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 TMemTableEC.

The component implements a TDataSet descendant where the data is stored in-memory and not in a database which is compatible to the many data aware components available like grids and edit controls.

Furthermore it introduces functionality which is used by our grid component TDBGridEC to have sortable columns and incremental searching without the need for coding a single line.

The data for this memory table could be filled in manually or can be automatically synchronised with an external data store by linking it to one of our TCustomDataProviderEC descendant components. The TCustomDataProviderEC virtual baseclass is kept quite simple so it should be not problem to derive your own data provider class from it to synchronise data with different varities of data storage systems. Further information could be obtained by the documentation of the TIBDataProviderEC component.

There are different ways how data changes can be handled. It is possible that every change is directly applied only to the in-memory table, also directly to the external data store or that all changes are recorded and applied later in one pass.

Selecting the option to record changes additionally gives you the ability to define any number of savepoints so that you can rollback and commit changes in blocks for transactional purpose. This way even nested transactioning can be implemented.

For sorting you could choose between natural sort order where every record stays on the position where it was inserted or by specifying multiple sort orders you can switch between. A sort order can consist of multiple columns where ascending, descending and case sensivity could be specified for every single column seperately. For operating on larger datasets a sortorder can be indexed. This also gives you the ability to use the GotoKey method for fast record retrival.

In memory indexing can be used very flexible becuase it is quite fast. Building an index on a single integer column for 100,000 records takes less that half a second on a PIII 850 Mhz with 256 MB RAM under Win2k.

In contrast to other memory tables we implemented the whole datastorage with linked lists and AVLTrees and not with arrays. This is more memory consuming but has it’s advantages on massive data changes.

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

ClearTable

Description: Clears the whole in-memory table and all record changes.

Prototype : void __fastcall Clear(void)

Return values : none

Type: public

Locate

Description: See description in VCL help. Be aware that Locate does not take advantage of indexes. Therefore use GotoKey instead.

LocateNext

Description: Similar to Locate, but search starts with the record after the current one and not at the beginning of the table.

SetKey

Description: When calling SetKey the dataset enters the dsSetKey state. After that you could set field values for searching. The searching process is started by calling the GotoKey method. Keep in mind that you can not only use the index of the current sort order for searching but every index which was defined.

Prototype : void __fastcall SetKey(bool clear = true)

Parameters: **clear** - If set to true the key buffer is cleared other wise all current values in the record buffer are kept.

Return values : none

Type: public

GotoKey

Description: After the dataset was brought to the dsSetKey state by calling SetKey and key values have been specified the search process can be started by calling this method. If the specified record is not found the current record position is maintained.

Prototype : bool __fastcall GotoKey(bool exact = true, int orderitemid = 0, int colcount = -1, bool center = true)

Parameters: **exact** - If set to true a record is only retrieved if all columns which are taken into account exactly match a record.
If set to false the next matching record according to the specified sortorder is retrieved if no matching record exists. If also no next record exists the function returns false.

orderitemid - specifies the id of the sortorder of which the index will be used for searching. If set to 0 the current sort order's index is used.

colcount - If set to -1 all columns of an index will be used for searching. Otherwise only the first n columns will be used.

center - If set to true the new record is centered whithin a related grid control otherwise the position within the grid is kept.

Return values : **true** if a record is found, otherwise **false**

Type: public

GotoBookmark

Description: See description in VCL help. This method additionally implements a parameter center which is similar to that one of GotoKey

Prototype : void __fastcall GotoBookmark(void *bookmark, bool center)

CreateBlobStream

Description: See description in VCL help.

Load

Description: Retrieves records from the linked data provider and loads them into the dataset.

Prototype : void __fastcall Load(bool clear = true, bool keeppos = false)

Parameters: **clear** - If set to true the in-memory table is clear before retrieving records, otherwise the new records are added to the existing ones.

keeppos - If set to true the current active record is preserved if possible. Preserving the current record is not possible if you set clear to true and the current record's value(s) of the current sort order is (are) not unique.

Return values : none

Type: public

Save

Description: Saves all records of the dataset to the linked data provider. This method is only useful with data providers supporting data stores which are not able to store single record changes and where the whole data has to be stored in one pass (e.g. CSV files)

Prototype : void __fastcall Save(void)

Return values : none

Type: public

Rollback

Description: Rolls back changes which have not yet been applied to the linked data provider. Using this method is only possible if UpdateMode is set to memtabumRecordChanges.

Prototype : void __fastcall Rollback(TMemTabChangeListNode *savepoint = NULL)

Parameters: **savepoint** - all changes which have been made since the specified savepoint are rolled back. If NULL is specified all changes are rolled back. Keep in mind that all savepoints which are located temporaly behind this savepoint are no longer valid. See property CurrentSavePoint for more details.

Return values : none

Type: public

Commit

Description: Commit pending changes which have not been commited yet. Using this method is only possible if UpdateMode is set to memtabumRecordChanges.

Prototype : void __fastcall Commit(bool applychanges = true, TMemTabChangeListNode *savepoint = NULL)

Parameters: **applychanges** - if set to true the changes are also written to the linked data provider. You could specify false if you are working with a data provider which does not support writing changes on a per record base, where you would write all records back with the Save method when finished.

savepoint - all changes which have been made before the specified savepoint are commited. If NULL is specified all changes are commited. Keep in mind that all savepoints which are located temporaly before this savepoint are no longer valid. See property CurrentSavePoint for more details.

Return values : none

Type: public

UndoLastChange

Description: Last change is undone. Using this method is only possible if UpdateMode is set to memtabumRecordChanges. Committed changes cannot be undone.

Prototype : void __fastcall UndoLastChange(void)

Return values : none

Type: public

6 Properties

CurrentSavePoint

Description: Represents the current save point of the dataset. If you want to refer to this point at a later time simply save this pointer to a variable of a compatible type. A savepoint can be used to rollback all changes up to, or to commit all changes which have been made before a specified point in time. Whenever any change took place in the dataset a new savepoint is generated. Keep in mind that if you save multiple CurrentSavePoints of different points in time, committing or rolling back up to one save point could invalidated other save ponts as well. This property can only be used if UpdateMode is set to memtabumRecordChanges.

Definition : __property TMemTabChangeListNode *CurrentSavePoint = {read=GetCurrentSavePoint}

Type: public

Changes

Description: Points to a list of all changes which have not been committed yet. If you are interrested in any details take a look at the headerfiles. This property can only be used if UpdateMode is set to memtabumRecordChanges.

Definition : __property TMemTabChangeList *Changes = {read=FChanges}

Type: public

ReadOnly

Description: If set to true modifications through data-sensitive controls is no longer possible.

Definition : __property bool ReadOnly = {read=FReadOnly, write=FReadOnly, default=false}

Type: published

ReadChanges

Description: After posting a new/modifed record by the linked data provider to the underlying data store, the record is reread by the dataprovider if this property is set to true. This could be useful if changes are made by the data storage system itself on posting modifications and these changes should be reflecred by the dataset. An example would be triggers of SQL servers.

Definition : __property bool ReadChanges = {read=FReadChanges, write=FReadChanges, default=false}

Type: published

UpdateMode

Description: UpdateMode specifies how modification to the dataset are handeled by dataset and the linked data provider.

memtabumNone - Modification are directly applied to the in-memory table. No changes are recorded and no actions are initiated for the linked data provider on modifications.

memtabumRecordChanges - All changes are recorded. No actions are initiated for the linked data provider on modifications.

memtabumDirectUpdate - No changes are recorded. Every modification is immediately applied to the data store by the linked data provider.

Definition : __property TMemTabUpdateMode UpdateMode = {read=FUpdateMode, write=FUpdateMode, default=memtabumNone};

Type: published

OrderItemId

Description: Use this property to switch between the different sort orders which have been specified by the OrderItems property. The active record will allways be maintained. If 0 (zero) is specified a natural sort order is assumed. That means all records will maintain their position where they have been inserted. A negative value can be used to revert the order of a referenced OrderItem.

Definition : __property int OrderItemId = {read=GetOrderItemId, write=SetOrderItemId, default=0}

Type: published

OrderItems

Description: Specifies different sort orders for the dataset you could switch between during runtime in the following form for every line: <id>;<clause>;<indexed>. Id is a positive numeric value the sortorder will be referenced by. Indexed can be true or false and specifies if an in-memory index will be maintained for the specified sort order to speed up operations. Clause has the following syntax:
<fieldname> [DESC] [IGNORECASE] [, <fieldname> ...]
where fieldname specifies the name of the column, if DESC is specified this column will be used in descending order and IGNORECASE specifies that sorting will not be case sensitive for this column. Multiple columns can be specified to build compound sortorders. As for sort orders the current windows local scheme is used, IGNORECASE is mostly useless because already part of the windows scheme

Definition : __property TMemTabOrderItems *OrderItems = {read=GetOrderItems, write=SetOrderItems}

Type: published

DataProvider

Description: Pointer to the dataprovider which connects the dataset to an external data store.

Definition : __property TCustomDataProviderEC *DataProvider = {read=FDataProvider, write=SetDataProvider};

Type: published

MaxIndexCount

Description: Specifies the max number of in-memory indices which could be created. This property could only be changed as long as the dataset is not opened. For every possible index a four byte pointer has to be reserved in every record buffer. So increasing this value may give you more flexibility during operation but may also waste memory.

Definition : __property int MaxIndexCount = {read=FMaxIndexCount, write=SetMaxIndexCount}

Type: published

FieldDefs

Description: Here you specify the structure of the underlying in-memory table. The following datatypes are supported so far: ftBCD, ftBlob, ftBoolean, ftCurrency, ftDate, ftDFateTime, ftFloat, ftGraphic, ftInteger, ftLargeint, ftMemo, ftSmallint, ftString, ftTime. Largeint fields are also supported under BCB 5 although the variant datatype does not support them. There are two special fields which could be defined: _BOOKMARK (type ftLargeint) to retrieve a unique identifier for every record. _MODIFIED (type ftBoolean) to see if there are uncommitted changes of a record.

Definition : __property TfieldDefs *FieldDefs = {read=FFieldDefs, write=SetFieldDefs}

Type: published

AutoCalcFields

Description: See description in VCL help.

Filtered

Description: See description in VCL help.

ProviderContinousRead

Description: Records are retrieved form the provider continously and not at one time. This could be useful when processing large datasets as records which are not needed would not be retrieved from the provider. However this functionality is limited in the way that some actions would cause a "fetch all" from the provider like specifying a sort order different from natural (OrderItemId = 0) or positioning on the last record. Also keep in mind that in this mode a transaction is active as long as the dataset is active.

Definition : __property bool ProviderContinousRead =
{read=FProviderContinousRead,
write=SetProviderContinousRead,
default = false};

Type: published

RecNoSupport

Description: If set to true the dataset supports record numbers. This enables grid controls to display a different type of scrollbar which reflects the number of records in a table and where the thumb can be used for more precise navigation. As all data is stored in linked lists internally there is a performance penalty when accessing records via record numbers on large datasets, because record numbers are allways calculated dynamically. However this should not be a problem with datasets smaller that 10k records.

Definition : __property bool RecNoSupport = {read=FRecNoSupport,
write=SetRecNoSupport,
default = false};

Type: published

7 Events

OnCreateFields

Description: This event is fired just after the field objects have been created and before they are bind to the dataset. If you want to define calculated fields and don't want to use the object inspector you can do it here.

Handler : void __fastcall (__closure *TNotifyEvent)(TObject *Sender)

Type: published

For a description of the additional events available take a look at the VCL docs of the TDataSet component.