to String, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.38 AsStringNullable Property

Allows to get and set the value as String or Null.

Class

TEDValue

Syntax

```
property AsStringNullable: StringNullable;
```

Remarks

If the value cannot be converted to String, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.39 AsTime Property

Allows to get and set the value as TTime.

Class

TEDValue

Syntax

```
property AsTime: TTime;
```

Remarks

If the value cannot be converted to TTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.40 AsTimeNullable Property

Allows to get and set the value as TTime or Null.

Class

TEDValue

Syntax

```
property AsTimeNullable: TTimeNullable;
```

Remarks

If the value cannot be converted to TTime, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.41 AsTimeStamp Property

Allows to get and set the value as TimeStamp.

Class

TEDValue

Syntax

```
property AsTimeStamp: TSQLTimeStamp;
```

Remarks

If the value cannot be converted to TimeStamp, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.42 AsTimeStampNullable Property

Allows to get and set the value as TimeStamp or Null.

Class

TEDValue

Syntax

```
property AsTimeStampNullable: TSQLTimeStampNullable;
```

Remarks

If the value cannot be converted to TimeStamp, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.43 AsUInt64 Property

Allows to get and set the value as UInt64.

Class

TEDValue

Syntax

```
property AsUInt64: UInt64;
```

Remarks

If the value cannot be converted to UInt64, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.44 AsUInt64Nullable Property

Allows to get and set the value as UInt64 or Null.

Class

TEDValue

Syntax

```
property AsUInt64Nullable: UInt64Nullable;
```

Remarks

If the value cannot be converted to UInt64, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.45 AsVariant Property

Allows to get and set the value as Variant.

Class

TEDValue

Syntax

```
property AsVariant: Variant;
```

Remarks

If the value cannot be converted to Variant, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.46 AsWideString Property

Allows to get and set the value as WideString.

Class

TEDValue

Syntax

```
property AsWideString: string;
```

Remarks

If the value cannot be converted to WideString, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.47 AsWideStringNullable Property

Allows to get and set the value as WideString or Null.

Class

TEDValue

Syntax

```
property AsWideStringNullable: WideStringNullable;
```

Remarks

If the value cannot be converted to WideString, then an exception will be raised.

Request Support DAC Forum Provide Feedback

6.25.1.1.2.48 AsWord Property

Allows to get and set the value as Word.

Class

TEDValue

Syntax

```
property AsWord: Word;
```

Remarks

If the value cannot be converted to Word, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.49 AsWordNullable Property

Allows to get and set the value as Word or Null.

Class

TEDValue

Syntax

```
property AsWordNullable: WordNullable;
```

Remarks

If the value cannot be converted to Word, then an exception will be raised.

 Request Support DAC Forum Provide Feedback

6.25.1.1.2.50 AsXML Property

Allows to get and set the value as XML.

Class

TEDValue

Syntax

```
property AsXML: IXMLDocument;
```

Remarks

If the value cannot be converted to XML, then an exception will be raised.

6.25.1.1.3 Methods

Methods of the **TEDValue** class.

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

Public

Name	Description
Assign	Copies the value from Source.
CanBeNull	Returns True if the value can be set to Null; otherwise - False.
Clear	Sets the IsNull property to True
Clone	Creates a copy of TEDValue with the same data type and value.
CreateValue	Creates a new TEDValue instance with the specified data type.
DataType	Returns the type of the stored Value.
GetHashCode	Returns an integer containing the hash code.
IsNull	The property returns True if the value is Null; otherwise, the property returns False.
ToString	The method converts a value to string representation, so that it is suitable for display.

See Also

- TEDValue Class
- TEDValue Class Members

Reserved.

6.25.1.1.3.1 Assign Method

Copies the value from Source.

Class

TEDValue

Syntax

```
procedure Assign(Source: TEDValue);
```

Parameters

Source

Request Support DAC Forum Provide Feedback

6.25.1.1.3.2 CanBeNull Method

Returns True if the value can be set to Null; otherwise - False.

Class

TEDValue

Syntax

```
function CanBeNull: boolean;
```


Request Support DAC Forum Provide Feedback

6.25.1.1.3.3 Clear Method

Sets the IsNull property to True

Class

TEDValue

Syntax

```
procedure Clear;
```

 Request Support DAC Forum Provide Feedback

6.25.1.1.3.4 Clone Method

Creates a copy of TEDValue with the same data type and value.

Class

TEDValue

Syntax

```
function Clone: TEDValue;
```

 Request Support DAC Forum Provide Feedback

6.25.1.1.3.5 CreateValue Method

Creates a new TEDValue instance with the specified data type.

Class

TEDValue

Syntax

```
class function CreateValue(ADataType: Word): TEDValue;
overload;class function CreateValue(ADataType: Word; ACanBeNull:
boolean): TEDValue; overload;class function
CreateValue(ADataType: Word; ACanBeNull: boolean;
AllowNullableString: Boolean; AllowNullableBytes: Boolean):
TEDValue; overload;
```

Parameters

ADataType

Specifies a data type.

ACanBeNull

Specifies whether it can be Null.

 Request Support DAC Forum Provide Feedback

6.25.1.1.3.6 DataType Method

Returns the type of the stored Value.

Class

TEDValue

Syntax

```
function DataType: Word;
```

 Request Support DAC Forum Provide Feedback

6.25.1.1.3.7 GetHashCode Method

Returns an integer containing the hash code.

Class

TEDValue

Syntax

```
function GetHashCode: Integer; override;
```

 Request Support DAC Forum Provide Feedback

6.25.1.1.3.8 IsNull Method

The property returns True if the value is Null; otherwise, the property returns False.

Class

TEDValue

Syntax

```
function IsNull: boolean;
```

Remarks

The value can be set to Null by assigning the True value to IsNull. The False value can't be

assigned to IsNull. IsNull can be set to False only by assigning a particular value to the Value property.

 Request Support DAC Forum Provide Feedback

6.25.1.1.3.9 ToString Method

The method converts a value to string representation, so that it is suitable for display.

Class

TEDValue

Syntax

```
function ToString: string; override;
```

 Request Support DAC Forum Provide Feedback

6.25.1.2 TEDValues Class

List of TEDValue.

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

Unit

EntityDAC.Values

Syntax

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

 Request Support DAC Forum Provide Feedback

6.25.1.2.1 Members

TEDValues class overview.

Properties

Name	Description
Count	Returns the number of values in the list.
Items	Lists the values in the list.

Methods

Name	Description
Add	Adds new values to the list.
Assign	Copies the contents of the Source value list to the current list.
Clear	Deletes all values from the list.
Delete	Deletes values from the list by Index.

 Request Support DAC Forum Provide Feedback

6.25.1.2.2 Properties

Properties of the **TEDValues** class.

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

Public

Name	Description
Count	Returns the number of values in the list.
Items	Lists the values in the list.

See Also

- TEDValues Class
- TEDValues Class Members

 Request Support DAC Forum Provide Feedback

6.25.1.2.2.1 Count Property

Returns the number of values in the list.

Class

TEDValues

Syntax

```
property Count: Integer;
```

 Request Support DAC Forum Provide Feedback

6.25.1.2.2.2 Items Property(Indexer)

Lists the values in the list.

Class

TEDValues

Syntax

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

Parameters

Index

 Request Support DAC Forum Provide Feedback

6.25.1.2.3 Methods

Methods of the **TEDValues** class.

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

Public

Name	Description
Add	Adds new values to the list.
Assign	Copies the contents of the

	Source value list to the current list.
Clear	Deletes all values from the list.
Delete	Deletes values from the list by Index.

See Also

- TEDValues Class
- TEDValues Class Members

6.25.1.2.3.1 Add Method

Adds new values to the list.

Class

TEDValues

Syntax

```
function Add(DataType: Word): TEDValue; overload;function
Add(DataType: Word; CanBeNull: boolean): TEDValue;
overload;procedure Add(Value: TEDValue); overload;
```

Parameters

DataType

Specifies the data type of the value

6.25.1.2.3.2 Assign Method

Copies the contents of the Source value list to the current list.

Class

TEDValues

Syntax

```
procedure Assign(Source: TEDValues);
```

Parameters

Source

The list of TEDValues

 Request Support DAC Forum Provide Feedback

6.25.1.2.3.3 Clear Method

Deletes all values from the list.

Class

TEDValues

Syntax

```
procedure Clear;
```

 Request Support DAC Forum Provide Feedback

6.25.1.2.3.4 Delete Method

Deletes values from the list by Index.

Class

TEDValues

Syntax

```
procedure Delete(Index: Integer);
```

Parameters

Index

The index of the deleted item

 Request Support DAC Forum Provide Feedback

6.25.2 Types

Types in the **EntityDAC.Values** unit.

Types

Name	Description
TEDValueClass	Class implementing the TEDValue basic class

Request Support DAC Forum Provide Feedback

6.25.2.1 TEDValueClass Class Reference

Class implementing the TEDValue basic class

Unit

EntityDAC.Values

Syntax

```
TEDValueClass = class of TEDValue;
```


Request Support DAC Forum Provide Feedback