

FreeIBComponents

Directly accessing InterBase from Delphi

Gregory H. Deatz

Copyright © 1998 Gregory H. Deatz

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

1 Introduction

FreeIBComponents and all associated source code is free! But that doesn't mean that it is public domain. In fact, it is not. See Chapter 4 [Licensing], page 11 for a detailed explanation of the license you are granted for using FreeIBComponents.

This suite of components is for easily accessing InterBase without using third-party products (although I supposed *this* is a third-party product...). This product was written entirely in Delphi, and has been designed to work with Delphi. I have heard some rumors that these components will work with C++-Builder. I don't have C++-builder, so I can't test this rumor, but it might be worth a shot!

1.1 Motivation

FreeIBComponents is a great jumping stone for all of us Delphi/InterBase developers. It frees us from using the BDE, and allows us to marry Delphi and InterBase even more tightly. Best of all, this tool comes to developers free of charge, with *all* source code included.

Why did I write FreeIBComponents? After all, there are a great many tools out there which marry InterBase and Delphi quite well. For example, Jason Wharton has a commercial product out there which avoids the use of the BDE, and the BDE itself is a wonderful product for accessing InterBase from Delphi. I could have bought Mr. Wharton's product or just continued using the BDE.

I'll give you a bit of background first. I and my company chose to use InterBase because of its superior transaction handling, its small footprint and its ease of administration. We also chose to use Delphi for client development because Delphi is a wonderful Windows development tool. The BDE has proven useful, but, among many other reasons, I have grown tired of distributing megabytes of files (that I didn't create) just to run my applications.

Because I have such faith in InterBase's product, it is only reasonable to do all I can to advocate its use, and to support it by building tools that exploit as many of InterBase's features as possible. The only way we, as a developer community, can expect other developers to share the burden for advocating InterBase is to provide free tools which will further enable all developers to produce great InterBase applications.

By getting developers to use InterBase more and more, we solidify and further justify our own decisions to use (and sell) the product.

To that end, I have created FreeIBComponents, my own little shot at benefiting the InterBase and Delphi developer community.

1.2 Acknowledgements

1.2.1 Hoagland, Longo, Moran, Dunst & Doukas

Hoagland, Longo, Moran, Dunst & Doukas is a law firm in New Brunswick, NJ. This law firm is one of the most technologically advanced law firms in New Jersey, due largely to the fact that their management team has a uniquely clear vision of the future use of technology in the legal field.

Hoagland, Longo, Moran, Dunst & Doukas is dedicated to providing quality legal services to the business community, healthcare and insurance industries. Since our founding in 1977 we have handled thousands of matters spanning the spectrum of business transactions and civil litigation including automobile negligence, commercial litigation, construction claims, coverage, D and O liability, employment practices liability, environmental claims, general liability, municipal liability, products liability, professional liability and workers compensation.

Very special thanks goes to Hoagland, Longo, Moran, Dunst & Doukas. If you ever require legal representation in the New Jersey area, you might want to look these guys up.

1.2.2 InterBase corporation

InterBase (<http://www.interbase.com>) corporation has provided their support by providing a home for both FreeIBComponents and FreeUDFLib (see Chapter 12 [About the author], page 41).

Obviously, without InterBase, FreeIBComponents wouldn't be worth a thing, so a special thanks goes to all at InterBase, with a strong vote of support to *please* continue the outstanding work!

1.2.3 Contributors and supporters

Here is a list of all who have contributed or otherwise supported the project. If your name has been omitted, it is not on purpose!

- Kenneth J. Doukas, Managing Partner of Hoagland, Longo, Moran, Dunst & Doukas.
(kenneth_j_doukas@hlmdd.com)
- Bill Karwin, Director of Publications at InterBase Corporation.
(bkarwin@interbase.com)
- Douglas Susan, Partner at Hoagland, Longo, Moran, Dunst & Doukas.
(douglas_susan@hlmdd.com)
- Mike Tossy, Director of Marketing at InterBase Corporation.
(mtossy@interbase.com)
- Karsten Strobel, (AIT-Augsburg@t-online.de)
Paul Bonnette, (pbonnette@commtech.net)
Geoff Worboys, Telesis Computing (gworboys@zeta.org.au)
Adrian Billingham, (adrian@softly.demon.co.uk)
Mark Watts, (Mark.Watts@universalsystems.com)
Pablos ?, (pablos@portal.ru)
Carlos ?, (decumont@ibm.net)
Ravikumar Ramanathan (rkumar@interbase.com)
Christian Viehbock (cviehboeck@zisser.com)
Ralf Bellmann (ralf.bellmann@jsdialog.com)
Dmitry Vodennikov (vod@zaural.ru)

1.3 Technical support

Since FreeIBComponents is free, it is important to give developers a sense of what kind of support they can expect. Although there are no guarantees of *any kind*, I want to give developers reasonable assurances of what they can, in general, expect from me.

I believe that there are four basic levels of support developers require:

1. Prompt bug resolution.
2. Thorough documentation.
3. Wish lists.
4. Standard help-desk style support. (How do I do ...? When should I do ... ?)

1.3.1 Bugs

FreeIBComponents is intended to be used in production environments! To that end, a demonstrated bug *will be fixed* as promptly as possible.

Because FreeIBComponents is completely free, developers also have access to the source code. If a bug is fixed, send it to me, and I will incorporate it in future “official releases” of the product.

If a bug cannot be replicated, I will have a difficult time finding it, so I can’t guarantee resolution of so-called phantom bugs, but I will do my best to resolve them.

If you provide me with a bug fix, your name will be included in the acknowledgements section of this manual. Bug reports should be submitted as e-mail messages *only*.

1.3.2 Documentation

The policy on resolving source code bugs also stands for correcting and maintaining good documentation. Please send me any and all corrections and/or enhancements to the documentation. *Every product needs thorough and complete documentation*. Even though this product is free, I have a desire to see it used in all environments. I can’t expect a developer to use it if there is no documentation.

I can’t guarantee that I will respond to you, only that your comments will be heard and possibly included in future releases.

1.3.3 Wish lists

Users can feel free to provide me with wish list requests and/or FreeIBComponents enhancements.

If you send me a wish list request, you will *not* receive a response from me, but you can be assured that your request will be considered for future releases.

If you send me an enhancement or extension to FreeIBComponents that you would like included in the distribution, I will consider it and respond accordingly. Keep in mind, however, that FreeIBComponents and *all* source code is free. Do not even think about asking to distribute proprietary products with FreeIBComponents.

If I distribute product enhancements written by others than myself, their names will be included in the acknowledgements section of this manual, and their source code will be acknowledged appropriately.

1.3.4 Help desk

I *don't* do help desk support, at least not in general. You can feel free to e-mail me with questions, but it is more likely than not that you will not receive a response from me.

Help desk style questions should be directed to an appropriate Internet support group. The mailing list `INTERBASE@mers.com`, for example, provides excellent support for InterBase related questions. Also, I tend to make all of my announcements exclusively to this list. (This list is also a newsgroup. See `www.mers.com` for more information.)

If you absolutely require my assistance for some issue, I can provide services for standard consulting rates.

1.4 About the documentation

I have used a product called *texinfo* to produce all documentation for FreeIBComponents.

From a single source file, I can produce a formatted manual, ready for print-out (using *tex*), I can produce html, ready for viewing in a browser, and I can produce rich-text for the production of a help file.

I don't see the need for producing a help file, so I haven't included one... just the original *texinfo* documentation, a ready-produced Acrobat file, and a single HTML manual.

If the reader wishes to find more information on using *texinfo* with Windows, search the internet for "GNU tex texinfo Windows". (O yeah, *texinfo* is free, protected under the GNU Public License)

2 Release notes

2.1 Planned features

A bunch of people have made requests like: What about a TFIBTable? What about a RequestLive property as opposed to the select- update- insert- delete- refresh- queries?

Well, in short, I'm not sure I see the real value in doing any of these. If anyone has some thoughts on these as well as other feature requests, I'm very interested!

1. **TFIBScript**. A SQL script component that allows the execution of consecutive commands. Possibly add "logic" to the scripting process to allow for better handling and processing of errors. Not a big deal (and not too important, but it sure would be cool).
2. **TFIBEvent**. An event component needs to be implemented which works seamlessly with the TFIBDatabase component. Probably not too hard to implement, but I just haven't gotten around to it.
3. **TFIBArray**. Arrays need to be supported, if I ever get around to it. Arrays will have a TField descendant making it possible for Delphi's data-aware components to make use of them. Wouldn't this be fun?
4. Further develop the Batch input and output routines to provide a nice data pump tool. Batch input and output already provide a convenient mechanism for dumping data into a database and pulling it out, but it might be nice to set up something a bit simpler for the end-user to set up.
5. Create a property editor for the properties of type TFIBXSQlda. This will allow the design-time setting of parameters.

2.2 7/14/98

Bug fix release...

- Forgetting to put spaces between parameters in parameterized queries can choke the parser... whoops... THIS HAS BEEN FIXED.
- Trying to post a record that has not yet had any data placed in it causes a temporary file access error. THIS HAS BEEN FIXED.
- The new method of temporary file access has a bug that breaks refreshed queries... I have fixed this for the next release. THIS HAS BEEN FIXED.
- Accessing a field to a closed query causes an access violation. FIXED.

2.3 7/8/98

FreeIBComponents works with Delphi 4! Installation hasn't changed a bit, either... Just install the FIBComponents.dpk, and everything is taken care of!

Also, using FreeIBComponents with C++-Builder should be a bit simpler than before...

I have high hopes that this release is all that it says it is, but being the pessimist that I am, I know that it isn't... With all the new features and fixes in this release, you might

come across some stumbling blocks. Let me know when you find them, and better yet, let me know how to fix it. It'll be fixed in my next release which will be devoted primarily to "maintaining" this release.

New features:

- Added a published property to `TFIBQuery` called `GoToFirstRecordOnExecute`. You can probably guess what this does... and it defaults to `TRUE`. (see Section 7.4 [TFIBQuery Properties], page 24).

This property is new, and it can cause some problems with existing applications. If you have applications that assume that `TFIBQuery` is *not* on the first record after execution, you'll want to ensure that this property is set to `False`. (i.e., ensure it in *code*)

- Lookup and Locate are implemented.
- Add `AsInteger`, `AsVariant`, `AsStream` to `TFIBXSQLVAR`.
- Cached updates are now supported in `TFIBDataSet`. See documentation on `CachedUpdates`
`ApplyUpdates`
`CancelUpdates`
`RevertRecord`
`UpdateRecordTypes`
`UpdateStatus`
`CachedUpdateStatus`
`Undelete`
`FetchAll`
`UpdatesPending`
`OnUpdateError`
- Parameterized queries accept the ":" or "?" now, and there is now a `ParamCheck` property for `TFIBQuery`. This can present problems for DDL commands, so you should be sure to set `ParamCheck` to `False` when executing DDL.
- The way that the data set is buffered has been completely reworked, and it is much more solid now...
- `SQLPreprocessor.*` has been removed from the project in favor of a simpler hand-written version.
- `DelimitedFile.*` has been removed from the project in favor of a simpler hand-written version.

Bug fixes:

- Dmitry Vodennikov (vod@zaural.ru) points out that the line
`if Qry.SQL.Text <> " then InternalRefreshRow;`
in `TFIBDataSet.InternalPost` is incorrect. It should be:
`if FQRefresh.SQL.Text <> " then InternalRefreshRow;`
Fixed.
- `FOpen` in `TFIBQuery` is incorrectly set to `False` when EOF is reached. Fixed.
- Ravi Kumar (rkumar@interbase.com) pointed out that `GetRecordCount` should return `FRecordCount - "The number of records deleted."` Fixed.

- Master-detail relationships aren't handled as well as they should be. When a query is opened it should check the datasource property, not necessarily when the data source changes. Fixed.
- SetAsXSQLVAR was allocating memory incorrectly, and it wasn't accounting for the SQL_VARYING (varchar) type. This is now fixed.
- There was a blob reading bug that now seems to be fixed...

Known/possible bugs:

Array type fields are not displayed in a result set. PERIOD. For example, in FIB-WISQL, the job table in the example employee.gdb... The field LANGUAGE_REQ does not even show in the result set.

Paul Bonnette has reported some nasty problems using ReportBuilder with FIB, but I don't know anything more... Anyone else see anything like this?

2.4 6/3/98

I believe it is now safe to call this software BETA quality...

Yes, we're now in beta!

This release contains no new features, only bug fixes:

- Using a DBGrid: Go to "last" record in a dataset, resize the grid, and notice that the number of rows does not update in the grid correctly. Scrolling around in the grid in this state can cause access violations. This bug is really annoying, but if you don't resize your grids, or you don't use grids, it won't surface. Someone found this bug for me, because I couldn't see the forest for the trees, but being the jerk that I am, I forgot who it was... If you want credit given to you, please let me know, so I can include you in my acknowledgments.
- TFIBDatabase.GetUserNames function incorrectly processed the returned buffer from isc_database_info. Pablos (pablos@portal.ru) provided the bug fix for this.
- New-ish feature: FIBWISQL now allows you to get a list of connected users.
- The SourceChanged method of TFIBDataSet incorrectly assumes that a "detail" dataset (one with a master source) is always active... Carlos ?, (decumont@ibm.net) provided this bug fix.
- Added a constant FIB_VERSION to 'FIB.pas'. This is the date stamp (and official version number) of the release of FreeIBComponents.

2.5 5/14/98

Enhancements:

- New database property: DBParamByDPB. See docs, see Section 5.1 [TFIBDatabase Properties], page 15.
- Made the function IndexOfDBParam "public". See docs, see Section 5.2 [TFIBDatabase Methods], page 19.
- Now overriding the SetRecNo method.

Bug fixes:

- Preparing a stored procedure which returns *no* results causes a “feUnknownSQLDataType” error. Fixed.
- Mark Watts (Mark.Watts@universalsystems.com) points out that the method `TFIBStringField.SetAsString` should allocate `Size + 1` Bytes.
- `TFIBQuery.Close` attempts to close any style query, whether it is a `SQLSelect` statement or not, but *only* `SQLSelect` statements can be “closed”, as they are the only statements that can be “opened”.
- When a `TFIBDataSet` was told that the database is disconnecting, it forgot to set `Prepared` to false. This reaks havoc with the system when the user tries to re-activate the dataset, after having disconnected the database and reconnected. Luckily, this problem was quite simple to solve...
- `TFIBTransaction` doesn’t handle design-time activation properly. Courtesy of Adrian ? (adrian@softly.demon.co.uk)
- Geoff Worboys (gworboys@zeta.org.au) points out the next few bugs in getting `FreeIBComponents` to work with C++-Builder:
 - C is case sensitive and it likes its EOF and BOF properties to be spelled Eof and Bof... Go figure...
 - I forgot to actually *declare* `FieldDef` in the `InternalInitFielddefs` method. This declaration is required for C++-Builder releases.
- Jose Molina (jmolina@datbasedm.es) points out that the scroll bar in the grid control doesn’t work as nicely as it should.
- When trying to start a multi-database transaction in `TFIBTransaction`’s `StartTransaction` method, an access violation is generated during the call to `StartMultiple`. It was caused because I was allocating the pteb structure in a manner that was inconsistent with the way IB wanted to see it. I fixed this by adding a `TISC_TEB_ARRAY` and `PISC_TEB_ARRAY` datatype to the `ibase.pas` file, and I used the array type for setting the info instead of a simple `PISC_TEB`.
- Assigning an “AsString” parameter that was the empty string generated an error. This is now fixed.

2.6 4/13/98

Documentation fixes/enhancements:

- Documentation on using parameters has been added. See see Section 7.3 [Parameters], page 24.
- Examples on setting up an updatable `TFIBDataSet` have been provided. See see Section 10.2 [Updatable `TFIBDataSets`], page 36.

Code fixes/enhancements:

- The `Timeout` properties of `TFIBDatabase` and `TFIBTransaction` are now implemented.

- Geoff Worboys of Telesis Computing (gworboys@zeta.org.au) has provided some fixes that might allow FreeIBComponents to run with C++-Builder. The first fix entailed changing the order of the parameters to the `EFIBError` constructor, and the next fix entailed modifying some code in the `InternalInitFieldDefs` procedure in `'FIBDataSet.pas'`. To use FreeIBComponents with C++-Builder, see Chapter 3 [Installation], page 10.
- `ForceClose` of `TFIBDatabase` is now cleaner.
- The `DataSource` property of `TFIBDataSet` can now be used for establishing master-detail style relationships between queries.
- The `TFIBStringField` class was not entirely defined. It has now been fixed.
- `TFIBTransaction` now includes a published property `Active` for starting and rolling back transactions.
- Design-time creation of fields now works.
- `TFIBDataSets` can now be activated at design time.
- According to Delphi, all fields must have a name, so field names have to be generated for fields that are results of expressions.
- Karsten Strobel (AIT-Augsburg@t-online.de) pointed out that my methods for dealing with bookmarks were incorrect. As far as FreeIBComponents is concerned, a bookmark is just an integer, so... `BookmarkSize` needs to be set to `SizeOf(Integer)` in the create method. Then... `GetBookmarkData` needs to move the record number to the "pointed-to" data and `SetBookmarkData` needs to treat `Data` as a `PInteger`.
- Paul Bonnette (pbonnette@commtech.net) pointed out that the reference to `'WPDataFileFormat'` in `'main.pas'` (of FIBWISQL) should be removed for FIBWISQL to compile.
- Karsten Strobel (AIT-Augsburg@t-online.de) pointed out that `ActiveBuffer` might not always return the correct value. This has been fixed.

2.7 4/3/98

1. Bug fix - Delphi doesn't like duplicate field names in a result set, so care had to be taken to resolve that conflict.
2. Bug fix - `InternalClose` required the unbinding and freeing up of fields if `DefaultFields` is true.
3. Bug fix - the parser for preprocessing SQL is now thread-safe.

3 Installation

Installation of FreeIBComponents is really very simple:

1. Choose a directory where you want all FreeIBComponents to live. For example, you might want to put FreeIBComponents in

```
c:\develop\Lib
-or-
c:\Program Files\FreeIBComponents
```

2. Make that directory.
3. Using Winzip or some similar archiving tool, unzip this distribution into the above directory of your choosing.
4. Run Delphi.
5. Click “File, Open”.
6. If you unzipped FreeIBComponents into ‘c:\develop\Lib’, for example, open the file ‘c:\develop\Lib\FIBComponents.dpk’.
7. You will get a “package” window which suggests that you compile the package.
8. So, compile and install the package.

That’s it. FreeIBComponents are installed!

FreeIBComponents is currently a Delphi-only release; however, I also intend to support C++-Builder. As far as I know, this release *might* work with C++-Builder (at least that’s the rumour). I’m not experienced using C++-Builder, so if someone wants to provide some docs just like the above for getting FreeIBComponents up and running in a C++-Builder environment, I will be happy to include them here.

What I do know about getting the C++-Builder environment to work? the way I understand it, the compiler define for C++-Builder is `$VER110`, so all C++-Builder specific code is wrapped within those constraints. You must also make sure that ‘GDS32.LIB’ is in your include path.

I do my best to keep this documentation up to date, and relatively comprehensive. You must keep in mind, however, that this is a free product, and since all source code is provided, you can always figure out how something works by looking into the code yourself.

4 Licensing

As a general note to the reader, the license to use this software can be simply stated as: “This product comes to you at no cost, but you are being given a *bunch* of free source code. Please take care to give credit where credit is due.” This code is protected under copyright, and its distribution is restricted as set forth in the below license agreement. It is not meant to hinder your use of it. In fact, it is meant to protect *all of us*, so that we can all continue to use this free software.

Here is my own plain english interpretation of the below license. Just remember, though, that this interpretation is unofficial—You are bound by the below agreement, not my interpretation of it!

If you develop a *tool* whose intended audience is the developer community (a derivative of FIB, or another developer tool which makes use of FIB), then you *must* give me credit. I don’t really care if you give me credit when developing an *application* whose intended audience is the general user community; however, the license agreement does require you to give me credit for any product which makes use of FreeIBComponents.

In any case, FreeIBComponents comes to you for free, and you can sell your derivative products without worrying about compensating me.

GENERAL SOFTWARE LICENSE AGREEMENT

CAUTION: THE COPYING, MODIFICATION, TRANSLATION OR DISTRIBUTION OF THE OBJECT CODE, PROGRAM, SOFTWARE OR SOURCE CODE IMPLIES ACCEPTANCE OF THE TERMS OF THIS GENERAL SOFTWARE PROGRAM LICENSE AGREEMENT. YOU SHOULD READ CAREFULLY THE FOLLOWING TERMS AND CONDITIONS BEFORE YOU COPY, MODIFY, TRANSLATE OR DISTRIBUTE THE OBJECT CODE, PROGRAM, SOFTWARE OR SOURCE CODE.

1.0 DEFINITIONS

1.1 Licensee - The person who has the privilege to copy, modify, translate or distribute the object code, program, software and source code, subject to the terms and conditions of this General Software License Agreement.

1.2 Object Code - The version of a computer program in machine language, and therefore, ready to be executed by the computer.

1.3 Program - A sequence of instructions for executions by a computer.

1.4 Software - The computer program plus program documentation, if applicable.

1.5 Source Code - The version of a computer program in assembly language or high-level language, and therefore, not ready

to be executed by the computer.

1.6 Work - All forms of tangible or intangible property, based whole, in part or derived from the object code, program, software or source code.

1.7 You - The person who has the privilege to copy, modify, translate or distribute the object code, program, software and source code, subject to the terms and conditions of this General Software License Agreement.

2.0 LICENSE

2.1 The copyright holder hereby extends a license to you to use its copyrighted object code, program, software and source code, subject to the terms and conditions of this General Software License Agreement.

2.2 This license is applicable to the object code, program, software and source code distributed under the terms of this General Software License Agreement, any work containing the object code, program, software or source code distributed under the terms of this General Software License Agreement, any modification of the object code, program, software or source code distributed under the terms of this General Software License Agreement, any translation of the object code, program, software or source code distributed under the terms of this General Software License Agreement and any work containing a modification or translation of the object code, program, software or source code distributed pursuant to the terms and conditions of this General Software License Agreement.

2.3 You may copy, modify, translate and distribute the object code, program, software or source code distributed under the terms of this General Software License Agreement, subject to the terms and conditions of this General Software License Agreement.

2.4 If you copy, modify, translate or distribute the object code, program, software or source code distributed under the terms of this General Software License Agreement, you must publish and make known in a clear and conspicuous manner on each copy, modification, translation or distribution of the object code, program, software or source code that the copy, modification, translation or distribution of the object code, program, software or source code is subject to the terms and conditions of this General Software License Agreement and provide a copy of this General Software License Agreement with each copy, modification, translation or distribution of the object code, program, software

or source code.

2.5 If you derive, publish or distribute any work that is based whole or in part on the object code, program, software or source code distributed under the terms of this General Software License Agreement, or any modification or translation thereof, you must publish and make known in a clear and conspicuous manner on each such work that the work is subject to the terms and conditions of this General Software License Agreement and provide a copy of this General Software License Agreement with each work.

2.6 If you copy, modify, translate or distribute the object code, program, software or source code distributed under the terms and conditions of this General Software License Agreement, you must provide clear and conspicuous notice that you have copied, modified, translated or distributed the object code, program, software or source code distributed under the terms of this General Software License Agreement, and indicate the date of each such copy, modification, translation or distribution.

2.7 If you copy, modify, translate or distribute the object code, program, software or source code distributed under the terms of this General Software License Agreement, or publish or distribute any work that is derived, in whole or in part, from any copy, modification, translation or distribution of the object code, program, software or source code distributed under the terms of this General Software License Agreement, you cannot impose any further obligations or restrictions on any third person or entity other than what is contained in this General Software License Agreement.

3.0 NO WARRANTY

3.1 THE OBJECT CODE, PROGRAM, SOFTWARE AND SOURCE CODE ARE PROVIDED "AS IS" WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR USE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE OBJECT CODE, PROGRAM, SOFTWARE AND SOURCE CODE IS WITH YOU. SHOULD THE OBJECT CODE, PROGRAM, SOFTWARE AND SOURCE CODE PROVE DEFECTIVE, YOU ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

4.0 LIMITATION OF DAMAGES

4.1 IN NO EVENT WILL THE COPYRIGHT HOLDER OR ANY OTHER PERSON OR ENTITY BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, COMPENSATORY, GENERAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR THE INABILITY TO

USE THE OBJECT CODE, PROGRAM, SOFTWARE AND SOURCE CODE, EVEN IF THE COPYRIGHT HOLDER OR ANY OTHER PERSON OR ENTITY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

5.0 MISCELLANEOUS

5.1 The article and paragraph headings appearing in this General Software License Agreement have been asserted for the purpose of convenience and ready reference. They do not purport to, and shall not be deemed to define, limit, or extend the scope or intent of the articles and paragraphs to which they pertain.

5.2 This General Software License Agreement embodies the entire agreement respecting its subject matter. There are no promises, terms, conditions or obligations other than those expressly set forth herein. Unless otherwise expressly set forth herein, this General Software License Agreement supersedes all previous communications, representations, agreements, either verbal or written, warranties, promises, covenants or undertakings.

5.3 This General Software License Agreement shall not be modified, altered, amended or supplemented, except in writing signed by all parties hereto.

5.4 This General Software License Agreement shall be governed by the laws of the State of New Jersey.

5 TFIBDatabase – connecting to databases

TFIBDatabase abstracts the process of getting connected to InterBase. It's purpose is virtually identical to that of the Delphi's **TDatabase**, but **TFIBDatabase** does not directly manage transactions. Transactions are managed in the **TFIBTransaction** component (see Chapter 6 [TFIBTransaction], page 21).

5.1 Properties

Boolean Connected Published

This property indicates if the database is connected. Setting it to true will attempt to establish a connection with the database.

Integer DataSetCount Public, read-only

A count of all attached objects. The name **DataSet** is a bit misleading, as this number includes all attached blob streams, **TFIBQueries** and **TFIBDatasets**.

TFIBBase DataSets Public, read-only

Given an integer index, it returns the **TFIBBase** at the given index. This is used internally for broadcasting important messages to attached components. (For example, if the database is disconnecting, all attached data sets must be told).

String DBName Published

This is the database name. To connect to a local InterBase database, this is just the file name. To connect to an InterBase database on a remote server using TCP/IP, this is

```
<server name>:<file name on server>
```

To connect to an InterBase database on a remote server using NetBeui, this is

```
\\<server name>\<file name on server>.
```

String DBParamByDPB Public

This property allows the developer to inspect and set DPB parameters without looking at the **DBParams** string list.

For example **DBParamByDPB[isc_dpb_user_name]** is used to set and inspect the user name.

TStrings DBParams Published

When trying to establish a connection to a database, the database parameters are a textual representation of the parameters you want to pass to the InterBase server in order to establish the connection. For example:

```
isc_dpb_user_name=SYSDBA
isc_dpb_password=masterkey
isc_dpb_role=whatever
```

See the InterBase API Guide for further documentation on database parameters. For some understanding of how database parameter buffers are generated

in `FreeIBComponents`, take a look at the `GenerateDPB` procedure defined in `'FIB.pas'`.

When trying to create a database, `DBParams` contains the parameters part of the InterBase `CREATE DATABASE` DDL command. For example, to create a database as shown in the below DDL example,

```
CREATE DATABASE 'c:\test\exampleDB.gdb'
  USER "SYSDBA"
  PASSWORD "masterkey"
  PAGE_SIZE 4096
```

you would set the `DBName` property to `'c:\test\exampleDB.gdb'`, and set `DBParams` to

```
USER "SYSDBA"
PASSWORD "masterkey"
PAGE_SIZE 4096
```

Then, invoke the `CreateDatabase` method.

TISC_DB_HANDLE Handle Public, read-only

The InterBase database handle. This is used in all calls to the InterBase API.

TFIBTransaction PrimaryTransaction Published

`FreeIBComponents` does not mandate that a single database connection manage a single transaction; however, it is very convenient to let a database specify a primary transaction to be used.

Integer Timeout Published

By specifying a timeout, the developer can indicate how long the database should wait before automatically terminating the connection.

Integer TransactionCount Public, read-only

How many transactions are attached to this database?

TFIBTransaction Transaction Public, read-only

Given an integer index, it returns the `TFIBTransaction` at the given index. This is used internally for broadcasting important messages to attached components. (For example, if the database is disconnecting, all attached data sets must be told).

Boolean UseLoginPrompt Published

The developer can choose to use Delphi's canned Login prompt by setting this property to true.

The following is a list of properties that return interesting information about the state of the database. These are just wrappers for the API call, `isc_database_info`. To get more information on what these properties mean, refer to the InterBase documentation.

Long Allocation Public, read-only

See `isc_info_allocation` in the InterBase API Guide.

Long BaseLevel See <code>isc_info_base_level</code> in the InterBase API Guide.	Public, read-only
String DBFileName See <code>isc_info_db_id</code> in the InterBase API Guide.	Public, read-only
String DBSiteName See <code>isc_info_db_id</code> in the InterBase API Guide.	Public, read-only
Long DBImplementationNo See <code>isc_info_implementation</code> in the InterBase API Guide.	Public, read-only
Long DBImplementationClass See <code>isc_info_implementation</code> in the InterBase API Guide.	Public, read-only
Long NoReserve See <code>isc_info_no_reserve</code> in the InterBase API Guide	Public, read-only
Long ODSMinorVersion See <code>isc_info_ods_minor_version</code> in the InterBase API Guide	Public, read-only
Long ODSMajorVersion See <code>isc_info_ods_version</code> in the InterBase API Guide	Public, read-only
Long PageSize See <code>isc_info_page_size</code>	Public, read-only
String Version See <code>isc_info_info_version</code> in the InterBase API Guide	Public, read-only
Long CurrentMemory See <code>isc_info_current_memory</code> in the InterBase API Guide	Public, read-only
Long ForcedWrites See <code>isc_info_forced_writes</code> in the InterBase API Guide	Public, read-only
Long MaxMemory See <code>isc_info_max_memory</code> in the InterBase API Guide	Public, read-only
Long NumBuffers See <code>isc_info_num_buffers</code> in the InterBase API Guide	Public, read-only
Long SweepInterval See <code>isc_info_sweep_interval</code> in the InterBase API Guide	Public, read-only
TStringList UserNames See <code>isc_info_user_names</code> in the InterBase API Guide	Public, read-only
Long Fetches See <code>isc_info_fetches</code> in the InterBase API Guide	Public, read-only
Long Marks See <code>isc_info_marks</code> in the InterBase API Guide	Public, read-only

Long Reads	Public, read-only
See <code>isc_info_reads</code> in the InterBase API Guide	
Long Writes	Public, read-only
See <code>isc_info_writes</code> in the InterBase API Guide	
Long BackoutCount	Public, read-only
See <code>isc_info_backout_count</code> in the InterBase API Guide	
Long DeleteCount	Public, read-only
See <code>isc_info_delete_count</code> in the InterBase API Guide	
Long ExpungeCount	Public, read-only
See <code>isc_info_expunge_count</code> in the InterBase API Guide	
Long InsertCount	Public, read-only
See <code>isc_info_insert_count</code> in the InterBase API Guide	
Long PurgeCount	Public, read-only
See <code>isc_info_purge_count</code> in the InterBase API Guide	
Long ReadIdxCount	Public, read-only
See <code>isc_info_read_idx_count</code> in the InterBase API Guide	
Long ReadSeqCount	Public, read-only
See <code>isc_info_read_seq_count</code> in the InterBase API Guide	
Long UpdateCount	Public, read-only
See <code>isc_info_update_count</code> in the InterBase API Guide	
Long LogFile	Public, read-only
See <code>isc_info_log_file</code> in the InterBase API Guide	
String CurLogFileName	Public, read-only
See <code>isc_info_cur_logfile_name</code> in the InterBase API Guide	
Long CurLogPartitionOffset	Public, read-only
See <code>isc_info_cur_log_part_offset</code> in the InterBase API Guide	
Long NumWALBuffers	Public, read-only
See <code>isc_info_num_wal_buffers</code> in the InterBase API Guide	
Long WALBufferSize	Public, read-only
See <code>isc_info_wal_buffer_size</code> in the InterBase API Guide	
Long WALCheckpointLength	Public, read-only
See <code>isc_info_wal_ckpt_length</code> in the InterBase API Guide	
Long WALCurCheckpointInterval	Public, read-only
See <code>isc_info_wal_cur_ckpt_interval</code> in the InterBase API Guide	
String WALPrvCheckpointFilename	Public, read-only
See <code>isc_info_wal_prv_ckpt_fname</code> in the InterBase API Guide	

Long	WALPrvCheckpointPartOffset	Public, read-only
	See <code>isc_info_wal_prv_ckpt_poffset</code> in the InterBase API Guide	
Long	WALGroupCommitWaitUSecs	Public, read-only
	See <code>isc_info_wal_grpc_wait_usecs</code> in the InterBase API Guide	
Long	WALNumIO	Public, read-only
	See <code>isc_info_wal_num_id</code> in the InterBase API Guide	
Long	WALAvergeIOSize	Public, read-only
	See <code>isc_info_wal_avg_io_size</code> in the InterBase API Guide	
Long	WALNumCommits	Public, read-only
	See <code>isc_info_wal_num_commits</code> in the InterBase API Guide	
Long	WALAvergeGroupCommitSize	Public, read-only
	See <code>isc_info_wal_avg_grpc_size</code> in the InterBase API Guide	

5.2 Methods

Create	(<i>AOwner: TComponent</i>)	Constructor
	Create an instance of a TFIBDatabase.	
Destroy		Destructor
	Free up all resources associated with this instance.	
Integer	AddDataSet (<i>ds: TFIBBase</i>)	Function
	Add an object of type TFIBBase to the components list of attached sets. It returns the index where it entered the set.	
Integer	AddTransaction (<i>TR: TFIBTransaction</i>)	Function
	Add a transaction to the databases list of connected transactions.	
CheckActive		Procedure
CheckInactive		Procedure
CheckDatabaseName		Procedure
	Raise an exception if the database connection is not active, if it is active or if there is no database name specified, respectively	
Close		Procedure
ForceClose		Procedure
	Ask (or force) the database connection to close. Forcing the database connection to close can be messy—Use it at your own risk.	
CreateDatabase;		Procedure
DropDatabase;		Procedure
	Try to create (or drop) the named database, using the <i>DBParams</i> as the “rest” of the CREATE DATABASE command.	

Integer FindTransaction (<i>TR</i> : TFIBTransaction)	Function
Return an index (or -1) of <i>TR</i> .	
Integer IndexOfDBConst (<i>st</i> : String)	Function
Try to find the parameter in the database parameters list. Return -1 if nothing was found.	
Open	Procedure
Try to open the named database.	
RemoveDataSet (<i>Idx</i> : Integer)	Procedure
RemoveDataSets	Procedure
Remove the indexed set from the list of data sets, or remove all of them.	
RemoveTransaction (<i>Idx</i> : Integer)	Procedure
RemoveTransactions	Procedure
Remove the indexed transaction, or remove all of them.	

5.3 Events

TNotifyEvent OnConnect	Published Event
This event is kicked off after a database succesfully connects.	
TNotifyEvent OnTimeout	Published Event
This event is kicked off after a database connection “times out”.	

6 TFIBTransaction – managing transactions

6.1 Properties

Boolean Active	Published
This allows the design-time activation of transactions. When set, a transaction is active. “Deactivating” a transaction is equivalent to rolling it back.	
Integer DatabaseCount	Public, read-only
How many databases are a part of this transaction? (Hey! InterBase supports multi-database transactions... So does FreeIBComponents!)	
TFIBDatabase Databases	Public, read-only
Given an integer index, return the database at the index.	
Integer DataSetCount	Public, read-only
Return the number of data sets attached to the transaction.	
TFIBBase DataSets	Public, read-only
Given an integer index, return the data set at the index.	
TISC_TR_HANDLE Handle	Public, read-only
The InterBase transaction handle.	
Boolean InTransaction	Public, read-only
Is the transaction active?	
Integer Timeout	Published
How long should this transaction wait before automatically committing or rolling back?	
TTransactionAction TimeoutAction	Published
When the transaction times out, should the transaction commit, roll back or commit-retaining?	
PChar TPB	Public, read-only
A read only look at the transaction parameter buffer.	
Short TPBLength	Public, read-only
The length of the transaction parameter buffer.	
TStrings TRParams	Published
A textual representation of the transaction parameter buffer.	
See the InterBase API Guide for the names of the parameters to provide.	

6.2 Methods

Create (<i>AOwner: TComponent</i>)	Constructor
Create an instance of a TFIBTransaction .	
Destroy	Destructor
Free up all resources associated with this instance.	
Integer AddDatabase (<i>db: TFIBDatabase</i>)	Function
Integer AddDataSet (<i>ds: TFIBBase</i>)	Function
Add a database to the list of databases that this transaction manages. Add a data set to the list of data sets this transaction manages.	
CheckDatabasesInList	Procedure
CheckInTransaction	Procedure
CheckNotInTransaction	Procedure
Raise an exception if there are no databases in the transaction's database list.	
Raise an exception if the transaction is active. Raise an exception if the transaction is not active.	
Commit	Procedure
CommitRetaining	Procedure
Rollback	Procedure
StartTransaction	Procedure
These methods pretty much explain themselves...	
Integer FindDatabase (<i>db: TFIBDatabase</i>)	Function
Find the referenced database in the database list, and return the index or -1, if the database was not found.	
RemoveDatabase (<i>Idx: Integer</i>)	Procedure
RemoveDatabases	Procedure
Remove the indexed database from the transaction's list of databases, or remove all databases.	
RemoveDataSet (<i>Idx: Integer</i>)	Procedure
RemoveDataSets	Procedure
Just like above, but remove the data sets.	

6.3 Events

TNotifyEvent OnTimeout	Published Event
This event is kicked off after a transaction has timed out.	

7 TFIBQuery – executing queries

TFIBQuery is the first step on the way to developing a descendant of TDataSet. This component is an extremely simple component that will quickly and efficiently execute an InterBase SQL statement.

7.1 When to use TFIBQuery

FreeIBComponents includes two query-style components: TFIBQuery and TFIBDataSet (see Chapter 10 [TFIBDataSet], page 33). Either of these components can execute any valid DSQL statement.

Many developers will ask the obvious question: “When is desirable to use TFIBQuery and when is it desirable to use TFIBDataSet?”

TFIBQuery is an extremely simple component which is a very light-weight layer on top of the InterBase API. When used to execute SQL-Selects, its results are unbuffered, hence unidirectional. TFIBDataSet, on the other hand, is built primarily for SQL-Select statements. It buffers its result set, so that the result set is completely scrollable. Also, it is a descendant of TDataSet, which exposes FreeIBComponents to all data-aware components.

So, use TFIBDataSet when you require the use of data-aware components or a scrollable result set. In any other situation, however, it is advisable that you use TFIBQuery, which requires much less overhead.

7.2 Extended FreeIBComponents query syntax

FreeIBComponents uses an extended SQL syntax for its SQL statements. The extended syntax is strictly for the purposes of naming parameters. The extension is quite simple, but it is important that you note this for writing parameterized queries.

In standard InterBase SQL, a parameterized query would be entered as follows:

```
select ...  
  from ...  
 where <field> = ?
```

In FreeIBComponents, this query *must* be written as:

```
select ...  
  from ...  
 where <field> = ?<param_name>
```

Where <param_name> is made up of alphanumerics, the character ‘_’, and the character ‘\$’.

If you do not use this syntax for parameterized queries, the query will *not* execute correctly.

7.3 Using parameterized queries

The passing of parameters to `TFIBDataSet` and `TFIBQuery` is handled in a like manner.

The property `Params` exposes the raw InterBase extended SQL descriptor through an object, not too interestingly named `TFIBXSQlda`.

When I get the time, I'll try to explain all the gory details of using the `TFIBXSQlda` class. For now, you can refer to the source code, and I'll just provide some silly examples of its use:

```
var
  MyDataSet: TFIBDataSet;
  MyQuery: TFIBQuery;

...
MyDataSet.SelectSQL.Text := 'select * from t1 ' +
                             'where f1 = ?f1 and ' +
                             'f2 = ?f2 and ' +
                             'f3 = ?f3 and ' +
                             'f4 = ?f4';

MyQuery.SQL.Assign(MyDataSet.SelectSQL);

// Assign f1 the value 1
MyDataSet.Params[0].AsLong := 1;
// Assign f2 the value 3
MyDataSet.ParamsByName['f2'].AsLong := 3;
// Assign f3 the value 'hello world'
MyDataSet.ParamsByName['F3'].AsString := 'hello world';
// Assign f4 the date 3/4/98
MyDataSet.Params[3].AsDateTime := StrToDate('3/4/98');

// Assign f1 the value 1
MyQuery.Params[0].AsLong := 1;
// Assign f2 the value 3
MyQuery.ParamsByName['f2'].AsLong := 3;
// Assign f3 the value 'hello world'
MyQuery.ParamsByName['F3'].AsString := 'hello world';
// Assign f4 the date 3/4/98
MyQuery.Params[3].AsDateTime := StrToDate('3/4/98');

...
```

Just so you know, the `Params` property is run-time editable only. If someone wants to write a property editor for params, it would be nice.

7.4 Properties

Boolean BOF

Is the query at the beginning of the query?

Public, read-only

TFIBDatabase Database	Published
The database component to which the query is attached.	
PISC_DB_HANDLE DBHandle	Public, read-only
A pointer to the InterBase database handle.	
Boolean EOF	Public, read-only
Has the query returned all result rows? (Only meaningful for select queries).	
Integer FieldIndex	Public, read-only
Given a field name, return the index of the named field.	
<code>i := qry.FieldIndex['INT_FIELD'];</code>	
Boolean GoToFirstRecordOnExecute	Published
This property tells the query to go to the first record in the result set after opening it.	
By default, this property is set to true. It exists primarily for use in TFIBDataSet , which sets this value to False for its internal TFIBQuery 's.	
Boolean Open	Public, read-only
Is the query open?	
TFIBXSQLDA Params	Public, read-only
TFIBXSQLDA is a FreeIBComponents extension to InterBase's extended SQL descriptor. For the time-being, you're stuck looking at the source code and the InterBase API Guide for more information.	
String Plan	Public, read-only
Once a query has been prepared, you can get the query plan by querying this property.	
Boolean Prepared	Public, read-only
Has this query been prepared yet?	
Integer RecordCount	Public, read-only
The current count of records returned from the query. If result set is to return 100 rows, <i>RecordCount</i> will be 100 only after all records have been visited. That is, after looking at the first record, <i>RecordCount</i> is 1, and so on.	
TStrings SQL	Published
This the the SQL query that you wish to execute.	
TFIBSQLTypes SQLType	Public, read-only
This property tells you what type of query you are about to execute.	
Possible values are:	
<pre>TFIBSQLTypes = (SQLUnknown, SQLSelect, SQLInsert, SQLUpdate, SQLDelete, SQLDDL, SQLGetSegment, SQLPutSegment, SQLExecProcedure, SQLStartTransaction, SQLCommit, SQLRollback, SQLSelectForUpdate, SQLSetGenerator);</pre>	

TFIBTransaction Transaction	Published
To what transaction does this query belong?	
PISC_TR_HANDLE TRHandle	Public, read-only
A pointer to the transaction handle.	

7.5 Methods

Create (<i>AOwner: TComponent</i>)	Constructor
Create an instance of a TFIBQuery .	
Destroy	Destructor
Free up all resources associated with this instance.	
BatchInput (<i>InputObject: TFIBBatchInputStream</i>)	Procedure
BatchInput executes the parameterized query in <i>SQL</i> for all input in the referenced <i>InputObject</i> .	
See Chapter 8 [Batching streams], page 28 for more information on batching streams.	
BatchOutput (<i>OutputObject: TFIBBatchOutputStream</i>)	Procedure
BatchOutput outputs the select query in <i>SQL</i> to the referenced <i>OutputObject</i> .	
See Chapter 8 [Batching streams], page 28 for more information.	
CheckClosed	Procedure
CheckOpen	Procedure
CheckValidStatement	Procedure
Raise exceptions for the following possible conditions: The query is not closed, the query is not open, or the query does not have a valid statement.	
Close	Procedure
Close the query.	
TFIBXSQlda Current	Function
TFIBXSQlda Next	Function
Return a FreeIBComponents extended SQL descriptor for the current record, or the next record. Next returns nil if there are no records left in the result set.	
For the time-being, you will have to refer to InterBase documentation on extended SQL descriptors, and the FreeIBComponents source for information on TFIBXSQlda	
ExecQuery	Procedure
Execute the query in <i>SQL</i> .	
FreeHandle	Procedure
Free up the InterBase resources associated with the query. In other words, “unprepare” the query.	
Prepare	Procedure
Prepare the query for execution.	

7.6 Events

TNotifyEvent **OnSQLChanging**

Published Event

When the SQL query is being modified, kick off an event. If an exception is raised in this event, then the SQL is not changed.

8 Batching input and output

It is well recognized that a good set of database components needs to have some convenient mechanism for migrating data back and forth from the database.

The two abstract classes, `TFIBBatchInputStream` and `TFIBBatchOutputStream` make it possible to input and output data in virtually any format.

8.1 The `TFIBBatch` abstract class

The `TFIBBatch` abstract class is the base for the two stream classes below. It's declaration is as follows:

```
TFIBBatch = class(TObject)
protected
  FFilename: String;
  FColumns: TFIBXSQlda;
  FParams: TFIBXSQlda;
public
  procedure ReadyStream; virtual; abstract;
  property Columns: TFIBXSQlda read FColumns;
  property Filename: String read FFilename write FFilename;
  property Params: TFIBXSQlda read FParams;
end;
```

Descendants of this class can specify a file name (for input or output), and a `TFIBXSQlda` representing a record or parameters. The `ReadyStream` procedure is called right before performing the batch input or output.

This is an abstract class, thus all discussions of it are a bit “abstract”. By referring to the concrete examples in Section 8.4 [Delimited Files], page 30 and Section 8.5 [Raw Files], page 31, the reader will gain a deeper understanding of their usage.

8.2 The `TFIBBatchInputStream` abstract class

The `TFIBBatchInputStream` abstract class provides the basis for performing all batch input. Here is its declaration in ‘`FIBQuery.pas`’.

```
TFIBBatchInputStream = class(TFIBBatch)
public
  function ReadParameters: Boolean; virtual; abstract;
end;
```

To get a sense of how a `TFIBBatchInputStream` is used, let's have a look at the source for `TFIBQuery`'s `BatchInput` method:

```
procedure TFIBQuery.BatchInput(InputObject: TFIBBatchInputStream);
begin
  if not Prepared then
    Prepare;
  InputObject.FParams := Self.FSQldParams;
  InputObject.ReadyStream;
```

```

    if FSQLType in [SQLInsert, SQLUpdate, SQLDelete] then
        while InputObject.ReadParameters do
            ExecQuery;
        end;
    end;

```

An example of how a developer would use this method is as follows:

```

...
var
    InputObject: TFIBInputRawFile;
...

InputObject := TFIBInputRawFile.Create;
try
    InputObject.Filename := 'Test.txt';
    (*
    * Qry is some previously declared TFIBQuery that is connected
    * to some existing database, etc...
    *)
    Qry.SQL.Text := 'insert into t1 ' +
                    '(int_field, str_field, memo_field) ' +
                    'values ' +
                    ' (?a, ?b, ?c)';
    Qry.BatchInput(InputObject);
finally
    InputObject.Free;
end;

```

Once again, refer to Section 8.4.1 [Delimited Input], page 30 and Section 8.5 [Raw Files], page 31 for a concrete example of batch input streams.

8.3 The TFIBBatchOutputStream abstract class

Just like TFIBBatchInputStream, the TFIBBatchOutputStream abstract class provides the basis for performing all batch output. Here is its declaration in 'FIBQuery.pas':

```

TFIBBatchOutputStream = class(TFIBBatch)
public
    function WriteColumns: Boolean; virtual; abstract;
end;

```

To get a sense of how a TFIBBatchOutputStream is used, let's have a look at the source for TFIBQuery's BatchOutput method:

```

procedure TFIBQuery.BatchOutput(OutputObject: TFIBBatchOutputStream);
begin
    CheckClosed;
    if not Prepared then
        Prepare;
    if FSQLType = SQLSelect then begin
        try
            ExecQuery;

```

```

        OutputObject.FColumns := Self.FSQLRecord;
        OutputObject.ReadyStream;
        while (Next <> nil) and (OutputObject.WriteColumns) do ;
    finally
        Close;
    end;
end;
end;

```

An example of how a developer would use this method is as follows:

```

...
var
    OutputObject: TFIBOutputRawFile;
...

OutputObject := TFIBOutputRawFile.Create;
try
    OutputObject.Filename := 'Test.txt';
    (*
    * Qry is some previously declared TFIBQuery that is connected
    * to some existing database, etc...
    *)
    Qry.SQL.Text := 'select int_field, str_field, memo_field '
                    'from t1';
    Qry.BatchOutput(OutputObject);
finally
    OutputObject.Free;
end;

```

Once again, refer to Section 8.4.2 [Delimited Output], page 31 and Section 8.5 [Raw Files], page 31 for a concrete example of batch output streams.

8.4 Migrating data to and from delimited files

8.4.1 Delimited file input

The class `TFIBInputDelimitedFile`, defined in `'FIBMiscellaneous.pas'` is used for performing batch inputs of data from pipe-tilde or tab-crlf delimited files.

There are three important properties (aside from those already in the ancestor class) that affect the resulting delimited file.

Boolean <code>AutoDetect</code>	Property
<i>AutoDetect</i> tells FreeIBComponents to try to figure out if the file is a tab-crlf delimited file or if it is a pipe-tilde delimited file.	

TFIBDelimitedFormat <code>InputFormat</code>	Property
<i>InputFormat</i> tells FreeIBComponents what format to expect in the delimited file. The possible values are <code>fdfTabs</code> or <code>fdfPipes</code> . If <i>AutoDetect</i> is set to <code>True</code> , this property is ignored.	

Boolean `SkipTitles`

Property

SkipTitles tells FreeIBComponents if it should treat the first record of a delimited file as titles. Obviously, field titles are not useful in batch inputs, so this feature allows you to skip them.

8.4.2 Delimited file output

`TFIBOutputDelimitedFile` has two important properties (aside from those properties already discussed in its ancestor class).

TFIBDelimitedFormat `OutputFormat`

Property

This property indicates what format (tab-crlf or pipe-tilde) should be used for outputting the delimited file.

Boolean `OutputTitles`

Property

This property indicates if the developer wishes to output the field titles at the top of the file.

8.5 Migrating data to and from raw files

`TFIBInputRawFile` and `TFIBOutputRawFile` are the respective input and output classes for migrating data to and from raw files.

A raw file is the equivalent of what InterBase outputs in its external files. The great thing about raw files in FreeIBComponents is that they are not limited to straight character format.

Whatever structure is defined by your query is what goes into the file!

Raw files have no available options, and they are probably the fastest way (aside from an external tables themselves) to get data in and out of InterBase. Their only drawback is that fixed-width files can be a chore to deal with.

Well, I guess you win some and lose some...

9 Blob streams

FreeIBComponents supports InterBase blobs through the use of two blob streams called `TFIBBlobStream` and `TFIBDSBlobStream`.

When the developer wishes to read and write to blobs in a `TFIBQuery` he or she should *always* use `TFIBBlobStream`. `TFIBDSBlobStream` is strictly for internal use by `TFIBDataSet`.

It is important to mention `TFIBDSBlobStream`, but it will not be documented here, as it is strictly an *internal* class.

Right now, I haven't got time to go into detail on the `TFIBBlobStream` class. If you have grown comfortable studying my code, then hopefully you will have some luck studying its source code in `'FIBMiscellaneous.pas'`

10 TFIBDataSet – The fun part!

Possibly the coolest aspect of FreeIBComponents is that developers can avoid the BDE but still make use of the plethora of data-aware components that are out there, including reporting tools such as ReportPrinterPro and QuickReports.

Delphi (and Borland) did a great thing by putting together the **TDataSet** abstract class, and I will do my best to document not only the use of **TFIBDataSet**, but *how* I built it as well.

Note: When outlining properties, methods and events for **TFIBDataSet**, I will only document those properties, events and methods that are *not* inherited from **TDataSet**.

10.1 Overriding Delphi's TDataSet abstract class

Overriding Delphi's **TDataSet** is a challenging task! But, it is not terribly difficult. Hopefully, this manual, in conjunction with the source code, will give you a deeper understanding and greater appreciation for the elegance, simplicity and power in Delphi. Kudos to the Borland Delphi development team!

Below is a list of all methods that were overridden to get to **TFIBDataSet**:

PChar AllocRecordBuffer (Abstract) Function

This method is called whenever Delphi requires a new record buffer for caching data away for short periods of time. **TDataSet** *never* directly reads this buffer. It is the job of the rest of the abstract methods below to return comprehensible information to **TDataSet**.

TStream CreateBlobStream (*Field: TField, Mode: TBlobStreamMode*) Function

For this method to do anything meaningful, **TFIBDataSet** must override it. A **TBlobField** uses blob streams to read and write blob data, so this function is absolutely essential if **TFIBDataSet** expects to provide blob access.

DoBeforeEdit Procedure

DoBeforeInsert Procedure

The only reason that **TFIBDataSet** overrides them is so that it can do preliminary checks to see if the underlying data set is updatable.

FreeRecordBuffer (*var Buffer: PChar*) (Abstract) Procedure

It frees up the record buffer that was allocated in **AllocRecordBuffer**.

GetBookmarkData (*Buffer: PChar, Data: Pointer*) (Abstract) Procedure

A bookmark is a way of quickly and uniquely identifying a record. Since FreeIBComponents is completely buffered, bookmark data is basically just an integer.

This routine copies the bookmark data in *Buffer* to the buffer pointed to by *Data*.

Boolean GetCanModify Function

This is just a helper function that helps to determine if a data set is updatable.

TFieldClass GetFieldClass (*FieldType*: TFieldType) Function

InterBase supports very long strings, and returns them as part of the actual record (as opposed to blobs). The standard TStringField won't look at strings longer than dsMaxStringSize. FreeIBComponents defines a TFIBStringField which allows strings of any length.

This function simply overrides the default behavior of TDataSet to use a TFIBStringField for all strings instead of a TStringField.

Boolean GetFieldData (*Field*: TField, *Buffer*: (Abstract) Function
Pointer)

Given a record buffer (*Buffer*) and a TField (*Field*), copy the data for that field into the TField.

TGetResult GetRecord (*Buffer*: PChar, *GetMode*: (Abstract) Function
TGetMode, DoCheck: Boolean)

Copy the requested record into *Buffer*, and return grOK, grEOF, grBOF or grError depending on what happens.

Integer GetRecordCount Function

This method returns the *current* record count of the record set. It is only accurate if the entire record set has been scanned.

Word GetRecordSize (Abstract) Function

Return the record size of the data set.

Important note: GetRecordSize returns a Word which has a maximum value of 65536. This means that even though an InterBase record could certainly be wider than this, Delphi's TDataSet cannot support InterBase records any larger.

InternalAddRecord (*Buffer*: Pointer, *Append*: (Abstract) Procedure
Boolean)

Buffer is a preconstructed record buffer whose contents has a record to be inserted into the data set.

FreeIBComponents ignores the *Append* parameter, as new records are always appended.

InternalClose (Abstract) Procedure

This method is called when TDataSet needs to close the data set.

InternalDelete (Abstract) Procedure

This method is called to delete the current record.

InternalFirst (Abstract) Procedure

This method is called to go to the first record in the data set.

InternalGotoBookmark (*Bookmark*: Pointer) (Abstract) Procedure

This method instructs TFIBDataSet to jump to the referenced record.

InternalHandleException (Abstract) Procedure

Internally handle an exception. Right now, this method just calls

```
Application.HandleException(Self),
```

which is the “default” behavior of most components.

InternalInitFieldDefs (Abstract) Procedure

As a SQL-Select query is being prepared, **InternalInitFieldDefs** is called to ensure that the correct **TField**’s are created for displaying data back to the user.

InternalInitRecord (*Buffer: PChar*) (Abstract) Procedure

Given a previously allocated buffer *Buffer*, ensure that *Buffer* is set up as an appropriately *empty* record.

InternalLast (Abstract) Procedure

This method is called to jump to the last record in the data set.

InternalOpen (Abstract) Procedure

This method is called when **TDataSet** needs to open the data set.

InternalPost (Abstract) Procedure

InternalPost is called when a record must be “posted” to the server. Depending on the state of the data set, this will either call an **UPDATE** routine or an **INSERT** routine.

InternalRefresh Procedure

This method is called to refresh the current record from the server.

InternalSetToRecord (*Buffer: PChar*) (Abstract) Procedure

Given *Buffer*, which is a buffer containing an existing record in the data set, jump to that record.

Boolean IsCursorOpen (Abstract) Function

Determine if the “cursor” is open. Since **TFIBDataSet** is buffered, this always returns true if the data set is open.

SetBookmarkFlag (*Buffer: PChar, Value: TBookmarkFlag*) (Abstract) Procedure

This sets the bookmark flag.

SetBookmarkData (*Buffer: PChar, Data: Pointer*) (Abstract) Procedure

This tells *Buffer* that it is bookmarked with *Data*.

SetFieldData (*Field: TField, Buffer: Pointer*) (Abstract) Procedure

An extremely important method that sets field data. Given a *Field*, copy the data in *Field* into the appropriate field position in *Buffer*, and mark the record as having been modified.

This is it! Obviously, quite a bit more of *new* code was added to `TFIBDataSet`, but the above method descriptions give a reasonable explanation of what has been overridden. You are cordially invited to study the source code at your leisure. I invite comments and bug fixes!

There are many features of a `TDataSet` that have been exposed in `TFIBDataSet`, but I have not had the time, nor the inclination to test them. As developers come across inconsistencies in the way that `TFIBDataSet` behaves and the way it *should* behave, I'd like to know.

10.2 Making a `TFIBDataSet` updatable

Making a `TFIBDataSet` updatable is extremely easy. The primary SQL property of `TFIBDataSet` is `SelectSQL`, but there are four other SQL properties that allow the user to treat a query as an updatable data set: `DeleteSQL`, `InsertSQL`, `RefreshSQL`, and `UpdateSQL`.

Each of these queries allow the user to delete a row, insert a row, refresh a row, and update a row.

Let's assume that you have a table in your database called `t1`:

```
create table t1 (
  f1 integer constraint primary key,
  f2 integer,
  f3 varchar(128)
);
```

Now, your `SelectSQL` property says:

```
select * from t1
```

To be capable of *deleting* data, just make the `DeleteSQL` property say this:

```
delete from t1
  where f1 = ?old_f1
```

To be capable of *inserting* data, just make the `InsertSQL` property say this:

```
insert into t1
  (f1, f2, f3)
  values
  (?f1, ?f2, ?f3)
```

To be capable of *refreshing* data, just set the `RefreshSQL` to

```
select f1, f2, f3
  from t1
  where f1 = ?f1
```

To be capable of *updating* data, just set the `UpdateSQL` property to

```
update t1
  set f1 = ?f1,
      f2 = ?f2,
      f3 = ?f3
  where f1 = ?old_f1
```

It's that easy! You will notice the use of “OLD_” and “NEW_”. This is to refer to the “old” field value or the “new” field value, respectively.

You should also note that this convention for setting up “live” queries is a bit more complicated than merely setting a `RequestLive` property, but it offers complete flexibility. For example, you might want to set it up so that users don't have the option to modify the primary key, so you could set the `UpdateSQL` property to the following

```
update t1
  set f2 = ?f2,
      f3 = ?f3
 where f1 = ?old_f1
```

As you explore using this method for establishing “live queries”, you will find that this is rather simple and intuitive.

10.3 Properties

Boolean CachedUpdates Published

Does this data set use cached updates? If so, the data set retains all information regardless of a database close or transaction end.

See `ApplyUpdates`, `CancelUpdates`, `RevertRecord`, `UpdateRecordTypes`, `UpdateStatus`, `CachedUpdateStatus`, `Undelete`, `UpdatesPending`.

TFIBDatabase Database Published

What database does this query query?

PISC_DB_HANDLE DBHandle Public, read-only

A pointer to the InterBase database handle.

TStrings DeleteSQL Published

TStrings InsertSQL Published

TStrings RefreshSQL Published

TStrings SelectSQL Published

TStrings UpdateSQL Published

The *SelectSQL* is the primary SQL property that should be used for executing queries.

The rest of the queries are intended to allow the developer to make a query delete-able (*DeleteSQL*), insert-able (*InsertSQL*), row-level-refresh-able (*RefreshSQL*), and updatable (*UpdateSQL*).

For more detailed information on creating updatable data sets, see Section 10.2 [Updatable TFIBDataSets], page 36.

TFIBXSQLDA Params Public, read-only

This property allows the developer to specify values for a parameterized query.

Boolean Prepared Public, read-only

Have the set of data set queries been prepared?

TFIBQuery FQDelete	Public, read-only
TFIBQuery QInsert	Public, read-only
TFIBQuery QRefresh	Public, read-only
TFIBQuery QSelect	Public, read-only
TFIBQuery QUpdate	Public, read-only

These properties give the developer direct access to the associated **TFIBQuery**'s that underly the **TFIBDataSet**.

TFIBSQLTypes StatementType	Public, read-only
-----------------------------------	-------------------

This returns the statement type of the *QSelect* query.

TFIBTransaction Transaction	Published
------------------------------------	-----------

Under what transaction does this query execute?

PISC_TR_HANDLE TRHandle	Public, read-only
--------------------------------	-------------------

A pointer to the InterBase transaction handle.

TFIBUpdateRecordTypes UpdateRecordTypes	Published
--	-----------

Set of these possible values: (*cusUnmodified*, *cusModified*, *cusInserted*, *cusDeleted*, *cusUninserted*).

UpdateRecordTypes indicates those records that are visible to the user when *CachedUpdates* is *True*.

If *CachedUpdates* is *False*, this property means nothing.

Boolean UpdatesPending	Public, read-only
-------------------------------	-------------------

If *CachedUpdates* is *True*, then this value indicates if there are cached updates “pending”.

10.4 Methods

ApplyUpdates	Procedure
CancelUpdates	Procedure

When *CachedUpdates* are enabled, these routines globally apply or cancel the pending updates.

BatchInput (<i>InputObject: TFIBBatchInputStream</i>)	Procedure
BatchOutput (<i>OutputObject: TFIBBatchOutputStream</i>)	Procedure

Execute a batch input/output routine on the given *Input(Output)Object*.

In a **TFIBDataSet**, the batch input/output routines operate against the *QSelect* query.

TCachedUpdateStatus CachedUpdateStatus	Function
---	----------

For the currently selected record, return the current cached update status of the record.

CheckDatasetClosed	Procedure
CheckDatasetOpen	Procedure

Raise an exception if the data set is open or closed, respectively.

FetchAll	Procedure
Fetch all records now.	
Prepare	Procedure
Prepare the data set to be opened.	
RevertRecord	Procedure
When CachedUpdates are enabled, RevertRecord reverts the currently selected record to its state immediately following the last call to ApplyUpdates or CancelUpdates.	
It is basically a way of cancelling an update on an individual record, as opposed to the entire dataset with CancelUpdates.	
TUpdateStatus UpdateStatus	Function
Return the current update status of the currently selected record.	
Undelete	Procedure
When CachedUpdates are enabled, Undelete allows the user to undelete a record that has been deleted but not yet “applied”.	

10.5 Events

TNotifyEvent DatabaseDisconnecting	Published Event
TNotifyEvent DatabaseDisconnected	Published Event
These events fire when a database is about to be disconnected and after a database has been disconnected, respectively.	
TNotifyEvent DatabaseFree	Published Event
This event is fired when a Database component is freed from memory.	
TNotifyEvent TransactionEnding	Published Event
TNotifyEvent TransactionEnded	Published Event
These events are fired when a transaction is about to end and after a transaction has ended, respectively.	
TNotifyEvent TransactionFree	Published Event
This event is fired when a transaction is freed from memory.	
TFIBUpdateErrorEvent OnUpdateError	Published Event
This event is called if a particular record could not be posted or deleted during a ApplyUpdates.	
This applies only to data sets that have CachedUpdates enabled.	
TFIBUpdateRecordEvent OnUpdateRecord	Published Event
This event is called as a record is about to be posted/deleted during an ApplyUpdates.	

11 Miscellany

The file ‘FIB.pas’ contains some miscellaneous constants that are of some interest:

- `FIB_VERSION` contains the current working version of FreeIBComponents. The version number is noted by the official release date.
- `FIBLocalBufferLength`, `FIBBigLocalBufferLength`, and `FIBHugeLocalBufferLength` are used for declaring various sized buffers when the client expects to receive information back from the server. (See ‘FIB.pas’ to see the actual values.)
- `FIBErrorMessages` is a list of all of the canned error messages posted by the `FIBError` routine.

12 About the author

Gregory Deatz is a senior programmer/analyst at Hoagland, Longo, Moran, Dunst & Doukas, a law firm in New Brunswick, NJ. He has been working with Delphi and InterBase for approximately two and a half years and has been developing under the Windows API for approximately five years. His current focus is in legal billing and case management applications. He is the author of FreeUDFLib, a free UDF library for InterBase written entirely in Delphi, and FreeIBComponents, a set of native InterBase components for use with Delphi 3.0. Both of these tools can be found at <http://www.interbase.com/download>. He can be reached via e-mail at gdeatz@hlmd.com, by voice at (732) 545-4717, or by fax at (732) 545-4579.

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Acknowledgements	1
1.2.1	Hoagland, Longo, Moran, Dunst & Doukas	1
1.2.2	InterBase corporation	2
1.2.3	Contributors and supporters	2
1.3	Technical support	3
1.3.1	Bugs	3
1.3.2	Documentation	3
1.3.3	Wish lists	3
1.3.4	Help desk	4
1.4	About the documentation	4
2	Release notes	5
2.1	Planned features	5
2.2	7/14/98	5
2.3	7/8/98	5
2.4	6/3/98	7
2.5	5/14/98	7
2.6	4/13/98	8
2.7	4/3/98	9
3	Installation	10
4	Licensing	11
5	TFIBDatabase – connecting to databases	15
5.1	Properties	15
5.2	Methods	19
5.3	Events	20
6	TFIBTransaction – managing transactions	21
6.1	Properties	21
6.2	Methods	22
6.3	Events	22

7	TFIBQuery – executing queries	23
7.1	When to use TFIBQuery	23
7.2	Extended FreeIBComponents query syntax	23
7.3	Using parameterized queries	24
7.4	Properties	24
7.5	Methods	26
7.6	Events	27
8	Batching input and output	28
8.1	The TFIBBatch abstract class	28
8.2	The TFIBBatchInputStream abstract class	28
8.3	The TFIBBatchOutputStream abstract class	29
8.4	Migrating data to and from delimited files	30
8.4.1	Delimited file input	30
8.4.2	Delimited file output	31
8.5	Migrating data to and from raw files	31
9	Blob streams	32
10	TFIBDataSet – The fun part!	33
10.1	Overriding Delphi’s TDataSet abstract class	33
10.2	Making a TFIBDataSet updatable	36
10.3	Properties	37
10.4	Methods	38
10.5	Events	39
11	Miscellany	40
12	About the author	41