

User's Guide for SqlitePass Database Components Version 0.34

Last Revision
2008-03-22

SqlitePass components provide a easy and fast access to sqlite databases

At the moment, it partially supports reading from / writing to databases created with :



The [Kexi project](#)
for Linux and Windows



The [SQLite Expert](#)
for Windows



[SQLite Administrator](#)
Database manager for Windows

SQLITE4FPC

[Sqlite4Fpc](#)
Database component for FPC

This project is open source, released under [LGPL license](#). Libraries and components are free and currently tested with Delphi 4 & 6 - Lazarus 0.9.24 – FPC 2.0.4. Let me know in you want to use those components with other Delphi versions.

User's Guide Summary

Packages Description.....	3
Packages Installation	7
Quick Start.....	9
The TSQLitePassDatabase component.....	10
How to select a database.....	10
Opening and closing a database	11
How to use an alternative sqlite library	11
Working with fields.datatypes	12
How to use Transactions	15
How to work with foreign databases.....	15
Others properties (to be documented.....)	15
The TSQLitePassDataset component.....	17
Roadmap for version 0.35.....	20

Packages Description

Content of SQLitePass_x.xx Directory

• Demo program in /Demo

/Databases	Databases samples in Kexi or SqliteAdministrator format.
/Delphi4	Demo Project files for Delphi 4
/Lazarus	Demo Project files for Lazarus
/Sources	Source files for the demo project. <ul style="list-style-type: none"> • *.pas files are shared by Delphi and Lazarus projects • *.dfm files are used by Delphi • *.lfm files are used by Lazarus

• Documentation in /Documentation

• /UserGuide.odt or UserGuide..pdf	The file you are reading...
• /Changes.odt or Changes.pdf	Latest changes – Versions history

• Components Packages in /Packages

• Packages/Delphi4

D4_SqlitePassDbo_Runtime.dpk	Runtime package. Should be compiled first (see the installation section)
D4_SqlitePassDbo_DesignTime.dpk	DesignTime package, used to : <ul style="list-style-type: none"> • Register and display components in Delphi IDE, • Register property editors in Delphi IDE.

• Packages/Delphi6

D6_SqlitePassDbo_Runtime.dpk	Runtime package. Should be compiled first (see the installation section)
D6_SqlitePassDbo_DesignTime.dpk	DesignTime package, used to : <ul style="list-style-type: none"> • Register and display components in Delphi IDE, • Register property editors in Delphi IDE.

• Packages/Delphi7

D7_SqlitePassDbo_Runtime.dpk	Runtime package. Should be compiled first (see the installation section)
D7_SqlitePassDbo_DesignTime.dpk	DesignTime package, used to : <ul style="list-style-type: none"> • Register and display components in Delphi IDE, • Register property editors in Delphi IDE.

• Packages/Lazarus

Laz_SqlitePassDbo_Runtime.dpk	Runtime package. Should be compiled first (see the installation section)
Laz_SqlitePassDbo_DesignTime.dpk	DesignTime package, used to : <ul style="list-style-type: none">• Register and display components in Lazarus IDE,• Register property editors in Lazarus IDE.

Components sources files in /Sources

/Sources : Components source files. Sources files are shared by Delphi and Lazarus

SqlitePassApi_v3.pas	Loads the external Sqlite engine library (sqlitepass3.dll or libsqlitepass3.so) and registers the sqlite functions to be used by the SqlitePass components.
SqlitePassDbo.inc	Include file used to define compiler settings...etc
SqlitePassDbo.pas	SqlitePass Database Objects interfaces definitions. Implementations are stored in SlitePassDatabase.inc, SqlitePassDatabaseParts.inc, SqlitePassDataset.inc, SqlitePassRecordset.inc, SqlitePassSelectStmt.inc, SqlitePassEngine.pas.
SqlitePassDbo.lrs	Lazarus ressource file used to display components icons in IDE.
SqlitePassConst.pas	Constant definitions
SqlitePassErrorLang.pas	Ressource strings for language support. Backup this file and translate the strings in your own language . Replace the SqlitePassErrorLang.pas with your own file. Compile and overwrite any existing runtime package. (see installation section). Only English file is available so far. A French one will be done shortly. Translators are welcome !
SqlitePassKexiDef.pas	Kexi specific constants definitions
SqlitePassUtils.pas	Implements a TObjectList (The code is just a copy from FCL) and other objects, mainly to provide code compatibility between Delphi 4 and Lazarus.

- **/Sources/DesignTimeEditors : Dialogs boxes to help you during design process in IDE.**
- *.pas files are shared by Delphi and Lazarus projects
- *.dfm files are used by Delphi
- *.lfm and *.lrs files are used by Lazarus

RegisterSqlitePassDbo.pas	Registers and displays components and property editors in IDE
SqlitePassChooseDatasetDialog.pas	Used by TsqlitePassDataset.Dataset property. Displays an treeview of available dataset in the current database.
SqlitePassIndexesDialog.pas	Used by TsqlitepassDataset.Indexes property. Displays a dialog to manage tables indexes.
SqlitePassCustomFieldDefsDialog.pas	Used by TsqlitePassDatabase.DataTypeOptions.CustomFieldDefs property. Display a dialog to manage custom field definitions
SqlitePassFieldDefsDialog.pas	Used by TsqlitepassDataset.FieldDefsInfo property. Display a dialog showing field definitions of the currently selected dataset.
SqlitePassSortByDialog.pas	Used by TsqlitepassDataset.SortedBy property. Displays a dialog to select fields to be sorted .
SqlitePassDataTypesDialog.pas	Used by TsqlitePassDatabase.DataTypeOptions.TranslationRules property. Display a dialog to manage TranslationRules.
SqlitePassMasterDetailFieldsDialog.pas	Used by TsqlitepassDataset.MasterFields property. Displays a dialog to link Master and Detail fields.
SqlitePassDesignErrorLang.pas	Defines DesignTime error messages.
CreateNewIndex.pas	Used by the indexes dialog when creating new table index.
Renameltem.pas	Used by the indexes dialog when renaming a existing table index.

Packages Installation

- **Download the last stable SQLite library** from sqlite.org or use the file provided with the SqlitePass package (..\SQLitePass_x.xx\SqliteLibrary). Decompress and install on your system.

This file (sqlite3.dll or sqlite3.so) should be placed in a system directory, ..\Windows\system32 for example.

From version 0.28, SqlitePass uses a special Sqlite library version (sqlitepass3.dll or libsqlitepass3.so) as the default sqlite library (this library exports the following functions to get schema information from an sql statement :

- `sqlite3_column_database_name`
- `sqlite3_column_database_name16`
- `sqlite3_column_table_name`
- `sqlite3_column_table_name16`
- `sqlite3_column_origin_name`
- `sqlite3_column_origin_name16`
- `sqlite3_table_column_metadata`

The compiled win32 (sqlitepass3.dll) and the Linux-i386 (libsqlitepass3.so) versions of this library are included in ..\SQLitePass\SqliteLibrary. A tutorial on how to compile the library is available on the SqlitePass website.

- **Delphi users** (*Delphi 4 example. Should apply for further versions*) :

- **Uninstall any previous version of SqlitePassDbo components :**

Choose [Components] [Install Packages] from IDE menu.

Select the SqlitePass package and click Remove.

- **Compile Runtime package :**

Choose [File] [Open] from the IDE menu and select

..\SQLitePass\Packages\Delphi4\D4_SqlitePassDbo_Runtime.dpk. Compile this package and move the resulting file from

SQLitePass_vX.XX\Packages\Delphi4\D4_SqlitePassDbo_Runtime.bpl to a directory included in the Delphi search path (like ..\Delphi4\Bin or ..\Windows\system32 for example). Delete or overwrite any previous D4_SqlitePassDbo_Runtime.bpl.

- **Compile and install Designtime package :**

Choose [File] [Open] from the IDE menu and select

..\SQLitePass\Packages\Delphi4\D4_SqlitePassDbo_Designtime.dpk. Compile this package, then choose install.

- **Check installation :**

Select SqlitePassDbo on component palette pages and drop a SqlitePassDatabase and a SqlitePassDataset on a new form. Check the components versions are correct in object inspector.

• **Lazarus users (>=0.9.20) :**

• **Uninstall any previous version of SqlitePassDbo components :**

Choose [Components] [Configure Installed Packages] from IDE menu
Select the Laz_SqlitePassDbo_Runtime and Laz_SqlitePassDbo_DesignTime packages and click [Uninstall the selection] then [Save and quit the dialog].

• **Compile Runtime package :**

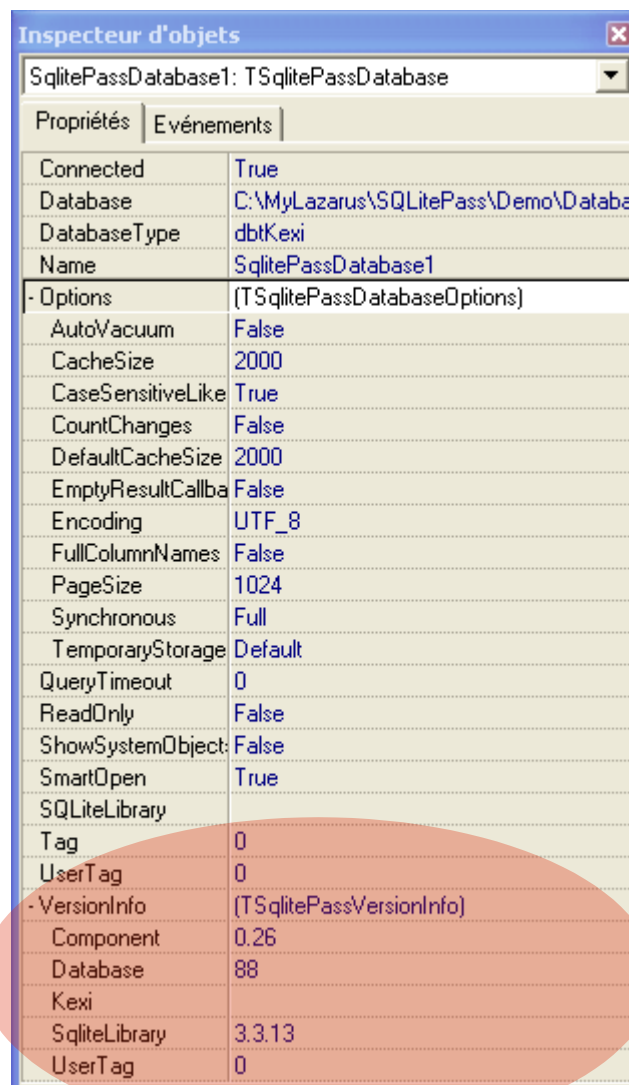
Choose [Components] [Open Package file] from IDE menu and select ..\SQLitePass\Packages\Lazarus\Laz_SqlitePassDbo_Runtime.lpk. Compile this package.

• **Compile and install DesignTime package :**

Choose [File] [Open] from the IDE menu and select ..\SQLitePass\Packages\Lazarus\Laz_SqlitePassDbo_DesignTime.lpk. In package dialog, Choose compile then install. This will rebuild the IDE.

• **Check installation :**

Select SqlitePassDbo on Component palette pages and drop a SqlitePassDatabase and a SqlitePassDataset on a new form. Check the components versions are correct in object inspector.



Quick Start

A demo application is available in `..\SQLitePass_x.xx\Demo\Sources`. This application was first developed as a simple test program for the `SQLitePass` components. Due to many changes during development process and to avoid problems when changing properties behavior in IDE, the `SQLitePassDatabase` and `SQLitePassDataset` components are directly created by code at runtime.

This application tests also the design-time dialogs boxes used by the IDE Object Inspector. To do so, it links directly to `*.pas` | `*.dfm` | `*.lfm` files stored in `..\SQLitePass_x.xx\Sources\DesignTimeEditors` directory.

The source code is (or will be) self documented and should cover the basic usage of the components.

Another way is to describe a simple application using `SQLitePass` component.

It could look like this :

1. Place a `SQLitePassDatabase` component on a form,
2. Choose Database file the Database property (a database `*.kexi` file from `..\SQLitePass_x.xx\Demo\Databases` for example),
3. Set Connected property to True,
4. Place a `SQLitePassDataset` component on the form
5. Set Database property to the name of `SQLitePassDatabase` component (by default this is `SQLitePassDatabase1`),
6. Select a dataset from the DatasetName property dialog editor,
7. Set Active property to True.
8. Place a standard Delphi or Lazarus `DataSource` component on the page,
9. Set DataSet property to the name of the `SQLitePassDataset` component (by default this is `SQLitePassDataset1`)
10. Place any DBAware component like `DBGrid`, `DBNavigator`...etc, on the form.
11. Set `DBGrid`'s, `DBNavigator`...etc, `DataSource` property to the name of `DataSource` component (by default this is `DataSource1`)
12. This is it !

The TSqlitePassDatabase component

The TsqlitePassDatabase component is the main link between your application and the sqlite library. It currently supports sqlite engine version 3.xx

PROPERTIES

- [AttachDatabase](#)
- [Close](#)
- Compact
- [Connected](#)
- Database
- ➔ Databases
- DatabaseType
- ➔ Datasets
- ➔ DatatypeOptions
- DetachDatabase
- ➔ Engine
- ➔ IndexDefs
- IsSystemTable
- [Open](#)
- ➔ Options
- ➔ QueryDefs
- QueryTimeout
- ReadOnly
- RefreshDefinitions
- ShowSystemObjects
- SmartOpen
- SQLiteLibrary
- ➔ TableDefs
- ➔ [Transaction](#)
- ➔ Translator
- ➔ Triggers
- UserTag
- ➔ VersionInfo
- ➔ Views

EVENTS

- OnAfterConnect
- OnAfterDisconnect
- OnBeforeConnect
- OnBeforeDisconnect
- OnDataTypeConversion



How to select a database

Property Database: String

Represents the physical database file you want to connect. At design time, shows up a file selection dialog box.

Opening and closing a database

Property Connected: Boolean

Set this property to True to connect the database defined in the [database property](#).

Set it to False to disconnect the database and all the datasets associated with it.

You can also verify if a database is connected using :

```
if MyDatabase.Connected  
then...
```

Property Open

Same as Connected := True

Property Close

Same as Connected := False

How to use an alternative sqlite library

Property SqliteLibrary: String

Represents an alternative library file to be used instead of the default one.

By default, sqlitepassDatabase tries to use the sqlitepass3.dll or libsqlitepass3.so file located in the system path directory (..\windows\system32\ or ..\user\lib for example).

Enter a complete library file path to use a different library.

From version 0.28 TsqlitePassDatabase needs the sqlite library compiled with the

ENABLE_METADATA precompiler directive. These libraries are available from

<http://source.online.free.fr> or you can compile your own following the tutorial available on the same internet site.

Property DatabaseType: String

Represents the database type detected from the database file extension you are using. (*.kexi for kexi database for example).

Once the database type is recognised, the TsqlitePassDatabase component sets this property and creates an internal translator to take care of the database specifications.

You can also define this property by yourself if the file extension doesn't match the correct database type.



Working with fields.datatypes

One of the main difficulties when working with Sqlite databases is to detect and translate properly the fields.datatypes since sqlite datatype are not formally defined.

SqlitePass implements several ways to define a datatype

The **Database.DatatypeOptions** property gives you the opportunity to set custom or default behaviors for a given database and lets you also define how fields.datatype will be retrieved and translated into pascal datatypes using [TranslationRules](#) and **CustomFieldDefs**.

Database.DatatypeOptions Properties

- ApplyCustomFieldDefs
- SetDefaultValues
- LoadFromDatabase
- SaveToDatabase
- BooleanExtension
- ➔ [CustomFieldDefs](#)
- DefaultFieldType
- ➔ [DetectionMode](#)
- DateSeparator
- DateStorage
- DecimalSeparator
- LoadOptions
- LongTimeFormat
- ShortDateFormat
- SaveOptions
- TimeSeparator
- ➔ [TranslationRules](#)

Property DetectionMode: **TsqlitePassDataTypeDetectionMode;**

TSqlitePassDataTypeDetectionMode = (dmTypeName, dmDbSpecific, dmCustom, dmCustomFieldDefs, dmForceStr, dmNone);

The DetectionMode controls how the TsqlitePassDatabase component will behave when it tries to retrieve the database fields definition.

It can have one of the following values :

dmTypeName

FieldTypes are retrieved from the initial SQL CREATE statement of the table for example :
'CREATE TABLE cars_names (id INTEGER PRIMARY KEY, companycode Integer, name Text(200))'

Then the database uses the [TranslationRules](#) (a collection of translation Rule) to match the datatypes names found in the database table, for example 'integer' with a pascal datatype (ftInteger in this example).

dmDbSpecific

Field.datatypes are set directly by sqlitepass, depending on the Database.Databasetype value.

Note : This can only be used with the 'dbtKexi' Databasetype.

dmCustom

Field.datatypes are first preset directly by sqlitepass, depending on the Database.Databasetype value.

If the first match fails, it uses the TranslationRules if you defined some

Finally, it fires the OnDataTypeConversion event letting you modified directly the Field.DataType Of course, the OnDataTypeConversion Event must be assigned...

dmForceStr

Converts any Datatype to ftString;

Property [TranslationRules](#): [TSqlitePassFieldTypesTranslationRules](#);

A collection of translation rules.

TranslationRules are stored in *.DFM or *.LFM file but can also be stored directly in the database

A translation rule is made of :

- A datatype name (whatever you want depending on how fields datatypes are named inside the database or the table create statement)
- A matching rule (mmExact, mmExactNoCase, mmPartial, mmPartialNoCase, mmAll);
- A resulting datatype (ft... pascal fieldtype)

The matching rule can be set for each translation rule and can be one of the following values:

mmExact

the datatype found in the database table definition must match exactly the rule datatype name. It is case sensitive.

For example if the 'Int' name is found in the database table definition then it will not match the ('Integer',mmExact,ftInteger) rule thus ftUnknown will be returned

mmExactNoCase

Same as mmExact but not case sensitive

mmPartial

the datatype found in the database table definition can partially match the rule datatype name. It is case sensitive.

For example if the 'Int' name is found in the database table definition then it will match the ('Integer',mmExact,ftInteger) rule and will return ftInteger

mmPartialNoCase

Same as mmPartial but not case sensitive

mmAll (Default)

Tries to determine the datatype using mmExact.

If it fails then another try is done using mmPartial

If the matching rule failed, the DefaultFieldType is returned (ftUnknown as default).

At design time, a specific editor lets you define the translation rules.

Property [CustomFieldDefs](#): [TSqlitePassCustomFieldDefs](#);

A collection of custom fielddefs.

A custom fielddef can be used to overwrite the fielddefs translation already done by the [DetectionMode](#) and the [TranslationRules](#) properties.

A custom fielddef is made of :

- A TableName and a FieldName to identify the field
- A FieldType: TFieldType (ftinteger, ftstring...etc)
- A FieldSize
- A FieldPrecision

This property can be very usefull when a fielddef cannot be built on the datatype. For example, two BLOB fields in a table can store text or graphic data but are declared with the same datatype in the table create statement.

At design time, a specific editor lets you define the translation rules.



How to use Transactions

Property **Transaction**: **TSqlitePassTransaction**

Sqlite engine supports only one active transaction at the same time. In other words, you can't use nested transactions. The TSqlitePassTransaction object will automatically handle this, so any attempt to start a transaction while one is running will have no effect.

Transactions are really helpful to speed up and secure data operations. You should use them as often as you can.

If no transaction is active, the TSqlitePassDatabase will always try to start a new transaction before writing to the database and commit it. This could be time consuming if you need to update or add many records at the same time and you should proceed like this :

Database.Transaction.Start;

...

Your code here to update or create records;

...

Database.Transaction.Commit;

Procedure Start

Starts a new transaction unless one is already started.

Procedure Commit

Ends the transaction and write data to the database.

Procedure Rollback

Ends the transaction and discards any change made to the database.



How to work with foreign databases

Procedure AttachDatabase: **String**;

Enables to attach one or several foreign databases to the current one. The attached databases must be compatible with the current one (the databases must have been created with the same database manager application). Once a database is attached, its content becomes available as part of the current database. Then you can access to tables, queries... as if they were part of the main database.

Usage : AttachDatabase(DatabasePath: String);

Procedure DetachDatabase

Detaches a previously attached database.



Others properties (to be documented...)

Property **Options**: **TSqlitePassDatabaseOptions**

Represents the database optional settings.

Property **TableDefs**: **TSqlitePassTableDefs**

The TableDefs property is a collection that gives you access to the tables definitions stored in the database.

Property `QueryDefs`: `TSqlitePassQueryDefs`

The `QueryDefs` property is a collection that gives you access to the queries definitions stored in the database.

Property `IndexDefs`: `TSqlitePassIndexDefs`

The `IndexDefs` property is a collection that gives you access to the indexes definitions stored in the database.

Property `ViewDefs`: `TSqlitePassViewDefs`

The `ViewDefs` property is a collection that gives you access to the views definitions stored in the database.

Property `TriggerDefs`: `TSqlitePassTriggerDefs`

The `TriggerDefs` property is a collection that gives you access to the Triggers definitions stored in the database.

The TSqlitePassDataset component

The TSqlitePassDataset is a link between your application and the database content.

TSqlitePassDataset enables you to access tables, queries or even to create direct SQL queries to read and write data from/to your database.

PROPERTIES

- Database
- DatabaseAutoActivate
- DatasetName
- DatasetType
- ExportMethod
- FieldDefsInfo
- FilterRecordLowerLimit
- FilterRecordUpperLimit
- IndexDefs
- LocateMoveState
- LocateRecordCount
- LocateSmartRefresh
- LookUpSmartRefresh
- MasterFields
- MasterSource
- MasterSourceAutoActivate
- ParamCheck
- ParamCount
- Params
- ProcessMessages
- ReadOnly
- Sorted
- SortedBy
- SQL
- SQLSelectStmt
- VersionInfo

METHODS

- ApplyFilter
- BookmarkValid
- CloseBlob
- Create
- CreateBlobStream
- Destroy
- GetFieldData
- Locate
- LocateFirst
- LocateLast
- LocateNext
- LocatePrior
- Lookup
- LookupFirst
- LookUpLast
- LookupNext
- LookupPrior
- Resync

Sort
EVENTS

Property **Active**

Classic dataset behavior.

When set to True, opens the dataset and displays data if dataware components are linked to the datasource.

When set to False, closes the dataset and frees the memory used to store dataset records.

Property **Datasource**

Classic dataset behavior.

Property **DatabaseAutoActivate**

When set to True, opens automatically the database if needed.

Property **Database**

Selects the TsqlitePassDatabase component you want to depend on.

Property **DatasetName: String;**

Once you are connected to a database, enters a table name or a query name. At design time, a dialog will let you choose your dataset among all the available database datasets.

Property **DatasetType**

This property is only read. It gives you information about the currently selected dataset and can be one of the following values :

dtUnkown : The dataset type could not be recognized or the DatasetName property is empty.

dtTable : The dataset is a table.

dtQuery : The dataset is a query.

dtView : The dataset is a view.

dtSqlDirect : The SQL property has been modified or you entered a new SQL query. When the SQL text is changed, the DatasetName will automatically be set to " assuming that the DatasetName and SQL text don't match anymore.

Property **SmartOpen: Boolean;**

When set to True, the TsqlitePassDataset will automatically take care of opening the database connection if the database is not connected. Set it to False to manually control the database connection.

Property **SQL**

Represents the SQL statement used to retrieve data from the database.

For tables, it will automatically be set to :

SELECT * FROM TableName; if all fields need to be retrieved from the table.

Or to

SELECT field1, field2... FROM TableName; if only some fields need to be retrieved from the table.

For queries, it will reflect the query SQL statement.

You can also directly write your own SQL statement to fit your needs or to interact directly with the database.

In this case, the datasetname property will be set to "" (empty) and the datasetType will be set to dtDirectSql.

Property **IndexDefs**

The IndexDefs property gives you access to the indexes definitions for the selected table. Indexes are only available if the DatasetType is a 'dtTable' type.

Property **MasterSource**: TDataSource

Classic table MasterSource behavior.

Property **MasterFields**: String

Classic table MasterFields behavior. At design time, a dialog will let you create or modify the relation between MasterFields and DetailFields.

A relation is defined like this : MasterFieldName=DetailFieldName

If you want set several relations, they must be separated by a ';'.

MasterFieldName1=DetailFieldName1;MasterFieldName2=DetailFieldName2

Property **Filter**: String;

Classic dataset filter behavior. The filter property takes a SQL WHERE clause but without the WHERE word at the beginning. You can also use wildcard characters as described in the Sqlite help.

Example : country = 'France'
customer like '%cur%'

Property **Filtered**: Boolean;

Determines whether or not the different filters are activated. The TsqlitePassDataset component can handle three filter levels that will be applied in this priority order :

- 1 : MasterFields/DetailFields property
- 2 : Filter property
- 3 : RecordLowerLimit/RecordUpperLimit properties

Property **FilterRecordLowerLimit**: Integer;

This is a range filter. If greater than 1, the -nth first records will not be retrieved. In other words, if FilterLowerLimit = 4, the fifth record will be the first one retrieved from the query.

Property **FilterRecordUpperLimit**: Integer;

This is a range filter.

If greater than 1, the -nth first records will be retrieved. In other words, if FilterUpperLimit = 4 and FilterLowerLimit = 0 then only the four first records will be retrieved from the query.

If lesser than 0, the -nth last records will be retrieved. In other words, if FilterUpperLimit = -9 and FilterLowerLimit = 0 then only the nine last records will be retrieved from the query.

Property **SortedBy**: String;

The SortedBy property takes a SQL ORDERBY clause but without the 'ORDER BY' expression at the beginning. At design time, a dialog will let you create or modify the sort order.

Example : 'car_names ASC, car_types DESC'.

Property **Sorted**: Boolean;

Determines whether or not the sortedBy property is activated.

Roadmap for version 0.35

Update of this user guide, code cleaning, bugfixes and data export method...
Add CacheLookUp, OnFieldGetValue OnFieldSetValue implementation...etc

TODO

Enable to bind variables values in a filter expression.