

# User's Guide for SqlitePass Database Components Version 0.30



Last Revision  
2007-09-12

SqlitePass components provides a easy and fast access to sqlite databases

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



**SQLite Administrator**  
**Database manager for  
Windows**

This project is open source, released under [LGPL license](#). Libraries and components are free and currently tested with Delphi 4 - Lazarus 0.9.22 – 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 .....	5
Quick Start.....	8
The TSqlitePassDatabase component.....	9
The TSqlitePassDataset component.....	12
Roadmap for version 0.31.....	14

# 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"><li>• *.pas files are shared by Delphi and Lazarus projects</li><li>• *.dfm files are used by Delphi</li><li>• *.lfm files are used by Lazarus</li></ul>

### • Documentation in /Documentation

<ul style="list-style-type: none"><li>• /UserGuide.odt or UserGuide..pdf</li></ul>	The file you are reading...
<ul style="list-style-type: none"><li>• /Changes.odt or Changes.pdf</li></ul>	Latest changes – Versions history

### • Components Packages in /Packages

#### • Packages/Delphi4

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

#### • Packages/Lazarus

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

## • 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 (sqlite.dll or sqlite.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.
SqlitePassDbo.lrs	Lazarus ressource file used to display components icons in IDE.
SqlitePassEngine.pas	Implements a basic sql engine used to communicate with sqlite library. Mainly used by TsqlitePassDatabase
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. ( <a href="#">see installation section</a> ). Only English file is available so far. A French one will be done shortly. Translators are welcome !
SqlitePassKexiDef.pas	Kexi specific constants definitions
SqlitePassParser.pas	Implements a basic sql parser. Provides methods to split, extract or replace a sql statement part.
SqlitePassUtils.pas	Implements a TobjectList object to provide code compatibility between Delphi 4 and Lazarus. The code is just a copy from FCL.

- /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
- \*.lrm 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.
SqlitePassISortByDialog.pas	Used by TsqlitepassDataset.SortedBy property. Displays a dialog to select fields to be sorted .
SqlitePassMasterDetailFieldsDialog.pas	Used by TsqlitepassDataset.MasterFields property. Displays a dialog to link Master and Detail fields.
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](http://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) 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 version of this library is included in ..\SQLitePass\SqliteLibrary. For Linux or other platforms users, a tutorial on how to compile the library is available on the SqlitePass website.

- **Delphi users (Delphi 4) :**

- **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\_v0.26\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.

Inspecteur d'objets	
SqlitePassDatabase1: TSqlitePassDatabase	
Propriétés   Événements	
Connected	True
Database	C:\MyLazarus\SQLitePass\Demo\Database
DatabaseType	dbtKexi
Name	SqlitePassDatabase1
- Options	(TSqlitePassDatabaseOptions)
AutoVacuum	False
CacheSize	2000
CaseSensitiveLike	True
CountChanges	False
DefaultCacheSize	2000
EmptyResultCallba	False
Encoding	UTF_8
FullColumnName	False
PageSize	1024
Synchronous	Full
TemporaryStorage	Default
QueryTimeout	0
ReadOnly	False
ShowSystemObject	False
SmartOpen	True
SQLiteLibrary	
Tag	0
UserTag	0
- VersionInfo	(TSqlitePassVersionInfo)
Component	0.26
Database	88
Kexi	
SQLiteLibrary	3.3.13
UserTag	0

## 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

**Property Connected:** Boolean

Set this property to True to connect the database selected in the [database property](#). Set it to False to disconnect the database and all the datasets associated with it.

**Property Database:** String

Represents the physical database file you want to connect.

**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 sqlitepass3.so file located in the system path directory (..\windows\system32\ 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. This library (only win32 dll version) is 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.

Once the database type is recognised, the TsqlitePassDatabase component will create an internal translator to match the database specifications.

Actually, TsqlitePassDatabase works mainly with databases created with kexi.

Because each external program, used to create sqlite databases, uses his own specifications on datatype format, data storage, queries storage ...etc, it is quite difficult to meet everyone needs.

This detection process should be changed in the following versions to enable more databases supports.

Feel free to indicate your favorite program or the one you would like to be supported.

Every contribution will be welcome.

**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 helpfull 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.

**Property Options: TSqlitePassDatabaseOptions**

Represents the database optional settings.

**Property TableDefs: TSqlitePassTableDefs**

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

**Property QueryDefs: TSqlitePassQueryDefs**

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

**Property IndexDefs: TSqlitePassIndexDefs**

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

**Property ViewDefs: TSqlitePassViewDefs**

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

**Property TriggerDefs: TSqlitePassTriggerDefs**

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

**Property AttachedDatabases:**

The AttachedDatabases is a collection that property gives you access to the currently attached 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.

## 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.

### 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 discribed 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 0.31

The following methods should be available from the 0.31 release

- Find
- FindNext
- FindPrevious
- FindFirst
- FindLast
- Locate
- Lookup
- LookupFields

### Find

The internal implementation could work as a subset of the original query by adding the searchFields and values to the WHERE clause of the original query.

Then it could scroll the subset with FindFirst, FindPrevious...

If we need to locate the record in the original subset, we retrieve the subset record primary key and then scan the original query result until we find the matching primary key.

The way we scan the original subset will depend on the sort order.

If the primary key is ordered we could use a dichotomic search to speed up operation..., otherwise a classic loop could be used.

### TODO

Enable to bind variables values in a filter expression.