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 component is a direct TDataSet descendant which could be integrated seamless with all other existing IBX components. It links to the same components (TIBDatabase, TIBTransaction) and uses the same core component (TIBSQL) like TIBTable and TIBDataSet internally.

It's special purpose is to access large datasets in a table like manner without some of the restrictions you will find when using TIBTable and TIBDataSet. On navigating to a specific record via key values in contrast to the standard IBX controls only the records necessary for display will be fetched and buffered . Also committing and restarting a transaction will not invalidate the current record pointer.

Of course it could also be used to access small datasets, but the benefits you will get over using TIBTable and TIBDataSet are much more noticable on large datasets.

If you want to get smaller result sets of queries which take a very long time to execute it is much more preferable to use our in-memory table TMemTableEC in conjunction with our Interbase data provider TIBDataProvider instead.

Allways keep in mind that this component tries not to achive all possible IB tasks in a perfect manner. Instead it is intended to do some very special tasks efficiently. The key to success is to choose the right component for the specific problem you have to solve. Therefore you could mix it with other IBX component without any problem.

To fulfill it's task the component maintains several queries internally and modifies them in a way that only the records necessary for display will be fetched.

The basic requirement to work efficient is to have an ascending and a descending index on every column you wish to be used for sorting and these columns must not contain NULL values. The reason for this restriction is that NULL columns are allways retrieved last no matter if the sort order is specified to be ascending or descending and this makes internal repositioning impossible on those columns. This requirement could easily be arranged for string fields by specifying a trigger setting the column to '' (empty string) if the column is NULL as empty strings are considered to be the lowest possible value for string columns and so are sorted correctly.

Secondly the sortorder specified must represent a unique value. The reason for this is that the ascending and the descending sortorder must result in an exactly reverse order of the retrieved rows. If the sortorder represents a non unique value the order in which the rows for duplicate values are retrieved is undefined.
To achive sorting of non unique columns simply append the primary key to the order by clause.

If you use a small dataset only having a few hundred records there is no need to have any indexes at all but on dataset having a few 100,000 records this is mandatory.

The porperties you have to specify at least for a fully functional (editable) dataset are only Database, Transaction, SelectSQL and KeyColumns.

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.

Please allways ensure that you are using the latest IBX components available from Jeff Overcash's Homepage, because many issues have been fixed since it was shipped with C++Buider 5.

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

5 Methods

GotoBookmark

Description: Sets the cursor to a record which was previously marked by GetBookmark

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

Parameters: **bookmark** - spcifies a pointer which was previously returned by a call to GetBookmark.

center - if true the record is centered within a linked grid control, otherwise the current row is kept.

Return values : none

Type: public

RefreshActiveRecord

Description: Refreshes the currently active record

Prototype : void __fastcall RefreshActiveRecord(void)

Parameters: none

Return value : none

Type: public

LockRecord

Description: Locks the current record

Prototype : void __fastcall LockRecord(void)

Parameters: none

Return value : none

Type: public

SetKey

Description: Use the SetKey method to change the dataset state to dsSetKey. After that you could set key values for locating to another record by calling GotoKey

Prototype : void __fastcall SetKey(bool clear = true)

Parameters: **clear** - spcifies if the record buffer should be cleard or if the values of the current record are maintained.

Return value : none

Type: public

GotoKey

Description: Use the GotoKey method to locate to a new record which was specified by a previous call to SetKey. If the method fails the current record is maintained.

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

Parameters: **exact** - if set to true the call is only successful if the key values exacly match a record. If set to false the next record according to the current sort order is retrieved if no matching record is found.

pkey - if set to false the method uses the key values of the current sortorder otherwise the key values of the primary key (specified by the KeyColumns property) are used**.**

colcount - if -1 is specified all columns of the key are used for searching, otherwise the first n columns of the key are used.

center - if true the record is centered within a linked grid control, otherwise the current row is kept.

Return value : true if the function succeeds, false if not.

Type: public

Sync

Description: Use this method to syncronise the record position of two IBScrollSetEC datasets. It is also possible to sync two datasets which have a different record layout. The only thing necessary is that both datasets have compatible field types specified in their KeyColumns property. If sync fails an exception is rasied.

Prototype : void __fastcall Sync(TIBScrollSetEC *from, bool center = true)

Parameters: **from** - the dataset from which the current record location is taken for syncronisation.

center - if true the record is centered within a linked grid control, otherwise the current row is kept.

Return value : none.

Type: public

CreateBlobStream

Description: Described in the TDataSet documentation of the VCL.

6 Properties

Params

Description: Described in the TIBSQL documentation of the VCL.

Type: public

Database

Description: Specifies the database this component will conntect to.

Definition : __property TIBDatabase *Database = {read=GetDatabase,
write=SetDatabase}

Type: published

Transaction

Description: Specifies the transaction this component will use.

Definition : __property TIBTransaction *Transaction = {read=GetTransaction,
write=SetTransaction}

Type: published

SelectSQL

Description: Specifies the SQL code which will be executed as query.
Attention : if no OrderItemId property is specified the query must contain a order by clause

Definition : __property TStrings *SelectSQL = {read=GetSelectSQL,
write=SetSelectSQL}

Type: published

ModifySQL

Description: Every line contains a value assignment of the following form: <column>=<value> for applying the modifications made on the current record. Value could be a DataSet field name prefixed by a ':' or any other valid expression for value assignments within an UPDATE SQL statement. If this property is not specified a default is generated.

Definition : __property TStrings *ModifySQL = {read=GetModifySQL, write=SetModifySQL}

Type: published

InsertSQL

Description: Every line contains a value assignment of the following form: <column>=<value> for applying the values on a newly inserted record. Value could be a DataSet field name prefixed by a ':' or any other valid expression for value assignments within an INSERT SQL statement. If this property is not specified a default is generated.

Definition : __property TStrings *InsertSQL = {read=GetInsertSQL, write=SetInsertSQL}

Type: published

WhereClause

Description: This property represents the WHERE clause of the SelectSQL property. It is possible to change this value at runtime to achieve dynamic server side filtering. After changing this property the method Refresh should be called to reflect the changes. The active record is maintained after this call (Exception : the active record does not match the new filter condition)

Definition : __property AnsiString WhereClause = {read=GetWhereClause, write=SetWhereClause, stored=false}

Type: published

KeyColumns

Description: Specifies the unique identifiers of a single row. This property must allways be specified in the form <column> [;<column> ...]

Definition : _property TIBSSOrderItems *OrderItems = {read=GetOrderItems, write=SetOrderItems}

Type: published

MainTable

Description: For query generation it is allways assumed that the first table specified whithin the FROM clause is ment to be the main table (The main table is that one on which insertions and modifications will be applied). If you do not want the first table to be the main table within a multi table query (e.g. for optimization reasons) you could specify its name here.

Definition : __property AnsiString MainTable = {read=FMainTable, write=SetMainTable}

Type: published

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

GeneratorFields

Description: This property links generators to table fields in the format <fieldname>;<generator_name>;<increment>. This property is necessary to create autoincrement fields on new records so that the dataset component is able to keep track of a newly inserted record. If you would assign a new key value via a trigger the dataset will loose focus on it after it was posted. You must not assign this generator also via a trigger on that table otherwise it fill be fired twice. Beware that fieldname specifies the datataset fieldname and not the column name which may vary if you use aliases.

Definition : __property TIBSSGeneratorFields *GeneratorFields = {read=FGeneratorFields, write=SetGeneratorFields}

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>. Id is a positive numeric value the sortorder will be referenced by. Clause is the ascending order clause you want to assign for this id. (the descending sortorder is generated automatically). You could switch between the sort orders via SortOrderId property. When switching between sortorders the active record will allways be maintained.

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

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 the sort order of the querie's ORDER BY clause is used. 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

PessimisticLocking

Description: You could choose between optimistic (default) and pessimistic locking mode when modifying records. With pessimistic locking records are locked before the dataset switches to edit mode. The behavour is also interrelated to the isolation mode of the assigned transaction. As a guidline the following combinations are useful: optimistic + snapshot or pessimistic + read_commited.

Definition : __property bool PessimisticLocking = {read=FPessimisticLocking, write=SetPessimisticLocking, default=false}

Type: published

RefreshBeforeEdit

Description: If set to true the current record is reread before switching to edit mode to refrect any changes which have been made since it was last read. This is useful in combination of pessimistic locking and read_commited transaction to ensure the most current data is presented after the record has been locked.

Definition : __property bool RefreshBeforeEdit = {read=FRefreshBeforeEdit, write=FRefreshBeforeEdit, default=false}

Type: published

GetRecordsOnOpen

Description: Setting to false prevents fetching records on activating the dataset. This could be useful if it is desired to set filters or to locate to a specific record directly after opening the dataset.

Definition : __property bool GetRecordsOnOpen = {read=FGetRecordsOnOpen, write=FGetRecordsOnOpen, default=true}

Type: published

Suspend

Description: Setting to false prevents fetching records. When setting back to true Refresh has to be called explicity to fetch the current records. This property could be used to prevent unnecessary fetching of records esp. for applications operating over a WAN link. Records for example are fetched when the dataset is assigned to a datasource or when it is opened.

Definition : __property bool Suspend = {read=FSuspend, write=FSuspend, default=false}

Type: published

Active

Description: Described in the TDataSet documentation of the VCL.

AutoCalcField

Description: Described in the TDataSet documentation of the VCL.

Filtered

Description: Described in the TDataSet documentation of the VCL.

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.