Databases

From Free Pascal wiki

| Deutsch (de) | English (en) | español (es) | français (fr) | italiano (it) | português (pt) | русский (ru) | 中文（中国大陆） (zh_CN) |

This page is an introduction to the topic 'Lazarus and **databases'**. The following table provides an overview of supported databases.

Only the database components for which there are client libraries should be installed (if the database needs any client libaries), otherwise Lazarus could fail to start because of missing files. Then Lazarus must be reinstalled as uninstalling the component is not possible.

Contents

Database portal

References:

- General info
- Libraries
- Field types
- Controls
- FAQ
- SQL how-to
- Working With TSQLQuery
- In-memory database applications

Tutorials/practical articles:

- Overview
- 0 - Database set-up
- 1 - Getting started
- 2 - Editing
- 3 - Queries
- 4 - Data modules
- SQLdb Programming Reference

Databases

Advantage - MySQL - MSSQL - Postgres - Interbase - Firebird - Oracle - ODBC - Paradox - SQLite - dBASE - MS Access - Zeos

Supported databases

Database	Package name	Need client lib?	Need server?	Supported versions	Supported platforms
Advantage (http://www.advantagedatabase.com/)	TAdsDataSet	Yes	No	10.1 and greater	i386: Linux, Win32
DBase	DBFLaz	No	No	III+, IV, VII	All
TurboPower FlashFiler	FlashFiler	No	No	-	Win 32, (win64?)
In memory	memds	No	No	-	All
In memory	bufdataset	No	No	-	All
Firebird (http://www.firebirdsql.org/)	SQLdb	Yes	Depends[1]	1 - 2.5	i386: Linux, Win32
(Visual) FoxPro	DBFLaz	No	No	2.0, 2.5, 3.0 (not completely)	All
Interbase (http://www.embarcadero.com/products/interbase-smp)	SQLdb	Yes	Yes	4 - 6	i386: Linux, Win32
Microsoft SQL Server (http://www.microsoft.com/sqlserver/)	SQLdb	Yes	Yes	6-	FPC 2.6.2+. Linux, OSX, Win32, probably *BSD, probably Solaris[2]
MySQL (http://www.mysql.com/)	SQLdb	Yes	Yes	3.0 - 5.5	i386: Linux, Win32
ODBC	SQLdb	Yes	Depends	3.x [3]	i386: Linux, Win32
Oracle (http://www.oracle.com/)	SQLdb	Yes	Yes	-	-
Paradox	TParadoxDataSet	No	No	up to Table Level 7 (and up ??)	All
Paradox	TParadox	Yes	No		Win32
PostgreSQL (http://www.postgresql.org/)	SQLdb	Yes	Yes	6.6 - 8	i386: Linux, Win32
Sybase Adaptive Server Enterprise (ASE) (http://www.sybase.com/products/databasemanagement/adaptiveserverenterprise)	SQLdb	Yes	Yes	Any	Linux, OSX, Win32, probably *BSD, probably Solaris[2)]
SQLite (http://www.sqlite.org/)	SQLdb	Yes	No	sqlite3	All
SQLite (http://www.sqlite.org/)	SQLite(3)Laz	Yes	No	sqlite2,sqlite3	All
Text files	sdf	No	No	-	All

Note (1): You can use an embedded version of Firebird on Windows and Linux (possibly on macOS too), or you can connect to a Firebird server running on Windows/Unix/OSX/FreeBSD/other Firebird supported platforms

Note (2): These connectors use the FreeTDS library as a driver. The FreeTDS documentation indicates it should build on at least these platforms. Windows versions for x86 and x64 can be downloaded from e.g. [1] (ftp://ftp.freepascal.org/fpc/contrib/windows/dblib_x32.zip) and [2] (ftp://ftp.freepascal.org/fpc/contrib/windows/dblib_x64.zip)

Note (3): This version number refers to the ODBC standard, not to the version number of a driver or driver manager. There are ODBC 3.x drivers for most DBMSs.

The bindings to the database clients

If you want to use one of the databases that need client libraries, those libraries have to be installed. Not only on the computer you're programming on, but also on the computers where the application must run. Note that some databases (in particular MySQL) only work if the bindings which are compiled in the application are from the same version as those of the installed libraries. You can find out how to install those libraries (.so files on *nix systems, and .dlls on windows) on the website of the database developers. The binding units can be found in the packages/base directory in the fpc-sources. They basically consist of the client API calls like mysql_connect_database, which are completely different for each database. It is possible to write database applications using these units, but it is usually far more work and bug-sensitive than using the DB-unit Lazarus components.

Most of these bindings packages are hard-linked to the client libraries. This means that if the application is compiled with one of these units in it, the whole application can not be linked if the client libraries are not available on the workstation. This means that your program executable will not be generated if you do not have installed - for example - a MySQL client on your computer, and you are using the mysql4.pp unit in your program. If you succeed in compiling the program on a computer which has the MySQL client libraries installed, it still won't start on any other machine without the appropriate MySQL client libraries. In other words: for these databases, you need to install client libraries on your development machine, and you need to install these client libraries with your application.

To avoid such problems some of the packages are also able to link dynamically to the libraries. Before any calls to those libraries can be made, the unit has to be 'initialized'. This initialization fails if the database client isn't installed on the computer. If the program is ready using the client library, the unit has to be 'released'.

Datasets

Database use in Lazarus (or Free Pascal) is fundamentally based on the TDataset class. This represents a table or query to your application. However, like many other such fundamental classes, you don't use the TDataset class itself, you use a descendant of it. There are many of these. They provide access to different kinds of databases, such as local dbase or text files, or back-end databases such as PostgreSQL, Firebird, MySQL and so forth. Some dataset descendants link directly to database tables, while others use additional components or libraries to perform the link.

Dataset descendants, being non-visual components are (usually) part of the Free Component Library (FCL) rather than the Lazarus Component Library (LCL).

Datasets can be used both programmatically and with visual controls. A typical Lazarus database application will often use both methods. In either case, the first step is to create the TDataset descendant, initialise it to connect to the table or query you want, and open it. This can be done either in code at run time or by putting a component on your form and setting it's properties at design time. The details of this vary considerably with different TDataset descendants, so see the various guides under Databases for what has to be done for your database.

When the dataset is opened, a set of field components are created, one for each field or column of the table or query you opened. Each field component is a descendant of TField, appropriate to the particular data type of the

field, eg, TStringField.

Using datasets from code

Programmatic access will be explained in more detail in Using Dataset and Field components, but as a very simple overview:

- Use the TDataset descendant to open the table or query, filter the rows you want to see, and to move from row to row.
- Use the TField descendants to:
  - Access general information about fields
  - Access the specific data values for the current row. (use the As... properties, such as AsString, AsInteger, etc.)
- Access the fields of a TDataset descendant by using either:
  - The fields property, eg Fields[0] is the first field,
  - The FieldByName method, eg FieldByName('AGE') returns the field associated with the database field called 'AGE'

See Database_field_type for a list of field types.

Using the visual (data-aware) controls

To use databases in a simple, "RAD" style Lazarus application, you usually configure the dataset descendant at design time and the use the data-aware controls. To do this:

- Add the dataset descendant for the database of your choice, together with any supporting components, to your form, and open it (Set the ' Active' property to true)
- Add a TDataSource component (from the Data Access tab) to the form, and "link" it to the dataset (set the DataSet property)
- Add data-aware controls from the Data Controls tab to the form, and link each one to the DataSource (not dataset) component
- Most controls link to a single field, so you also need to set the Field for each tab.

See #Data Controls below for more details on the controls

Dataset State

Datasets can be in a number of states. While there are quite a few (look up TDataSetState in the source), the main ones to be aware of initally are

State	Function
dsInactive	The dataset is closed
dsBrowse	The user can browse through the dataset, looking at values
dsEdit	The user can edit values on the current row.Values are not saved until a post is performed.
dsInsert	A new row has been added, and the user can set the values.The record is not saved until a post is performed

The other states are fairly transitory, and are usually handled "automatically". They are used internally and in more complicated code. If your database only views data, and you open the dataset at design time, you can

largely ignore the state, as it will mostly be dsBrowse. However, most applications will want to change the data at some stage. If you are using data-aware controls, they will handle a lot of this automatically. If you change the text in a TDBEdit control, for example, it will put the dataset into dsEdit state - unless you are already in dsEdit or dsInsert. If you "scroll" to a different record while the current state is dsEdit or dsInsert, that record will be "posted" and the dataset revert to dsBrowse. However, if you are accessing the dataset from code, you will often have to change the state in code as well. The TDBNavigator control (see below) allows the user to change the state explicitly.

Dataset UpdateStatus

UpdateStatus determines the current state of the record buffer, if updates have not yet been applied to the database.

Example how to detect if ApplyUpdates will Insert, Update or Delete data:

```
procedure QueryAfterPost(DataSet: TDataSet);
begin
  case DataSet.UpdateStatus of
    usUnmodified : ShowMessage('Unmodified');
    usModified   : ShowMessage('Modified');
    usInserted   : ShowMessage('Inserted');
    usDeleted    : ShowMessage('Deleted');
  end;
end;
```

Value Explanation

- usUnmodified: Record is unmodified
- usModified: Record exists in the database but is locally modified
- usInserted: Record does not yet exist in the database, but is locally inserted
- usDeleted: Record exists in the database, but is locally deleted

Post and Cancel

If you have edited or inserted a record, the new values are held in a buffer.

- Calling the dataset cancel method removes the new record (insert) or reverts the values to their previous values (edit).
- Calling the dataset post method saves the values (edit) or record (insert). In some dataset descendants, they will be written to the database immediately, while in others they will be stored in a list of updates until a further call is made to save all changes to the database. Finally, even when they are written to the database, you may still have to call a "commit" method to make the database write them permanently. All of this also varies considerably with the dataset descendant, so look up the details for the one you are using.

Inserting a new record

To insert a new record into a TDataset descendent, one should use the method Insert (http://www.freepascal.org/docs-html/fcl/db/tdataset.insert.html). After that one can set the field values and then finally call Post to commit the new record, as the example below shows.

The example also shows how to insert BLOB data from a file - you can also use LoadFromStream to load the data from a stream.

```
  MyDataset.Insert;
  MyDataset.Fields[0].AsInteger := 4; //an integer field
  MyDataset.Fields[1].AsString := 'First Name'; //a string field
  (MyDataset.Fields[2] as TBlobField).LoadFromFile('SomeBlobfile.bin'); //blob field
  MyDataset.Post;
```

How to quickly jump to 1 particular record in the table

After selecting all records of the table

If you use SELECT * FROM to select all records of the table and then desires to quickly jump between them, you will have to build an index and search on it. It is more efficient to select only the record that you want.

Selecting only the desired record

One fast solution to jump to a particular record is to select only it, for example doing:

```
var
  MyDataset: TSQLQuery;
begin
  //...
  MyDataset.FieldDefs.Add('SessionId', ftLargeint);
  MyDataset.FieldDefs.Add('GameEvent', ftLargeint);
  MyDataset.FieldDefs.Add('TableId', ftInteger);
  MyDataset.FieldDefs.Add('LoggedIn', ftBoolean);
  MyDataset.FieldDefs.Add('PlayerId', ftInteger);
  MyDataset.Active := False;
  { Non-parameterized format; may run into issues with text containing ' and dates
  SQLText := Format('select * from "GameSession" WHERE "SessionId"=%d', [ASessionId]);
  }
  // Solution: parameterized query:
  // Actually, if this is done in a loop, you only need to set the SQL.Text once,
  // and only change the parameter value
  MyDataset.SQL.Text := 'select * from "GameSession" WHERE "SessionID"=:SessionID');
  MyDataSet.ParamByName('SessionID').AsLargeInt := ASessionID;
  try
    MyDataset.Active := True;
  except
    //...
  end;
```

You can then read information using something like this:

```
lPlayerId := MyDataset.Fields[4].AsInteger;
```

Filtering

You can filter your dataset to restrict the records to a subset you want (e.g. all surnames starting with Smith).

- Using .Filter:
  - TDbf, TBufDataset and descendants (including TSQLQuery) use the TDBF filtering parser; see Lazarus Tdbf Tutorial#Expressions for details on using this.
  - TMemDataset does not support .Filter
- Using a callback/event procedure with OnFilter where you can program your own filter function

Locate/lookup

Although more often used in non-SQL datasets (e.g. TParadoxDataSet, TDbf) you can jump between records using locate/lookup.

Using TSQLQuery

For more information about TSQLQuery see Working With TSQLQuery

Exporting

FPC/Lazarus contains functionality to let you export datasets to various formats; see e.g.

- fpXMLXSDExport
- fpdbfexport
- the other components on the Data Export tab

Of course, you could also do it manually (see e.g. FPSpreadsheet#Converting_a_database_to_a_spreadsheet for export to Excel format using fpspreadsheet)

Data Controls

To use any of these controls, add the control to a form and set at least the datasource property. Other key properties will be noted.

Datasource Control

This control keeps track of which record the linked controls currently are on. The datasource contorl must be linked to a dataset (e.g. a TSQLQuery).

Single Field Controls

These controls all attach to a single field. As well as datasource, set the field name. Controls include:

- DBText control Displays a text field (readonly, no border)
- DBEdit control Displays / edits a text field as an edit box
- DBMemo control Displays / edits a text field in a multi-line edit box
- DBImage control Displays a picture stored in a database as a BLOB. Note: by default, Lazarus stores a header with the image type before the image data in the database BLOB field. This is different from Delphi. However, you can make TDBImage Delphi compatible: see Lazarus For Delphi Users#TDBImage
- DBListBox control and DBComboBox Control Allow the user to insert values into a database field from the list of values in the Items property of the controls
- DBLookupListBox control and DBLookupComboBox control, see also TDBLookupComboBox Allow the user to insert values into a database field by displaying the contents of a lookup field in another table. Though these controls store their results in a single field, you need another field for the lookup values. **Note**: at lest for DBLookupComboBox, there is a bug with FPC 2.6.0 that requires the listfield to be present in the datasource as well, you can bypass it by declaring a calculated field with the same name as the listfield in the datasource's dataset that does nothing.
- DBCheckBox control Displays / edits a boolean field by checking/clearing a check box
- DBRadioGroup control Displays the items as in a normal radio group, reading/setting the field value from a matching values list

- DBCalendar control Displays / edits a date field using a calendar panel
- DBGroupBox control

DBGrid control

This control can show a number of fields in a row/column layout - in fact by default it shows them all. However, you can put entries into the columns collection to restrict it to specific fields and to set the widths and titles of each column.

Apart from the mentioned documentation, some more detail can be found here: Grids Reference Page#TCustomDBGrid

Navigator Control (http://lazarus-ccr.sourceforge.net/docs/lcl/dbctrls/tdbnavigator.html)

This control gives the user some direct control over the dataset. It allows the user to:

- Move to the next or previous record, or to the start or end of the records
- Add a new record (equivalent to a dataset.insert method call)
- Put the dataset into edit mode
- Delete a record
- Post or Cancel current changes
- Refresh the data (useful in multiuser database applications)

Key Properties:

- VisibleButtons: Lets you control what the user can do. For example, if deletes are not allowed, hide the delete button. If you have a DBGrid attached to the same dataset, you may decide you do not need the next and prior buttons.
- Width: If you do not show all buttons, you may want to set the width to (height*number_of_visible_buttons)

Running FPC database tests

Free Pascal database components include a fpcunit-based test framework, dbtestframework, that can be used to verify functionality. See the directory source\packages\fcl-db\tests\ in your FPC source tree. Included is a test framework that can be run on various database components, as well as some other tests (e.g. test of database export).

To run the test framework on a certain database:

1. Save source\packages\fcl-db\tests\database.ini.txt as source\packages\fcl-db\tests\database.ini

2. Modify source\packages\fcl-db\tests\database.ini to choose which database type you will use.

Example for Interbase/Firebird:

```
[Database]
type=interbase
```

3. In the same file, customize settings for your database. E.g. if you chose interbase before:

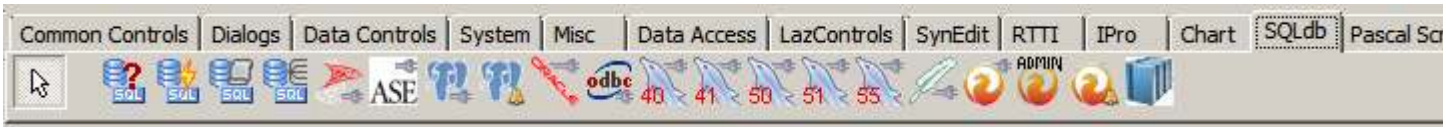
```
[interbase]
connector=sql
connectorparams=interbase
; Database name/path (note: database needs to exist already)
; You can use aliases (see aliases.conf in your Firebird documentation)
name=testdb
user=sysdba
password=masterkey
; your hostname may very well differ:
; Leave blank if you want to use an embedded Firebird database
hostname=192.168.0.42
```

4. Compile and run source\packages\fcl-db\tests\dbtestframework.pas (You can also use Lazarus to compile and run the GUI version, dbtestframework_gui) If you are using an embedded database on Windows (e.g. Firebird embedded or sqlite), copy the required DLL files to the directory first. The output will be in XML format (or displayed on your screen if you use dbtestframework_gui).

Please see source\packages\fcl-db\tests\README.txt for more details.

Database packages contained in Lazarus

sqldblaz.lpk



This package provides access to different databases. These include:

- Interbase/Firebird
- Microsoft SQL Server (except on Lazarus/FPC x64 for Windows)
- MySQL
- Oracle (except on Lazarus/FPC x64 for Windows)
- PostgreSQL (except on Lazarus/FPC x64 for Windows)
- SQLite (with support for the Spatialite extension)
- Sybase ASE (Adaptive Server Enterprise - not to be confused with Sybase ASA) (except on Lazarus/FPC x64 for Windows)
- any database that has an ODBC driver.

The components (TSQLQuery, TSQLTransaction, TIBConnection, TODBCConnection, TOracleConnection, TMSSQLConnection, TMySQL40Connection, TMySQL41Connection, TMySQL50Connection, TPQConnection, TSybaseConnection) are on the 'SQLdb' tab in the component palette.

- SQLdb Package

dbflaz.lpk

This package provides access to dBase and FoxPro databases. You can get more information in the Lazarus Tdbf Tutorial. The TDbf component is on the Data Access tab in the component palette.

sqlitelaz.lpk

This package provides access to SQLite databases. You can get more information in the Lazarus Database

Overview.

sdflaz.lpk

The component TSdfDataSet can be found on the Data Access tab in the component palette.

lazreport.lpk

The homepage of the report generator is http://lazreport.sourceforge.net/ (http://lazreport.sourceforge.net/). More informationen (et al. an additional link) can be found here. LazReport depends on the Printer4Lazarus package. With revision 11950 LazReport was included in the Lazarus SVN repository.

lazdbexport.lpk

See lazdbexport.

External packages / libraries

Zeos DataBase Objects

These components provide access to different databases. You can find more information here. This wiki also contains a Zeos tutorial.

Pascal Data Objects

There is now an alternative.

Support:

- MySQL 4.1 and 5.0
- sqlite-2 and sqlite-3
- pgsql-8.1
- interbase-5, interbase-6, firebird-1.0, firebird-1.5, firebird-1.5E, firebird-2.0, firebird-2.0E
- mssql (Microsoft library) and sybase (FreeTDS library)
- oracle

like prepared statements, binding, and stored procedures are supported by database API called Pascal Data Objects, which is inspired by PHP Data Objects. All the code and documentation necessary to use this new API is available on Sourceforge:

http://pdo.sourceforge.net

TPSQL

These components provide access via TCP/IP to PostgreSQL databases. You can find more information on this page.

FIBL

These components provide access to Interbase and Firebird databases. The homepage is http://sourceforge.net /projects/fibl (http://sourceforge.net/projects/fibl).

IBX

IBX For Lazarus are components to access Firebird databases: see IBX

FBLib Firebird Library

FBLib (http://fblib.altervista.org/) is an open Source Library No Data Aware for direct access to Firebird Relational Database from Borland Delphi/Kylix, Free Pascal and Lazarus.

Current Features include:

- Direct Access to Firebird 1.0.x, 1.5.x and 2.x Classic or SuperServer
- Multiplatform [Win32,Gnu/Linux,FreeBSD)
- Automatic select client library 'fbclient' or 'gds32'
- Query with params
- Support SQL Dialect 1/3
- LGPL License agreement
- Extract Metadata
- Simple Script Parser
- Only 100-150 KB added into final EXE
- Support BLOB Fields
- Export Data to HTML SQL Script
- Service manager (backup,restore,gfix...)
- Events Alerter

You can download documentation on FBLib's website (http://fblib.altervista.org/).

Unified Interbase

UIB provides access to Interbase, Firebird and YAFFIL databases. The homepage is www.progdigy.com (http:// www.progdigy.com). A svn repository is available under https://uib.svn.sourceforge.net/svnroot/uib .

TechInsite Object Persistence Framework (tiOPF)

More information about tiOPF can be found on this page.

Advantage TDataSet Descendant

The Advantage TDataSet Descedant provides a means of connecting to (and opening tables with) the Advantage Database Server. Advantage is a flexible, administration-free embedded database that provides Client/Server as well as Peer-to-peer access to Clipper, FoxPro and Visual FoxPro 9 DBF file formats, as well as a proprietary file format that provides a migration path allowing the use of newer features.

Key Features:

- Royalty-free peer-to-peer database access with migration path to Client/Server
- Multi-Platform (Clients supported on Windows and Linux, Server supported on Windows, Linux, and

NetWare)
- Supports Both navigational and relational SQL database access
- Full-text search engine
- Table, Index, Memo, and communication encryption
- Compatible with native TDataset components
- Online Backup
- Server supports Replication

For more information, see the Advantage Database Server (http://www.advantagedatabase.com) website.

ZMSQL, sql-enhanced in-memory database

For more information, see the ZMSQL wiki page (http://wiki.lazarus.freepascal.org/ZMSQL)

ZMSQL is an open source, TBufDataset descendant SQL enhanced in-memory database for Free Pascal (FPC), operating with semicolon-separated values flat text tables. Completely written in Pascal, it has no dependencies on external libraries. It uses JanSQL engine for SQL implementation.

It offers:

- Loading from and saving to flat text tables
- Use of SQL to query the data
- Copy data and schema from other datasets
- Option to predefine fielddefs or create it on-the fly
- Master/detail filtering
- Referential integrity
- Parameterized queries

The download contains the source code, some demo applications illustrating the features of the component as well as a readme.

See also

- Database Portal
- Lazarus DB Faq