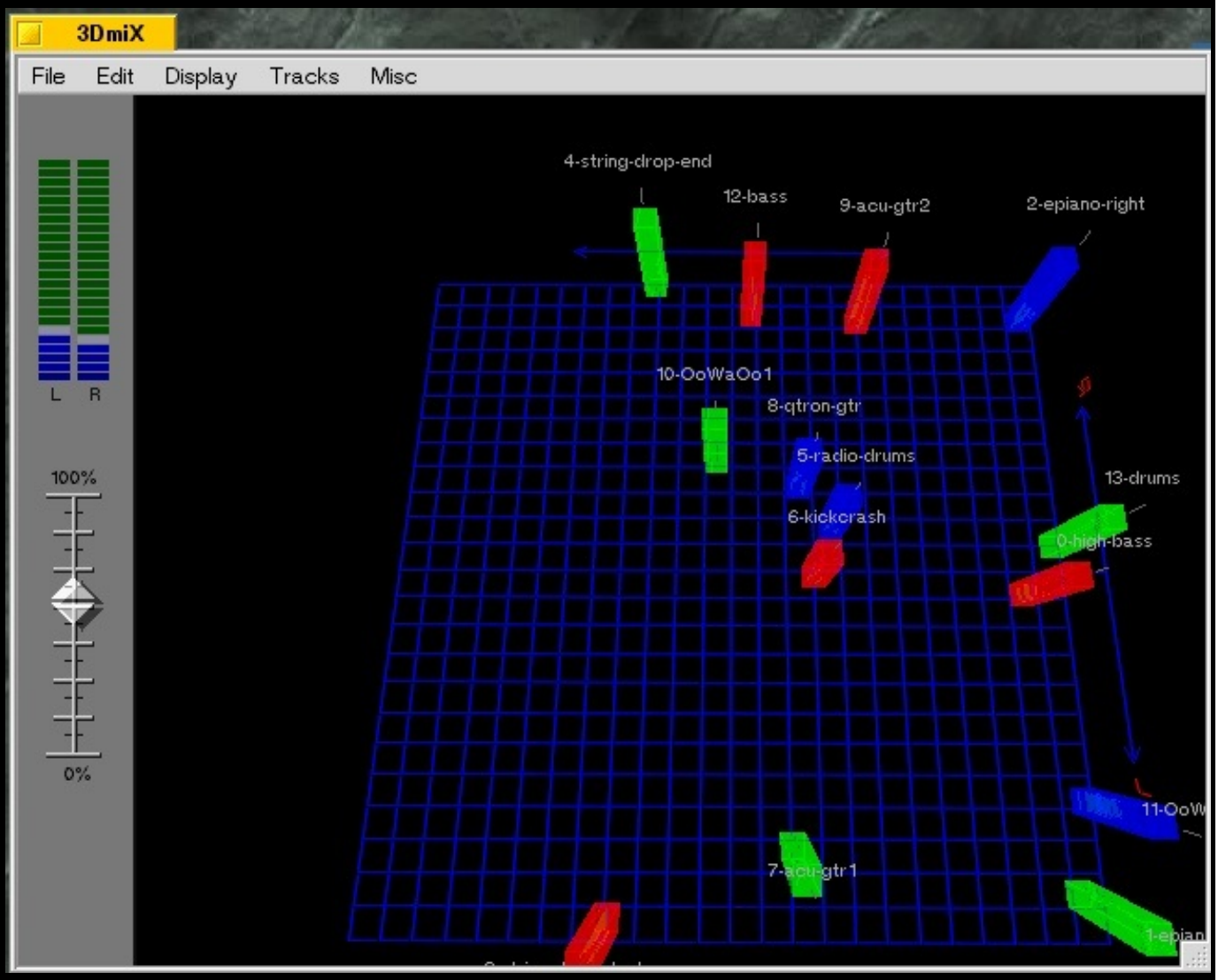
BeOS Programming For The Beginner

Return to The Archives

by John Kenneth Grytten (13 April 2000)

Introduction

My motivation for writing this article is to open your mind to BeOS programming. The key reasons for working in the BeOS is its responsiveness and excellent media performance. I will try to give an overview of developer tools and explain fundamentals of C++ development in BeOS. For coverage on BeOS 5, read BeNews Extra: BeMAGAZINE.



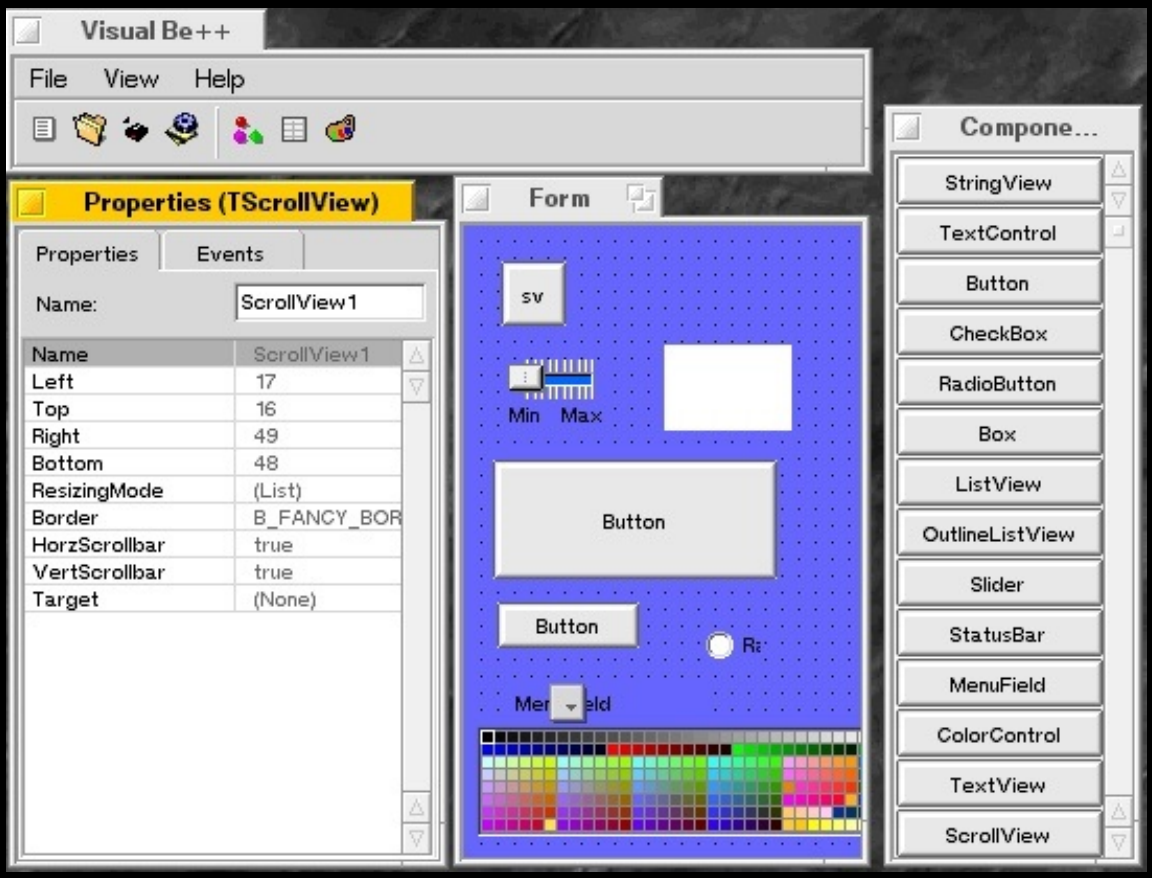
3dmiX - one innovative BeOS app.

An Overview Of Development Tools

The whole point of FreeBe (aka BeOS 5 Personal Edition) is to make it as small as possible for people to download. For this reason you won't find the developer tools included, but you can download the file "BeOS5DevTools.zip" (about 20 MB) separately from various mirrors.

Based on the open source EGCS tools from Cygnus, BeOS developers enjoy much of the same development environment as developers on Linux. In fact, GCC (GNU Compiler Collection) is one of the the most popular compilers on any platform. Be use Metrowerks' tools on the PowerPC platform, but thanks to the BeIDE (Integrated Developer Environment) BeOS native programs are source compatible between Intel and PowerPC. To make life easier, Be has also made available CrossDevelopment tools. In fact, 99% of the BeOS source tree is portable C and/or C++ code, and that's important to avoid difficulties in the transistion to new architectures. As an illustration, Be had very great success in doing a complete port from PowerPC to Intel in only about a year.

Looking at what other tools are available, you'll find many third party utilities and ports of popular crossplatform high-level languages. There is the NASM assembler for Intel, the Squirrel language for easy GUI programming (Logo dialect) without learning C++, Perl and REBOL for general scripting, PHP 4.0 and BeKaffe (Java VM) for client-server development -- and many more... I should probably mention Python and the Hugo language (for game development). Go to BeBits, your definitive guide to BeOS applications to find all this and much more. Personally, I would like to point out that the crossplatform Simple Directmedia Layer from Sam Lantinga has full BeOS support. SDL makes it relatively easy to program games and demos, as known from the demo scene.



Visual Be++ - drag and drop UI components like in Delphi

Be has announced that they are working with Sun Microsystems to incorporate the Java(tm)2 Platform, Standard Edition (J2SE) and PersonalJava(tm) technology into BeOS. Hopefully this will bring full Java support to BeOS users, and BeKaffe will no longer be the only Java virtual machine available. Java will bring the free visual development tool VisualAge from IBM, as other "pure java" apps. A very promising commercial BeOS native visual programming tool called Visual Be++ is being developed by Kelly Schrock and will allow developers to approch programming like in Delphi or Visual Basic, only in C++. There are also free tools, for instance BeBuilder and Interface Elements, that make GUI creation easier.

C++ and BeOS

Most Be developers don't use visual programming tools. The good news is that BeOS really delivers on the concept of object-oriented programming using C++ the way it is intended. For information on "Learning C++ as a new language" and opinions on the use of libraries in C++ from the creator of the language, I suggest you visit C++ Answers From Bjarne Stroustrup and his homepage. Many thanks to Slashdot for bringing these informative and inspiring answers. BeOS is written in C and C++. To make use of the various kits described in the the Be Book you are (at least for the moment) forced to develop programs in C++ using the concept of inheritance. Each object in a program is usually put in a separate source file with a descriptive name. You don't really need to learn everything in C++, but you absolutely need to know the basic OOP concepts. In fact, according to Bjarne Stroustrup, learning programming with C++ and a few good libraries (like in the Be API) can be very effective, because you can focus on the central ideas and introduce additional concepts as you learn the basic rules. Sadly, I don't have time to write a complete course in learning C++ using the Be API - but I believe that could be a very successful for many (instead of getting used to bad habits on other platforms).

Free development tools, the BeIDE and the elegant Be API (Application Programming Interface) together make C++ development in BeOS a breeze compared to other operating systems. I often compare developing in BeOS like programming in Java, only a bit more difficult and a lot more rewarding. With C++ you enjoy the superior performance of the BeOS system and the power of language flexibility, which sometimes is needed for large or special projects. Remember that real BeOS developers inherit functionality in existing components that are specially optimized to work with the rest of the system. I like to describe the popularity of the Be API in the developer community much like the concept of LEGO among "kids of all ages". That's right, it's easy to understand and you can build virtually anything you can think of with a set of basic yet very functional pieces. Both LEGO and the libraries of the BeOS consist of high quality parts that are designed for *seemless integration, high flexibility and extensive reuse without problems* - in the computer industry this translates to no fuzz and binary compatibility -- that you don't need to recompile a program because of OS updates. Of course this has some implications that developers should know about.

OpenGL - Be has officially licenced OpenGL from Silicon Graphics

The Be API provides functionality through fundamental "kits" like the The Application Kit and The Interface Kit, but also in the more specialized form of the The Media Kit and The OpenGL Kit. The rest is listed in the Be Book. The fundamental kits offer what you need for a basic application (advanced drag and drop included). The Media Kit gives you access to a low-latency network of media processing add-ons. Applications made with The Media Kit can support all media file formats and codecs supported by the OS, and you get advanced timing synchronization -- down to the lowest latency allowed by the hardware. Unlike in other operating systems, the kernel does not get in the way. The Media Kit allows you to play and record any media format (video and audio), you can even create your own media nodes to handle new sound cards or even new types of digital media besides video and sound (think more interactive content that requires accurate timing). In BeOS 5 there is the revamped Midi Kit for your needs in that area. The OpenGL Kit is currently being reworked to support hardware accelerated OpenGL on a wide range of video cards. The Network Kit offers high level interface to network programming. Because of its clean interface, Be is able to reimplement BONE (BeOS Networking Environment) in the kernel (the net_server lives in user space currently) without developers having to do anything at all to get the promised 2000% speedup in networking performance.

Working With The Be API

To make this easy, I'll concentrate on one fundamental concept of C++ development in BeOS: inheritance. To understand this you should at least know what OOP is and how it generally works. Let's first consider the following code:

```
#include <Application.h>

int main()
{
  // create a new BApplication object (on the stack)

  BApplication myFunkyApplication("application/x-vnd.funkyapp");

  // enter the message loop

  myFunkyApplication.Run();

}
```

Every application needs a BApplication object. Calling the Run() method magically connects your program to the operating system. Ignore the argument (MIME type) for now - let us focus on what is really important.

When you run this app, you will notice that it appears in the deskbar and you can kill it using the GUI, for example the Team Monitor. If we add new threads to our program they will be part of our applications team. Run() does not return before the application is told to quit. This is special to the BApplication object. Right now, we only have one thread in our program and that is the message loop. That means the application is given the possibility to send and receive BMessage objects. We can receive messages in the shape of BMessages from the OS but also directly from other apps.

At the moment we have not implemented any functionality to respond to any messages, except for the Quit() that is built into BApplication. As you have probably heard, the BeOS is pervasively multithreaded. That means, unless we are writing nonsense code like this or traditional console apps, we must use at least two threads to do anything useful! In other words, the system is designed from the ground to take advantage of multiple processors. Let me change the code to open a simple window.

```
// file: funkyapp.h

class FunkyApplication : public BApplication
{
public:
  FunkyApplication();
};

// file: funkyapp.cpp

#include <Window.h>
```

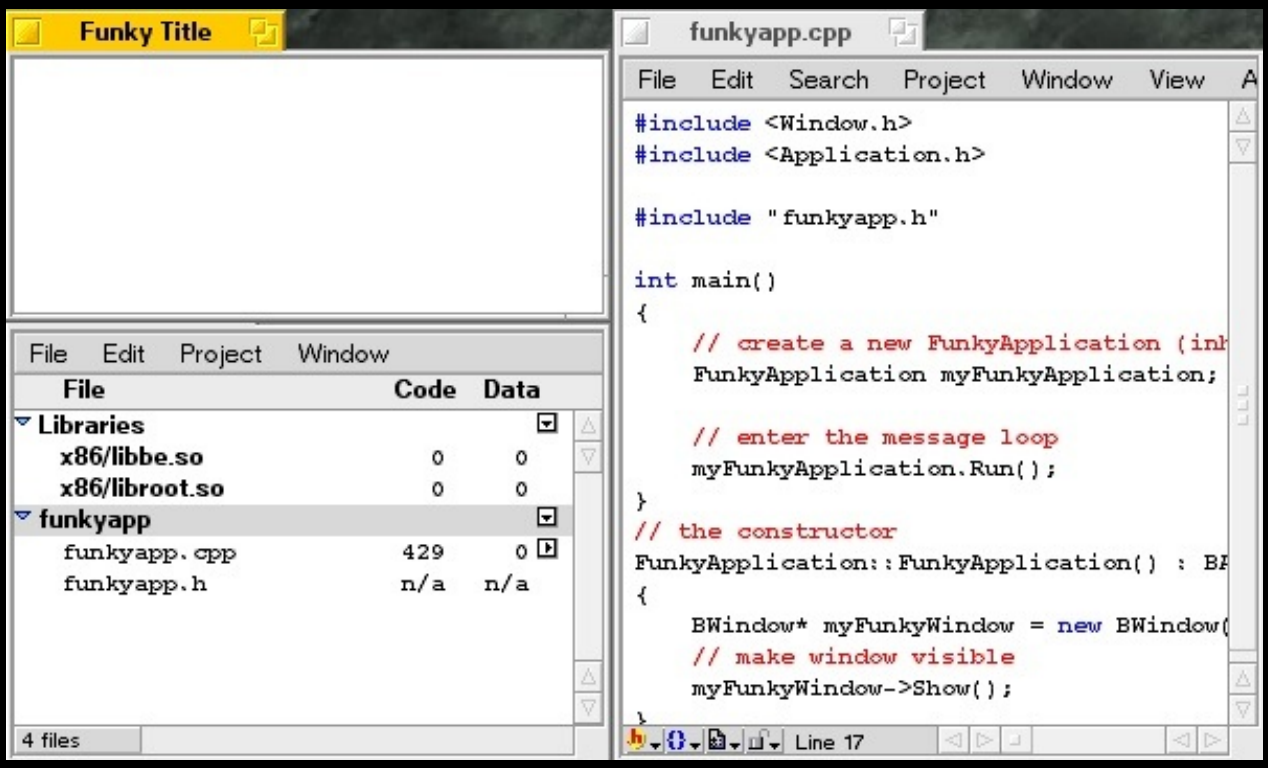
```
#include <Application.h>

#include "funkyapp.h"


int main()
{
  // create a new FunkyApplication (inherited from BApplication) object

  FunkyApplication myFunkyApplication;

  // enter the message loop

  myFunkyApplication.Run();

}

// the constructor

FunkyApplication::FunkyApplication()
  : BApplication("application/x-vnd.funkyapp")
{
  // we cannot do  BWindow myFunkyWindow(BRect(10,10,210,110), "Fu
  // like we did with BApplication, because Show() will return immediate
  // stack, the memory used by myFunkyWindow would be released whil

  BWindow* myFunkyWindow = new BWindow(BRect(10,10,210,110), "F

  // make window visible

  myFunkyWindow->Show();
}
```

Now, I have split the code in a header file (funkyapp.h) and a source file (funkyapp.cpp). While this is not so important in this case, you must know it is very important to write clean code and follow coding guidelines as a C++ programmer to keep the code maintainable.

In the header file, we define a new class FunkyApplication that inherits from BApplication. We also specify that we will add some code to the constructor in the source file.

In the source file, you see the constructor. This is pretty much the receipe for picking up existing objects, spicing them up and using them in your own app. Here I create the myFunkyWindow without inheriting anything, just like I did in the first example with BApplication, but in a real app I would put it in separate files, namely FunkyWindow.h and FunkyWindow.cpp using the same technique that I applied to BApplication to add the

BWindow. To add controls and other BView derived elements to a BWindow, you should do that in the constructor of the window. You should get the idea pretty quick, but feel free to experiment with code that crash in various ways.



The Be Integrated Development Environment and our funky application

Running the app opens a window with the title "Funky Title" and the properties and flags as defined. Reading in the Be Book you will realize that BWindow and BApplication are both derived from BLooper. You should note despite this, BApplication takes over the current thread while BWindow does not.

To do more advanced stuff with a GUI, there is really no way to learn but reading the Interface Kit introduction. If you have understood everything in this article you should be well prepared for self studying the Be Book. One great way to learn is to look in the folder /boot/optional/sample-code/ of your installation (first BeOS5DevTools.zip need to be extracted to /boot) and modify the sample code to experiment with the Be API as you read about the topics that interest you. You could for instance begin with HelloWorld and try to change window and font properties. I should mention there is a book called Programming the Be Operating System that gradually adds elements to the user interface, but I guess most existing developers have learned from the Be Book and a separate book on C++. The hardcopy version of the Be Book is called the Be Developers Guide, but it is not being updated with the latest changes.

Conclusion

While this article does not contain much more than the fundamentals and some clues on where to search for more info, it is my hope that you are now more hungry to learn more. If you are still wondering why BeOS 5 is a great OS, read the BeMAGZINE mentioned in the introduction. It's a great magazine with lots of info so I see no reason to repeat that here. C++ development in BeOS takes courage and motivation for doing things that is not possible on other platforms. Check out the latest BeOS headlines for what is currently happening in the community, if you don't find any interesting projects to support you should now be able to start up with your own. There is a lot more than programming skills to development, but you need to understand what it will take for your idea to materialize. Visual programming can

make the design of your application more effective, but you should also rethink your choice of operating systems for development -- and thus the very building blocks for your application. Now, play with the sample code and you may get rich and famous in the emerging market of the BeOS!

BeOS5-DevTools.zip comes with plenty of source code to play with

"At a risk of being called sexist, ageist and French, if you put multimedia, a leather skirt and lipstick on a grandmother and take her to a nightclub, she's still not going to get lucky."
- JLG comments on the adding of features to Windows in a New York Times Interview.

This quote from The quoteable Jean-Louis Gassee on BeDope.