1 Disclaimer

This software is delivered as it is. The author assumes no liability for damages, direct or consequential, which may result from it's use.

2 Copyright / Licensing

The software is owned by gig mbh berlin (www.gig-mbh.de).

Two different licenses are available:

1. Free License

   Everyone who wants to use the free license has to register with his full name and address via support@gig-mbh.de.

   Every software where parts of our free software were used for development has to be free also including source code.

   If you derive anything from our software it must be clearly stated that it was derived from it.

   Full source code is included.

2. Extended License

   Licenses have to be bought by a per developer basis. Site licenses would be available on demand.

   Applications built with this software could be deployed without royalty fees. They can be sold and don't need to include source code.

   Distribution of a derived version of our software is only allowed with the explicit agreement of the author.

   Full source code is included.

3 Support

Support is available via email at support@gig-mbh.de for free but it need not remain so in the future.

4 Introduction

This document describes the function of the component TCustomDataProviderEC.

This class defines an interface to enable our TMemTableEC dataset component to save and load data from different kind of data storage systems. Implementation is done by deriving and overwriting a minimum set of it's virtual methods.

For an example on implementation take a look at our TIBDataProviderEC component.

The component is completely written in C++ and was developed under C++Builder 5 Pro but it should be usable on C++ Builder 6 if compiled in it's environment.

Questions, bug reports , enhancement requests, suggestions for improving the docs and comments should be send to support@gig-mbh.de.

5 Methods

GetRecord

Description: Is called whenever a new record from the data store has to be retrieved. The contens of the record has to be assigned to the fields of the TMemTableEC dataset. The dataset can be accessed through the FDataSet pointer.

Prototype : virtual bool __fastcall GetRecord(bool first) = NULL

Parameters: **first** - If set to true the first record of the data store has to be retrieved, otherwise the next record has to be retrieved.

Return values : Return true until no more records are available.

Type: public

InsertRecord

Description: Is called whenever a new record has to be inserted into the data store. The fieldvalues can be accessed by the default TDatatSet properties and methods. The dataset can be accessed through the FDataSet pointer.

Prototype : virtual void __fastcall InsertRecord(void) = NULL

Parameters: none

Return values : none

Type: public

DeleteRecord

Description: Is called whenever a record has to be deleted from the data store. The field values can be accessed by the default TDatatSet properties and methods. The dataset can be accessed through the FDataSet pointer. If you implement a data provider which does not support single record changes which save the whole dataset contens in one pass, this method need not to be implemented.

Prototype : virtual void __fastcall DeleteRecord(void) = NULL

Parameters: none

Return values : none

Type: public

ModifyRecord

Description: Is called whenever a record has to be modifed in the data store. The field values can be accessed by the default TDatatSet properties and methods. For modification in most cases you will need the know the new and the old field values. By default you will receive the new field values. Informationon how to switch between old and new values you will find at the CurRecordBuf property description. The dataset can be accessed through the FDataSet pointer. If you implement a data provider which does not support single record changes which save the whole dataset contens in one pass, this method need not to be implemented.

Prototype : virtual void __fastcall ModifyRecord(void) = NULL

Parameters: none

Return values : none

Type: public

RefreshRecord

Description: Is called whenever a record has to be reread immediately after a new or modifed record was posted to the data store. This is the case if the RereadChanges property of the TMemTableEC component is set to true. If you have a data storage system which does not apply any changes to new or modifed records by itself this method need not to be implemented. The field values can be accessed by the default TDatatSet properties and methods. The dataset can be accessed through the FDataSet pointer.

Prototype : virtual void __fastcall RefreshRecord(void) = NULL

Parameters: none

Return values : none

Type: public

StartTransaction

Description: All read and write operation are allways initiated within a tranactional context. That means before a series of read or write operations begins a transaction is allways started and after that the transaction is allways ended even if any operation in between fails. The method also gives you the information what kind of action will follow.

Prototype : virtual void __fastcall StartTransaction(TDataProviderAction action) = NULL

Parameters: **action** - Specifies what kinf of action will take place within this transaction. Possible values are:

datprLoad - Load was called from the TMemTableEC component

datprSave - Save was called from the TMemTableEC component

datprApplyChg - Any kind of modification was initiated from the TMemTableEC component (delete, insert or modify record)

datprConRead - Continous read was started (see TMemTableEC component for further details.

Return values : none

Type: public

EndTransaction

Description: After finishing a set of read and write operation the current transaction is allways ended. If you want to distinguish between committing and rolling back the transaction, you have to take care of the conditions when you want to rollback and when you want to commit by yourself.

Prototype : virtual void __fastcall EndTransaction(void) = NULL

Parameters: none

Return values : none

Type: public

6 Properties

CurRecordBuf

Description: A pointer to the current record buffer. On insert, delete, read and refresh operations where you only have one record buffer you do not need to care about it. On modify operations you could assign FOldRecordBuf or FNewRecordBuf to this property to switch between the values before and after the modifications. After switching, all TDataSet field access actions refer to the apropriate record buffer.

Definition : __property char *CurRecordBuf = {read=GetCurRecordBuf, write=SetCurRecordBuf}

Type: protected