Firebird Conference
Prague 2006

# Developing Cross-Platform Applications with Firebird and wxWidgets

Milan Babuškov
http://www.flamerobin.org

# About the author

Education:

2001 - B.Sc. In Business Information System Engineering

2003 - M.Sc. In Internet Technology at University of Belgrade

Started to program as a 14 year old, making simple games in BASIC and later assembler on Motorola's 680x0 series CPUs. Programmed in C, Perl and Java. Now writing the code in C++ and PHP. Started to work with Informix at University, after few experiments with Paradox, MySQL and MSSQL, finally switched to Firebird. Starting from 2002, developing various software using Firebird, PHP and Apache.

Developer of open source FBExport and FBCopy tools, for manipulation of data in Firebird databases. In 2003, started a project to build a lightweight cross-platform graphical administration tool for Firebird. The project was later named FlameRobin, and is built entirely with open source tools and libraries.

Hobbies include playing basketball and writing cross-platform computer games, some of them very popular (Njam has over 60000 downloads on sf.net):

http://njam.sourceforge.net

http://abrick.sourceforge.net

http://scalar.sourceforge.net

Born in 1977. Married this year.

Live and work in Subotica, Serbia. Currently employed at small ISV company doing software for retail systems.

# Cross-platform options for desktop applications

When you install Firebird, you get the database engine and a set of tools that let you administer it. For users to efficiently access the database you need a "desktop" application. Firebird has C API available out of the box. However, there is an abundance of 3rd party interfaces for various programming languages and development environments. This paper concentrates on libraries and tools that allow user to create desktop applications that would run on multiple operating systems without (or with minimum) changes to the code. Here are the various toolkits and programming languages that are viable options today:

## *Java + JDBC*

Java is an object-oriented programming language, well proven in enterprise area. It is portable to various operating systems (Windows, Linux, Solaris, etc.) and hardware (Intel, Sparc, PowerPC). Java is semi-interpreted. The source code is first compiled into byte-code, and later run by Java virtual machine. For desktop applications it offers various toolkits like Swing and AWT. For database connectivity, there is a standard called JDBC, and there is suitable driver for Firebird called JayBird. Java is surely one of the options anyone should consider. It is easy to find programmers that know it, and the language itself has proven stable and robust. The main problem with Java shows when you want to run in on low-end hardware, as it has much bigger demands for memory and CPU power than the equivallent C++ application. Java is free as in beer, and there are even talks that Sun might open source the Java code soon.

http://java.sun.com

http://jaybirdwiki.firebirdsql.org/JayBirdHome

## *Python (wxPython + kinterbasd)*

Python is also an object-oriented language, with looser typing and more flexibility. Python is also compiled into bytecode (.pyc files) and then run. Interesting thing is that Python has been ported to Java and .Net virtual machines. It runs on Windows, Linux, Mac OS X, OS/2, etc. Python is very interesting option as wxWidgets library is ported to Python (and called wxPython). Beside that, there are bindings for other popular GUI libraries like PyQt and PyGtk. For database connectivity, Python uses the "Python Database API". For Firebird there is a Python Database API 2.0-compliant driver called

kinterbasdb. Python suffers from same problems as Java on low-end and older hardware. Python is an open-source project released under The Python License, which allows you to distribute your commercial applications without releasing the source code.

http://www.python.org/

http://www.wxpython.org/

http://kinterbasdb.sourceforge.net/

### *C++*

C++ is another object-oriented language. The main difference compared to Java and Python is that C++ compiler creates native executable files (programs). This makes them run much faster with less memory demand. C++ is ported to many (all?) platforms that exist today. There are various compilers available. Some are open source and run on multiple platforms (like GCC for example). Others are either closed-source or freeware and usually tied to a single platform. To name some compilers: Microsoft Visual C++, Borland C++, Digital Mars C++, Intel C++, Bloodshed Dev-C++, Apple C++. Good thing about C++ is that you could write code and compile with one compiler on one platform and with different compiler on another, whichever suits you best. However, the graphics systems (desktop environments) are different on each platform and have a different API. That's why we need wrapper libraries in order to "*write once, run everywhere*":

## Qt library

Qt is a library developed by Trolltech. It has a rich collection of classes that allow GUI programming. Qt is available on Windows, Mac OS X and platforms that run X11 server (Linux, Solaris, HP-UX, IRIX, AIX and many other Unix variants). Qt uses the window it gets from the host system to draw all the controls itself. That's why Qt applications don't look native, but they look the same on all the platforms. However, Qt offers "theming", so developers can create custom look of controls and windows.

Qt is dual licensed. Open source projects can used open sourced version, while closed source projects have to pay licenses. This makes Qt less attractive then other options to developers of commercial applications. On the other hand, Qt is used by the KDE Desktop Environment which is default on many Linux distributions, and has proven that it is fast, robust and reliable library with low memory requirements. Many people don't see KDE as such, which is because of layer of KDE libraries built upon Qt. Pure Qt applications (like LinCVS for example) run lightningly fast.

To design forms, you can use Qt Designer, and there is excellent IDE called KDevelop. KDevelop is part of the KDE, so it is currently not available on Windows.

http://www.trolltech.com/products/qt

## Gtk+ library

Gtk+ is a multi-platform toolkit for creating graphical user interfaces. It started as a toolkit for The Gimp (GNU Image Manipulation Program), but became so good that entire desktop environments (Gnome) are built on it. Same as with Qt, Gtk renders (draws) all of its controls on window given by the host system (Windows or X11). Gtk is written in C, which is one drawback compared to Qt: as C++ exceptions cannot propagate through C code, you have to be very careful not to throw C++ exceptions from asynchronous event-handling functions. Gtk comes in two versions Gtk1 and Gtk2. Gtk1 is an older library with limited set of widgets. However, it is really fast and low memory requirements. It is suitable for low-end machines and applications where eye-candy is not an important issue. Gtk2 is a modern set of widgets and it allows theming and has Unicode support. Important thing about Gtk1 is that is has much less dependencies, so it is easier to install and maintain. If we compare Qt and Gtk, one can say that Qt works as fast as Gtk1 and looks as good as Gtk2 on screen.

Gtk is an open-source library, released under LGPL license. This allows anyone to build commercial applications without having to pay or release the source code. Beside that, many people choose Gtk as it looks native on Gnome which is the default desktop environment on RedHat/Fedora and Ubuntu, two among leading Linux distributions. Works on Windows, Linux, FreeBSD, Solaris and should work on any platform where you have X11 window system and GNU GCC compiler available.

To design forms, you can use Glade, which is a standard builder for Gtk applications.

http://www.gtk.org/

## wxWidgets

wxWidgets is a cross-platform toolkit that exists for more than ten years and provides ports to many platforms. It's uncommon in sense that it wraps around native toolkit on a platform rather than drawing all the controls itself. On Windows it uses the Windows API. On Linux, FreeBSD and Solaris it gives option between Gtk (both Gtk1 and Gtk2), motif and X11. On Mac OS X it uses Carbon, and Cocoa port is on the way. All this makes the applications using it look and feel native on each platform. In fact, those are native applications, which just use the additional layer that makes platform differences almost transparent. All this make easier for users to accept the application, since everything looks familiar.

Beside basic graphic widgets, wxWidgets also offers various components needed in application development like: clipboard support, drag and drop, multi-threading, image loading and saving in various formats, HTML viewing support, printing, etc. All of those wrap a native way to do it on each platform. If something is missing, wxWidgets provide their own replacement if possible. Good example of this is tree widget on Gtk1, which Gtk1 does not provide, so wxWidgets draw their own generic version of the control.

In most cases, when working with wxWidgets, you don't have to care which platform the code it written for. For example, Gtk1 is partly compatible with Gtk2 so some applications can be compiled against both libraries. wxWidgets introduces a set of #ifdefs that mark the different parts, so programmers don't have to pay much attention and simply pick the one they want. Some of the widgets that don't exist in Gtk1 are emulated by wxWidgets. With wxWidgets, you first build the wxWidgets library against desired toolkit (WindowsAPI, Gtk1, Gtk2, X11, Motif, Carbon, etc.) and then you link your application against wxWidgets library. In most cases you don't need to change code at all in order to work on different platform.

wxWidgets can be built with various compilers. At FlameRobin project, we are successfully using GNU GCC and Microsoft Visual C++ from the very beginning. You can also use GCC on Windows using MinGW environment. Borland's Free C++ Compiler is also very good for smaller projects, while linker sometimes chokes when project gets really big (100+ units). It can also be built with DJGPP, DigitalMars, CodeWarrior, Sun CC, Symantec C++, Watcom C++, IBM Visual Age and AIX Compiler (xlc), although these are not extensively tested.

There are various closed and open source applications that allow you to design wxWidgets forms:

### *wxFormBuilder*

wxFormBuilder is an open source tool written in C++ using wxWidgets itself. It still young but already very powerful. It is a pure form designer, without support to edit event handlers. However, it is really fast and robust. Newer versions have all the common widgets available in it, and even some contributed ones.

http://www.wxformbuilder.org/

### *wxGlade*

wxGlade is an open source form designer written in wxPython (so it works on all platforms supported by wxWidgets). It was one of the first free tools to offer quality editing, but development is slow. It marks the places in code where developers should, and where they shouldn't write their code. It does not create event handlers automatically.

http://wxglade.sourceforge.net/

### *wxDesigner*

wxDesigner is a commercial form designer. It is one of the most complete tools, allowing you to build and test dialogs and write code for event handlers. Event handlers are not as easy as Delphi's double-click the control, but it is still quite usable and much easier than adding those by hand. It also features built-in editor, so it can almost be used as IDE (there are no options to compile, run, debug the code).

http://www.roebling.de/

### *VisualVx*

VisualVx is a freeware form designer and a RAD tool. It supports event handlers. The tool is Windows-only and doesn't work on other platforms. The main problem is that the

code it generates is somewhat hard for humans to read as it has too many code guards and makes obscure variable names.

http://visualwx.altervista.org/

### *DialogBlocks*

DialogBlocks is a commercial IDE and dialog editor written by Julian Smart, creator and lead developer of wxWidgets. It is cross-platform (although there are some glitches on Windows98, it works fine on modern operating systems). DialogBlocks allows editing of code and automatic creation of event handlers, which saves developers from tedious copy/pasting of event handlers. This is probably the best tool available for wxWidgets development.

http://www.anthemion.co.uk/dialogblocks/download.htm

### *wxDev-C++*

Of all the free tools, wxDev-C++ looks like the best one. It it both form-designer with event-handler editor and an IDE. It's IDE is closest to Delphi. You can design dialogs and frames, double click on buttons and other controls to create event handlers. Only drawback is that it is a Windows-only tool. This means that you would do development on Windows. However, the code it generates can be compiled on all other platforms as well. This basically means that you design dialogs and write code on Windows, and later you take an re-compile generated code on target platform. If you're looking for free integrated IDE and RAD tool, wxDev-C++ is the one. If you want to work on Linux, I recommend you use wxFormBuilder and write the event handlers by hand.

http://wxdsgn.sourceforge.net/

Beside wxDev-C++ and DialogBlocks, you can use various other IDEs for wxWidgets development:

- CodeBlocks - http://codeblocks.org/
- MinGW developer studio - http://www.parinyasoft.com/
- Eclipse - http://www.wxwidgets.org/wiki/index.php/Eclipse
- Chinook - http://www.degarrah.com/products/chinook/

# Cross-platform options for C/C++ access to Firebird

## *ODBC*

Open Database Connectivity (ODBC) is an industry standard way for accessing database management systems. There are ODBC implementations for various operating systems: Microsoft ODBC for Windows, UnixODBC for Linux/UNIX, iODBC for Mac, Solaris, FreeBSD, etc. To access database from application, wxWidgets provide wxODBC set of classes:

http://www.wxwidgets.org/wiki/index.php/ODBC

In case you don't use wxWidgets, there is libodbc++ library:

http://libodbcxx.sourceforge.net/

As for Firebird, there are various ODBC drivers, some open source (like the one supplied by the project) and some commercial. The quality and features differ, so better try it out before going deep into development.

http://www.firebirdsql.org/index.php?op=files&id=odbc

ODBC is great if you need to access other non-Firebird databases. However, supporting many DBMS-es meant it covers the lowest common denominator, so most of the advanced stuff (things provided by Firebird's Services API) is not available. Also, ODBC seems much thicker layer around Firebird C API than lightweight IBPP.

## *C API*

Firebird C API comes with default installation. There are even examples in "examples" directory. This is the fastest and most efficient way to access the database. However, it is also the most complex and thus most error prone. C API is useful if you don't want to depend on C++ library and compiler, especially if you're building a driver for other programming languages. But, if you're building a C++ desktop application, IBPP is much better option...

## IBPP

IBPP is a thin C++ wrapper around Firebird C API. It is not a simple translation of functions from C to C++, but rather nice object-oriented interface to database features. IBPP offers full access to all Firebird's functions, so you can create, drop, backup, restore and fix databases, manage users, etc. IBPP is easily integrated into your applications, and highly portable: works on Windows, Linux, MacOSX, FreeBSD, Solaris, and probably more. The library itself uses standard C++, so you don't need to install anything else.

IBPP is currently Firebird-only, so you're locked into Firebird if you decide to use it. One of the great advantages over C API is also that you're safe from API changes. If some future versions of Firebird decide to change the C API (which is not likely, but still) you don't need to change your code as the changes would be done in IBPP itself.

IBPP has an open-source license, which is very simple and allows the use of IBPP in both open and closed-source projects without any drawbacks.

Author note: My experience with IBPP has been excellent. I'm using it since 2002 in various commercial and open source projects. It has proven to be rock-solid and support from lead developer Olivier Mascia is great.

http://www.ibpp.org/

## SQLAPI++

SQLAPI++ is an interesting option. It's a wrapper similar to IBPP, but it can be used against different database systems (Firebird, Oracle, MSSQL, Sybase, DB2, MySQL, etc.). The main problem is that Firebird specific functions (like creating database for example) are left for developer to implement in native C API. Just like with ODBC, you get multiple DBMS support, but you're cut off from functions offered by Firebird's Services API. SQLAPI++ is shareware, priced at $249 at the time of writing.

http://www.sqlapi.com/

## DatabaseLayer

Some time ago, an interesting project has been started to provide native access to various

DBMS systems from wxWidgets. This is a nice alternative to wxODBC. It currently supports Firebird, MySQL, Postgresql and SQLite. You can also hook up and ODBC data source to it. Once more, there is no interface to Services API. DatabaseLayer is an open source project, released under same license as wxWidgets.

http://wxcode.sourceforge.net/components/databaselayer/

## Setting up wxWidgets application with Firebird support via IBPP

In the past, setting up wxWidgets builds was a matter of manual setup of makefiles. While it is still possible today, it's a tedious process, especially if your project has a lot of files and you build on multiple platforms. Since version 2.6 of wxWidgets, a new build system was introduced. It is called Bakefile, and although it's primary purpose is to serve wxWidgets development, it is a standalone open source project:

http://bakefile.sourceforge.net

Bakefile allows you to maintain a single definition file for all platforms and compilers. Simply edit the textual (xml-like) definition file and add your source files (.cpp/.h), resource files (icons and stuff) and installation files (files and directories that need to be installed with you application). Bakefile uses this definition file (which has .bkl extension) to create makefiles for various compilers, and also project files for MSVC.

Setting up bakefile build system for your own projects in not an easy task. The only real example you have is build system used for wxWidgets, but it is somewhat specific as it is oriented around building the library instead of building an application. Based on our experience with FlameRobin, we created a minimal Bakefile-based build system for anyone who wants to build wxWidgets application and use IBPP to access Firebird from it. This system shares FlameRobin license: Expat, also known as MIT license. It's a BSD style license and basically it means that you can take all the code, do whatever you want with it, use it in either open or close sourced project, and you don't have to give anything back (neither code or money). The only requirement is that you keep our copyright notice in files. Of course, for you own files and modifications you make, you can put whichever license you wish.

Minimal build system package should be in same package as this document. If it is not, or you wish to fetch the latest version, you can always check it out from FlameRobin's

Subversion repository:

svn co https://svn.sourceforge.net/svnroot/flamerobin/trunk/template-project

Once you download it, read HOWTO.txt for step-by-step setup.

This is just a starting point, after which you can add your own files. Beside minimal system, entire FlameRobin source code is available, so you can learn how to do some things, and you can even take some ready-made classes and functions and shorten the time needed to build the Firebird-based application. There are also plans to reduce the dependencies of some FlameRobin classes on others, so you could, for example, take ready-made DataGrid class without having to include 5-10 other files from the project.

## Comparison with Delphi

Delphi is a well known tool that set up a standard for rapid application development (RAD). It is a tool that most Firebird developers are using when developing desktop applications. Therefore, Delphi is a very good reference point to compare any other alternatives with.

### *Single integrated tool vs many separate specialized tools*

If you develop on Windows with wxDev-C++, then there isn't too much difference. IDE looks very similar to Delphi, you can design, write code, compile, run and debug from it. However, on Linux you need to use various tools in the chain. You can use the wxFormBuilder to design dialogs, and an IDE like CodeBlocks to write code, build and debug. Or, you can follow the old Unix tradition and use separate tools for each task. For example, I manage a project by having good structure of directories, I edit the code with Kate editor and use make directly from shell to build the executable. For debugging I use Kdbg. That is one example of Linux tool-chain used to create wxWidgets applications.

### *Dialog definition files*

Delphi uses .dfm files. wxDev-C++ has .wxform files which have a similar layout

(wxDev-C++ is written in Delphi itself). Most other tools use custom files in XML format, or wxWidgets special XRC format (XML-like) which enables loading and creating controls at runtime. A lot of tools keep XML representation only for design purposes and final product is either C++ or XRC file.

## *No properties*

Unlike C++ Builder, wxWidgets are pure C++, so there aren't any non-standard extensions like Borland's properties. You have to use GetXXX/SetXXX methods to access class data.

## *Event macros*

Event handling in Delphi is done behind the screens. In wxWidgets there are two approaches: event macros and dynamic binding with Connect() functions. Event macros look like this:

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
     EVT_CLOSE(MyFrame::OnClose)
     EVT_BUTTON(MyButton_ID, MyFrame::OnMyButtonClick)
END_EVENT_TABLE()
```

Tools that support event handlers add these automatically for you. These event handlers are needed to map the event (EVT_BUTTON is a button click for example) with event handling function: OnMyButtonClick in the example above. In the above example EVT_CLOSE is bound to MyFrame directly, but button click event is bound to one of the child buttons. Therefore we need to supply the button ID of that button: MyButton_ID. MyButton_ID is an integer value usually specified somewhere in class' the .h file. The event handling function looks like this:

```
void MyFrame::OnMyButtonClick(wxCommandEvent& event)
{
     // write your code here
}
```

### *Data aware controls*

IBPP is a thin C++ access layer. This means that you have classes like Database, Transaction and Statement, but there are no advanced concepts like Dataset or data-aware GUI controls. This means that more things have to be programmed by the developer. If you use wxODBC, there is wxGridTableBase, which can be used as a data source to fill visual grids, but that's about it. FlameRobin's code can help here, as there are classes that allow you to display data in the grid, and also to convert various data types to human-readable strings.

## Solving typical problems

When people are coming from single to multi platform development, there are issues awaiting which they aren't aware of. While wxWidgets tries to hide all the platform differences behind their classes, there are some things worth knowing before you run into brick wall with your application:

### *Layout*

One of the main problems with cross-platform environment are differences in the graphics systems on which application runs. Displays on one system can differ in DPI settings or number of colors, but with multi platform other things are added, like different sets of fonts for example. While you could force some users to use some settings you built your application for, it is generally not a good idea. This is why wxWidgets have gone a long way to stay away from fixed coordinate systems when designing dialogs.

In wxWidgets, all the layout should be done via sizers. Sizers make the dialogs and controls resize themselves appropriately in order for all the text to fit nicely on the screen. Of course, you can define rules how do you want layout to behave: which is the proportion of some controls, which controls you wish to grow if more size is available, etc. There is a lot of sizer classes, each with its own purpose. Most commonly used is the wxBoxSizer, which is a generic control that distributes controls vertically or horizontally. It is sufficient for many uses as you can combine multiple sizers one inside another (they don't add any visual spacing unless you explicitly tell them to). For most of the dialogs, wxFlexGridSizer sizer is appropriate solution. It can have fixed and growable rows or columns, and fits most of the needs. For users with special needs, there is always wxGridBagSizer, which is as flexible as editing HTML tables, i.e. any control can span multiple rows or columns - exactly as you want it.

It takes some time to get used to the sizers and the way they work, but it pays off. Never again would your dialogs have controls that are cut off, or text being too big to fit in. This also simplifies the development of applications which are translated in multiple languages, as you don't have to worry if text of some translation is going "out off the bounds".

Another important issue with layout is the native look and feel of applications. Microsoft, Apple and Gnome team define the Human Interface Guidelines (HIG) standards for each platform (Windows, Mac OS X, Linux and others that use Gtk+ respectively). These define the common layout of controls themselves (is the Ok button left and Cancel right or vice versa) and the spacing among them as well. To help with some of that, FlameRobin project has built a set of StyleGuide classes, which are also part of minimal build system, so you can reuse them into your projects. They return spacing constants for all of those platforms. Here are some more details about usage:

http://www.flamerobin.org/dokuwiki/doku.php?id=wiki:platformspecifics

## *Conversion between string types*

Since wxWidgets use wxString class for all string operations, it is a challenge to use any external library which uses standard C++ string (std::string). It is not easy for beginner to handle as you can build wxWidgets in both ANSI and Unicode more, so translation needs to take care of wxString representation. We have faced this problem in FlameRobin and created two utility functions: wx2std and std2wx which allow easy transliteration of strings. Just include StringUtils.h header file (supplied in template-project) and enjoy.

As for strings them selves, there are two kinds of strings in wxWidgets: ones that should be translated in localized versions of application, and ones that should not. Both need to be enclosed in string macros, so that they work properly for both ANSI and Unicode build. Translated strings should be enclosed with _(), and others with wxT(). For more info, read the following document:

http://www.flamerobin.org/dokuwiki/doku.php?id=wiki:wxstring

### *Fatal exception handling*

In C++ if an exception goes uncaught, the program crashes. While development tools like Delphi and C++ builder protect you by installing their own fatal exception handler, with wxWidgets you have to do it yourself. It is not hard, as wxWidgets provide cross-platform wrapper around it, and you can see it implemented in main.cpp of the minimal build system. However, one thing is important to remember: Gtk+ is written in C and its event loop is a C code. If some of code in your event handlers throws an C++ exception it will go back through a layer of C code, and cannot be caught by general try..catch handler with is put around application startup code. So, make sure you protect your event handlers with try..catch block if you think there is any chance of exception being thrown.

### *Thinking in cross-platform terms*

There are many things that are abstracted by wxWidgets, so it is worth mentioning in order that you don't start to reinvent the wheel. Standard paths (application installation directory, etc.) are covered with wxStandardPaths class. There is wxFont class, where you can say what type of font you want, and you'll get the appropriate one on the given platform. So, you don't say: I want Arial, but you say: I want Roman or Sans-serif, or something like that. Handling multiple displays (which is quite different on Linux and Windows) is encapsulated in wxDisplay class, and you should use off-screen coordinates to access the second display.

As a final note, make sure you recompile your application on every target platform each time you do some major change. There are things like, for example, handling key presses, that can yield surprises, as some keys are reserved for operating system use. Tip: stay clear from F9-F12 section.

## Conclusion

If you plan to develop database applications that need to run on multiple platforms, wxWidgets is one of the best choices available. It is robust, scalable and has much lower memory and CPU power requirements than most other alternatives. It is still not a the RAD level of Delphi, and it lacks easy to use data aware controls, but you get cross-platform functionality for free, so it isn't hard to put in a little bit more effort. Being open source and having a lot of active developers for over ten years shows that this framework would not become obsolete as happened to many others which were bound to a single platform or a single company.

# Table of Contents