

## IBX For Lazarus (Firebird Express)

p { margin-bottom: 0.08in; }h2 { margin-bottom: 0.08in; }h2.western { font-family: "Arial",sans-serif; font-size: 14pt; font-style: italic; }h2.cjk { font-size: 14pt; font-style: italic; }h2.cjl { font-size: 14pt; font-style: italic; }h3 { margin-bottom: 0.08in; }h3.western { font-family: "Arial",sans-serif; }h3.cjk { font-family: "DejaVu Sans"; }h3.cjl { font-family: "DejaVu Sans"; }a:link { }

IBX for Lazarus is derived from the Open Source edition of IBX published by Borland/Inprise in 2000 under the InterBase Public License. This version has been brought up-to-date by MWA Software and focused on the Firebird Database API for both Linux and Windows platforms. It is released under the InterBase Public License for the original code and under the compatible Initial Developers Public License for new software.

Positive reports have also been received about its use on MAC OSX.

Download:

- IBX for Lazarus 1.0-4 (tar.gz)

- IBX for Lazarus 1.0-4 (zip)

See also: [why IBX](#)

While the core of the product remains the original IBX software, this version includes a completely new set of property editors supporting SQL generation and testing using the Firebird Database engine direct from the IDE. These are intended to be a significant improvement on the Delphi Property Editors. IBSQLMonitor has also been re-organised in order to isolate the platform dependent aspects, allowing for the use of SV5 IPC for the Linux environment. The original Windows IPC is retained for the Windows environment. IBEvents has also been updated to ensure compatibility with Firebird Events.

The Database Interface unit (IBIntf) is now firmly focused on the Firebird 2 API for both 32-bit and 64-bit environments.

Support for generators has also been added compatible with the generator support added to IBX after the Open Source edition was published, supporting both "On New Record" and "On Post" generators.

IBX for Lazarus has been developed in the context of a single project porting a large application originally developed in Delphi to Lazarus. Testing has focused on this application and the Linux 64-bit environment (Ubuntu 10.04), with product testing on 32-bit Linux and Windows XP.

You are encouraged to download the software and report your experience to us - both good and bad so that a stable product can be made available to the Open Source Community. Please report your experiences to [ibx@mwasoftware.co.uk](mailto:ibx@mwasoftware.co.uk).

### Minimum Requirements

Lazarus version 0.9.30 and version 2.4.2 of the Free Pascal Compiler.

The Firebird 2.x client library must also be installed.

### Installation

The

Firebird Client Library should be installed on the system prior to installing into the Lazarus IDE. See the separate notes for using Firebird and IBX for Lazarus under Linux and Windows, respectively.

### Installation

into the Lazarus IDE is the same under both Linux and Windows. Unpack the source code archive into some suitable permanent location and open the "dclibx.lpk" package description file using the "Package->Open Package File" menu item to open the file.

When

the Package Editor opens, click on "Use->Install". Lazarus should now recompile itself and restart. Two new tabs should now be present on the Component Palette: "Firebird" and "Firebird Admin". Respectively, these contain the IBX Database Access and Service API components.

If

no IBX components are visible, then the most likely reason is that the Firebird Client Library has not been installed and/or cannot be located.

Using the IBX Components

The

IBX components should behave identically to their Delphi equivalents and many online tutorials are available on how to use them. Some notes on their use are given below, and an example program is also provided.

You should be aware of the following issues:

-  
The  
IBX components make use of the TThread component, and, as such require that multi-threading is enabled. Specifically, in the Linux environment, the "-dUseCThreads" option must be present in the "Compiler Options->Options->Custom Options" and set for every Lazarus project that uses them.

-  
Prior to FPC 2.6.0, the  
TIntegerField type may cause problems when porting code from Delphi to Lazarus.

-  
FMTBcd  
is not yet implemented by the Free Pascal Compiler. IBX for Lazarus thus uses the TFloatField type for extended floating point (64 bit) fields. This may cause problems where converting Delphi programs to Lazarus. The recommended approach is to change all TFmtBcdField types to TIBBcdFields. This will allow delphi forms to be converted to Lazarus. However, some of the conversions will not give the correct results. Typically, this will result in field values that appear to be of the order of several billion when the program is run. To resolve the problem, delete the field in the IDE Fields editor and then re-create it. The correct field type will then be used.  
Alternatively, all TFmtBcdField fields should be deleted prior to conversion and then recreated in the IDE.

The IBX Components

The

purpose of IBX is to provide an implementation of the Delphi/Lazarus TDataset model, and hence a data source for Data Aware components, and doing so by making direct use of the Firebird API. There is no middleware involved and the intent is to maximise performance.

Firebird

is an SQL database and a knowledge of SQL is generally necessary for all but basic use of IBX. IBX does not attempt to hide the SQL from the programmer. Indeed, it gives the programmer full use of SQL.

The following components are installed on the Firebird tab:

TIBDatabase

Every

project that uses IBX must have at least one TIBDatabase component.

This is usually placed on a data module or the main form, and represents

the connection to the database. Its properties identify the server on which the database is located, its name or pathname on that server, the login credentials, and the local character set when transliteration is required. It can also generate a login prompt for the user name and password, or support a user provided login form.

#### TIBTransaction

Every

project that uses IBX must have at least one TIBTransaction component. Firebird is a transaction oriented database and all operations must take place in the context of a transaction. Its properties determine the transaction isolation (see Firebird documentation). A TIBTransaction is typically provided with the TIBDatabase and linked to it by the TIBDatabase DefaultTransaction property.

#### TIBQuery

This

component is a descendent of TDataSet and generates the dataset from the results of an SQL query (Select statement or a Stored Procedure that returns a results set). The SQL query used is given by its SQL property. This can be parameterised (see below) with the values of the parameters set before the query is executed. When the "active" property is set to true then the query is executed and the results set returned.

When "active" is set to false, the results set is discarded.

Its properties must identify the database and the transaction used for executing the query.

#### TIBUpdateSQL

This

component can be linked from one or more TIBQuery components and is used to support updateable queries. It provides SQL statements to:

- 
- .Delete the current row in the results set
- 
- .Refresh (from the database) the current row in the results set
- 
- .Update the current row in the database to match the (modified) values in the database
- 
- .Insert a new row in the database.

#### TIBDataSet

This is also a TDataSet descendent and combines the functionality of TIBQuery and TIBUpdateSQL into a single component.

Its properties must identify the database and the transaction used for executing the query.

You

will normally want to use TIBDataset instead of a TIBQuery and TIBUpdateSQL pair. The main use of the latter combination is, for example, when a form uses a TIBQuery to provide a read only dataset, and a subclassed (inherited) form needs to update the dataset. The TIBUpdateSQL can be added to the subclassed form to provide the update capability.

#### TIBStoredProc

This component is used to execute a stored procedure (on the Database Server). Usually one that does not generate a results set.

Its properties must identify the database and the transaction used for executing the query.

#### TIBSQL

This

component is the basic SQL engine of IBX and is used internally by TIBQuery, TIBDataset and TIBStoredProc to perform SQL queries. It can be used directly by the programmer to effectively implement embedded SQL statements.

Its properties must identify the database and the transaction used for executing the query.

TIBSQL is essentially an object oriented encapsulation of the Firebird DSQL API.

#### TIBEvents

One

very useful feature of the Firebird Database is its ability to generate asynchronous "events" from a Stored Procedure or Trigger and which can then be acted upon by any active client that is listening on the event. Database clients can thus act immediately on changes made by another client without needing to regularly query the database.

The

TIBEvents component is used to register for and receive Firebird Events. Up to 16 events can be waited upon simultaneously. The name of each event to be listened to is set in the component's Events property. If you need to wait on more than 16 events, then additional TIBEvents components can be used.

The event notification is

asynchronous and takes place at the end of the transaction in which the event was generated. A separate thread is created by TIBEvents to wait on the event notification. When the event occurs it calls the "OnEventAlert" event handler to report which event has been received. Note that the event handler is run in the context of the main thread and hence there is no need to worry about thread synchronisation.

TIBSQLMonitor

This

component supports debugging and performance tuning by allowing one process to monitor the SQL function calls in the same or another process (on the same system).

The TIBDatabase trace flags

determine which function calls can be traced with respect to its database connection. However, a process only starts to broadcast its function calls after an explicit call to the IBSQLMonitor.EnableMonitoring procedure, and stops after a call to IBSQLMonitor.DisableMonitoring.

To

receive SQL Function call traces, you need to place a TIBSQLMonitor component on your form. The properties of this component can be set to filter the SQL function calls to what you are interested in. The OnSQL event handler is used to receive and process SQL function call trace events.

Note that the Windows implementation allows any process to monitor the SQL Trace events broadcast by another. The Linux implementation restricts monitoring to processes owned by the same user.

TIBTable

This

component provides a simple TDataSet descendent where the contents of the dataset are the same as a named Database Table. This component is useful for very simple novice applications, but TIBDataset should normally be preferred.

TIBExtract

This

component allows the extract of database metadata. It is part of the original IBX and has been retained for compatibility. However, the Firebird isql utility should normally be used for this purpose. No attempt has been made to bring this component up-to-date with respect to enhancements made to the Firebird SQL syntax.

TIBBatchMove

This component supports a table to table copy from a source IBX dataset to a TIBTable.

The

Firebird Admin tab provides the Service API components. These support various server side functions including user password maintenance and database backup/restore.

Using the Property Editors

IBX

for Lazarus comes with property and component editors for all IBX objects including those providing the service API. For the data access components (IBDataSet, IBQuery, IBSQLUpdate and IBSQL), the property and component editors provide assisted generation and testing of SQL statements using the Firebird Database Engine. For this to function correctly, a TIBDatabase component must be defined and linked to a live

database. A default transaction must be present and linked to the TIBDatabase and the Database property must be set for each data access component, and pointing to the TIBDatabase component.

The editors follow the same basic scheme with a combo box in the left hand frame listing available tables and listboxes showing the columns and Primary Keys respectively for the currently selected table.

A "Generate SQL" button will cause a Select, Insert, Update or Delete Statement, as appropriate, to be generated for the currently selected table, and presented into the right hand editor window. If one or more columns are selected then the statement is restricted to those columns, otherwise all columns are included in the statement. Note that Update statements may be restricted to avoid updating the primary key values. If these are internal keys that are not visible to the user, then there is little value in including them in the update statement.. Double-clicking on a column name will cause that column name to be inserted in the SQL statement.

The "Test" button can be used to test the current SQL statement for correctness. The error generated by the database engine, if any, will be returned.

In typical use, the "Generate SQL" function provides an initial set of SQL statements that can be edited to suit the actual purpose. The "Test" function can be used to check correct syntax and avoids having to compile and run the program in order to test SQL syntax correctness.

Notes on the SQL Syntax

The SQL statement syntax supported by IBX is the same as the SQL syntax supported by Firebird. Both the Data Manipulation Language (DML) and the Data Definition Language (DDL) can be used. However, there is one important difference and that is in the handling of parameterised queries.

In normal Firebird SQL, query parameters are represented by a '?' placeholder and are manipulated as positional parameters. IBX instead borrows from the Firebird Procedure and Trigger Language and uses named parameters, where a parameter name starts with a colon character and otherwise conforms with the requirements for a database column name. For example:  
 Select A.EMP\_NO, A.FIRST\_NAME, A.LAST\_NAME, From EMPLOYEE A

Where A.EMP\_NO = :EMP\_NO;

is a parameterised select query where ":EMP\_NO" is a parameter.

For a select query, the parameter value must be specified before the query is activated. For example:  
 IBQuery1.ParamByName('EMP\_NO').AsInteger := 1;

IBQuery1.Active := true;

If a TIBQuery or TIBDataset with a parameterised Select query is activated without a parameter value assigned, and the query has a DataSource property set, the component will query the Dataset addressed by DataSource. If it has a field with the same name as the parameter (leading ':' omitted) then the current value of the field is taken as the parameter value. This allows master/detail relationships to be simply established by careful choice of parameter names.

In Update, Insert, Delete and Refresh queries, a different convention is used for determining parameter values. In the case, and before the query is executed, the parameter values are taken from the values of the dataset's own fields with the same name as the parameter. For example, an Insert query thus inserts a new record with the column values set to the current field values.

Insert into EMPLOYEE(EMP\_NO, LAST\_NAME) VALUES(:EMP\_NO, :LAST\_NAME);

In the Refresh, Update and Delete queries, a Where clause is normally

given and this is usually specified to use the current values of the primary key fields, in order to select the current record. For example:  
Delete from EMPLOYEE Where EMP\_NO = :EMP\_NO;  
will delete the current record from the database.

One

further convention is also very useful for Update queries. A parameter name may be prefixed with “:OLD\_”. If so, the value it is set to is the value of the named field before any changes were made to it. This is important to allow a user to change the value of a field that is also a primary key. For example:

```
Update EMPLOYEE Set EMP_NO = :EMP_NO, LAST_NAME = :LAST_NAME
```

```
Where EMP_NO = :OLD_EMP_NO;
```

allows

change of both the employee's last name and the Employee Number. However, the Employee Number is also the primary key and it can only be changed if the original value is used in the where clause (to select the current record in the database), with the new value set in the main part of the Update statement.

Transactions

Firebird

is a transaction oriented database and all interactions with the database have to take place in the context of a transaction. At the end of a transaction, the transaction is either committed. That is all changes are made permanent, or they are rolled back and the state of the database restored to the state it was in before the transaction was started. Firebird allows multiple independent transactions to take place simultaneously and provides several possible isolation strategies in order to avoid the transactions interfering with each other.

In

a simple application, you only need to include a single TIBTransaction component with your application. The first dataset to be activated implicitly starts the transaction and, if, when it is deactivated, no other datasets are active, it will automatically commit the transaction. When the database is closed, the default action of the TIBTransaction is to commit all changes.

In a more advanced

application, you will want more control over committing or rolling back the transaction and TIBTransaction provides methods to start a transaction and to commit it or roll it back. In this case, it is usually advisable to explicitly start each transaction rather than relying on implicit starts as this avoids a dataset unexpectedly committing a transaction when it is closed. Commit and Rollback are then always under programmatic control.

Note that

when a transaction ends, all datasets referencing the transaction are automatically deactivated. A new transaction has to be started and those datasets reactivated if they are to continue to be populated.