Workshop: Platform Port for C++ from Borland Delphi to Kylix 3

# An Unlikely Couple

For C++ projects the Component Library for Cross-Platform (CLX) is the bridge that connects the Windows/Delphi and Linux/Kylix worlds. This article looks at a practical case: porting a Delphi and C++Builder program to Linux, a Delphi program to Kylix for C++ and a Delphi VCL program to CLX. All in a simple and timely fashion.

BY JOCHEN STÄRK

Many major Open Source projects such as Apache, Open Office, or Mozilla are not restricted to a single platform or processor type. Developers recognize the importance of providing multi OS capabilities for their programs. And porting commercial programs can open up new markets. One approach to this is a combination of Delphi/Kylix.

Kylix 3 finally allows developers to port Delphi 6/7 and C++Builder 6 projects (using Borland's C++ IDE) from Windows to Linux. And what's more, you can develop new projects in a manner that supports quick porting. To the best of our knowledge the longest period required to talk a clean Delphi CLX project into compiling on Linux was two hours, as reported in the Colliers Case Study [2].

This workshop describes how existing Delphi/VCL (Visual Component Library) projects can be converted to CLX (Component Library for Cross-Platform) projects and how CLX projects can be compiled on other platforms. For the first section – porting a thread sample – we will first need to load a small Delphi VCL project in Kylix for Delphi.

In section two we will be looking briefly at porting a Delphi CLX project to Kylix, and using Kylix to compile a simple Delphi CLX program. And finally, we will be using C++Builder to write a VCL program, converting it into a CLX program and including it in the Delphi CLX port.

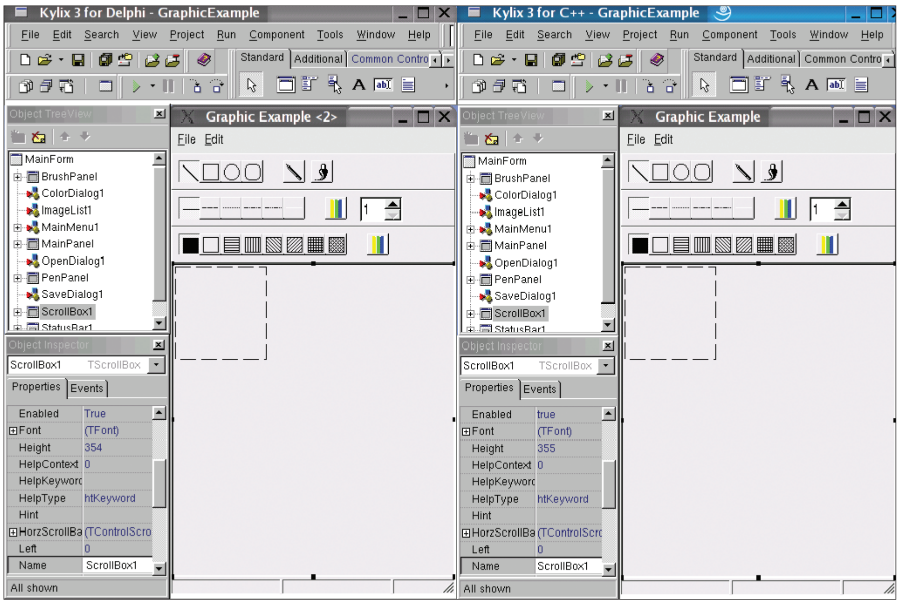
## Choice of Developer Platforms

It is preferable to use Linux CLX for project development, and then convert to Windows, rather than the other way round. Kylix is not only cheaper than Delphi or C++Builder – and to be accurate, any comparison should in fact be based on the total price of both Delphi and C++Builder – there is less danger of producing platform specific code, using Kylix.

The current Delphi and C++Builder versions contain a license for their Kylix counterparts: Delphi for the Kylix Delphi IDE and C++Builder for the C++Builder IDE. In fact, the Kylix product and the free Kylix Open Edition for use in Open Source projects include both IDEs.

There is a tool for importing C++Builder project files to Kylix, but there does not seem to be a tool for converting VCL source files to CLX. Depending on the project, conversion may involve as little effort as adding a few *Q* s to unit names, or as much as completely exchanging the database and the intra/inter-process communication (CLX supports neither the Borland Database Engine BDE nor proprietary Windows message formats).

Manual conversion is described, for example, by Dr. Bob (see [1]) and of



| | |
|---|---|
| Enabled | True |
| Font | (TFont) |
| Height | 354 |
| HelpContext | 0 |
| HelpKeyword | |
| HelpType | htKeyword |
| Hint | |
| HorzScrollBa | (TControlScro |
| Left | 0 |
| Name | ScrollBox1 |

| | |
|---|---|
| Enabled | true |
| Font | (TFont) |
| Height | 355 |
| HelpContext | 0 |
| HelpKeyword | |
| HelpType | htKeyword |
| Hint | |
| HorzScrollBa | (TControlScr |
| Left | 0 |
| Name | ScrollBox1 |

**Figure 1: Kylix for Delphi and Kylix for C++ look nearly the same – The Delphi version on the left, the C++ version on the right. This enables developers to work on differing platforms without the expense of retraining**

course the manuals ("Delphi 7", chapter 15.2, "C++Builder 6", Vol. 1, chapter 14.2). Converting the source code will not create a "native Linux version" but, under perfect circumstances, a CLX version with non-platform specific source code.

Although the change may appear insignificant at first, note that Windows relies on *dfm* files being renamed to *xfm* to decide the kind of code the IDE will generate from the form definitions contained in the files: CLX code for *xfm* and VCL code for *dfm* files. When porting, renaming the files will always be the first step, which is followed by changing *{$R *.dfm}* to *{$R *.xfm}* in the appropriate source file.

## Name Game

If you forget to rename these files, and then go on to perform valid modifications of the source files, stipulating *QForms* instead of *Forms* for example, Delphi and C++Builder will add the "missing" forms unit or header file to the source code next time you save the project.

The reason for this is that both assume a VCL project with a form, based on the *dfm* file suffix, and this necessitates using the *Forms* unit. However, if you remembered to rename the file, both Delphi and C++Builder on Windows will use some of the required CLX units when you load the file. Kylix always uses CLX units.

## Porting the Thread Sample Program

At this year's CeBIT I was regularly asked whether vBuilder multithreading code is portable, or necessitates writing special forks or similar for each application on Linux. The answer is, developers can use the glibc function on Kylix, more specifically, they can use *fork*. Of course, this kind of thing should be placed in an *#ifdef* block for this platform only. However, Kylix CLX thread classes can also be used in Delphi and Kylix for C++.

Here is some proof-of-concept code that simultaneously converts from VCL to CLX: Copy *Delphi7\Demos\Threads* to your Linux partition and rename *thsort.dfm* to *thsort.xfm*. Open *ThSort.pas* in an editor and change *{$R *.dfm}* to *{$R *.xfm}*. Additionally, rename *thrddemo.res* to *ThrdDemo.RES*. You can now open the file in Kylix for Delphi. The borderstyle message that appears is caused by a missing CLX forms property and can be ignored.

When you attempt to launch the program, the first thing you will notice is the missing *Forms* unit in *thrddemo*. Now change this to *QForms*. CLX uses the same properties, methods, and class names, however, they may be stored in different units.

Open the *ThSort* tab and delete the non-portable *Windows* and *Messages* units. *QControls*, *QStdCtrls* and *QExtCtrls* have been added *Controls*, *StdCtrls* and *ExtCtrls*, and the versions without a *Q* are no longer needed. Replace *Graphics* with *QGraphics*, *Forms* with *QForms* and *Dialogs* with*QDialogs*. Because the form definition file is missing, these changes could not be performed automatically in *SortThds*, thus you will need to manually add a *Q* to *Graphics* and *ExtCtrls*.

This should leave you with a fully functional CLX version of the former VCL application. If you want to have the same kind of fun with Kylix for C++Builder, you can try out the example in the *Examples/Apps/Threads* directory. You might like to run the Project Conversion Utility for the project file first (see the "Project Conversion Utility" insert).

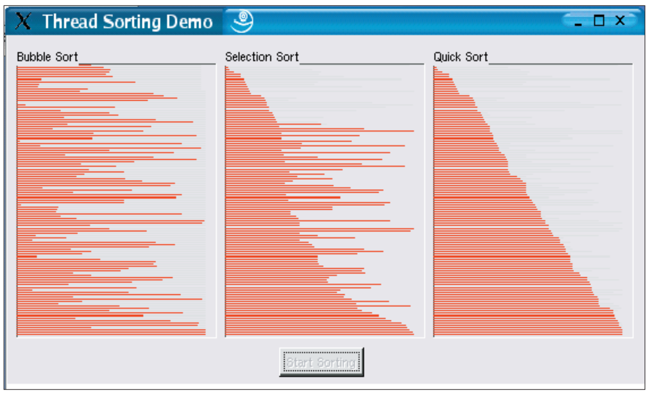


**Figure 2: A former VCL thread application as a CLX application on Linux. It classifies different algorithms by reference to the degree of sorting in the field. All threads and/or algorithms were launched simultaneously with the same line length**

## Additional Steps

*{$R *.dfm}* is now called *#pragma resource "*.dfm"* of course, and needs to be renamed to **.xfm*. Additionally, many *vcl.h* include files will need to be remapped to *clx.h*. *Synchronize(DoVisualSwap);* in *sortthd.cpp* becomes *Synchronize(&DoVisualSwap);*, as CLX and VCL use different call syntax. It is a good idea to replace *random(170);* in *thsort.cpp* with *random()%170;*, as the Linux random function manually

restricts random functions to a valid range. An appropriate *#ifdef* ensures the compatibility of the Windows version.

The problem with the C++ port is that the Kylix developers seemingly had good reason to use dynamic arrays rather than static ones in *Kylix3Pfad/examples/c/threads/*. Although the version with static arrays will run, you get a *EAccessViolation* message if you click (or if you click twice, at least) on *Start Sorting* shortly after sorting has completed. The back port to Windows does not exhibit this behavior.

## Porting a Delphi CLX Project to Kylix

Copy a Delphi CLX project, for example the CLX Explorer *Delphi7-Path\Demos\Clx\ClxExplorer*, to your Linux environment, launch Kylix for Delphi (*startdephi* ), and open the project. Clicking on *Start* should compile and launch the application on Linux (Figure 3). All done!

## A C++Builder/VCL Project in Kylix

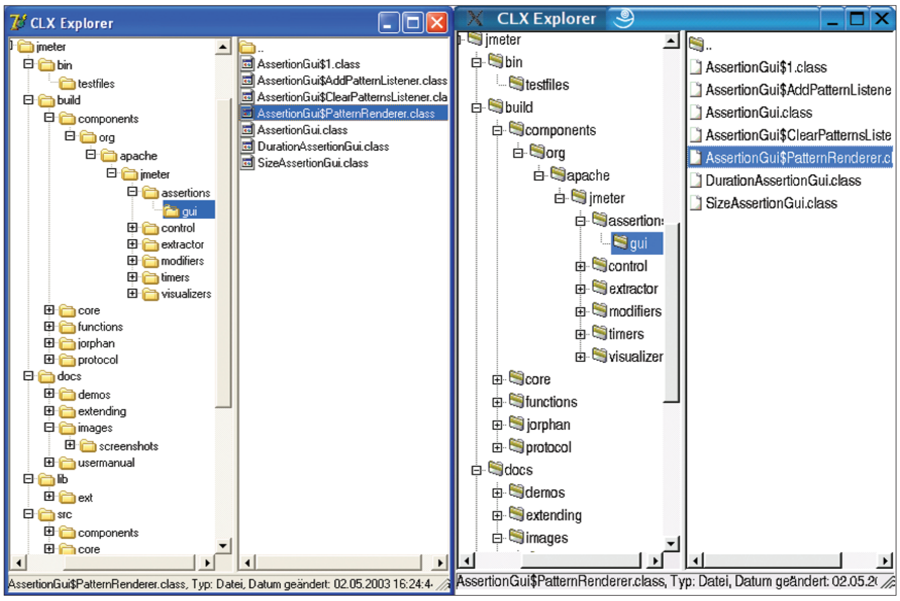
To make things slightly more complex, the following scenarios assumes that your current project is a C++Builder VCL project, rather than a Delphi CLX project. The following approach is required in this case: launch C++Builder and add button and a Timage control to a VCL application (*File* | *New* | *Application* ).

Change the name property of the form to *ImageForm*, to avoid collisions with *Form1* later, and set the *Stretch* property for *TImage* to *true*, to scale images to fit the image size.

The button will be used to modify the visibility of the image. To do so, double click on the button and add the required code to the event handler: *Image1->Visible = !Image1->Visible;*. Save this in the Linux share and close the project. Now rename *Unit1.dfm* to *Unit1.xfm* in the Linux share. Open *Unit1.cpp* in an editor and change the *#pragma resource *.dfm* line in *Unit1 .cpp* to *#pragma resource *.xfm*. Additionally change *#include <vcl.h>* to *#include <clx.h>* and save these changes.

Then launch the Project Conversion Utility in Kylix for C++ (see insert). Opening the exported file in Kylix will cause Kylix to add the required CLX units to *Unit1.h* the next time you save the file. You can now delete *#include <Controls.hpp>*, *#include <StdCtrls.hpp>* and *#include <ExtCtrls.hpp>* from *Unit1.h*, as the include files automatically added (*QControls*, *QStdCtrls* et cetera) have been replaced by the CLX versions of the VCL headers. Finally, change *#include <Forms.hpp>* to *#include <QForms.hpp>*.

If you launch a project that has been modified in this way on Windows, it will terminate immediately, and Kylix will issue a message saying that it cannot find the *vcl.h* header file (in *Project1.cpp* this time). You can then modify this file



AssertionGui$PatternRenderer.class, Typ: Datei, Datum geändert: 02.05.2003 16:24:4·

AssertionGui$PatternRenderer.class, Typ: Datei, Datum geändert: 02.05.2(

**Figure 3: Windows on the left, Linux on the right, simply by loading and compiling with Kylix**

### Other multi-platform tools

Kylix is not the only code development tool with multi-platform capabilities. Let's look at just two examples.

Omnis Studio 3 by Raining Data is an integrated, object oriented RAD environment for Mac, Windows, Solaris, and Linux. Programs are portable – assuming you avoid platform specific externals or system calls. Omnis Studio is mainly suited to developing frontends with matching backends, which in turn access MySQL, Oracle, DB2, and so on, natively or using ODBC. The Standard Edition costs somewhere in the region of 290 Euro. *http://www.omnis.net/products/studio*

IDE Revolution by Runtime Revolution has a similar approach. Text processing, multimedia, graphics and Internet applications, CGI and shell scripts, and database frontends developed using IDE Revolution will run on Linux/Unix, Windows, Mac OS and Mac OS X. The protocol language can handle associative arrays, regular expressions, HTTP, FTP, Sockets, ODBC and native MySQL, and provides for access to modules written in other languages.

Version 1.1.1 is current, the more functional version 2.0 is beta at present, although this phase should have been completed by end of year 2002. The Small Business Edition costs about $ 300 US, a Starter Kit is provided free of charge. *http://www.runrev.com*

### Listing 1: *onSelectItem* property

```
// Check if a non-directory entry
// has been selected
  if ((FileListView1->SelCount>0)&&((FileListView1->U
  Selections[0]->SR.Attr&faDirectory)==0))
// If so, load file
  ImageForm->Image1->Picture->LoadFromFile(U
  AnsiString(DirectoryTreeView1->Directory)+PathDelim+U
  AnsiString(FileListView1->Selections[0]->SR.Name));
```

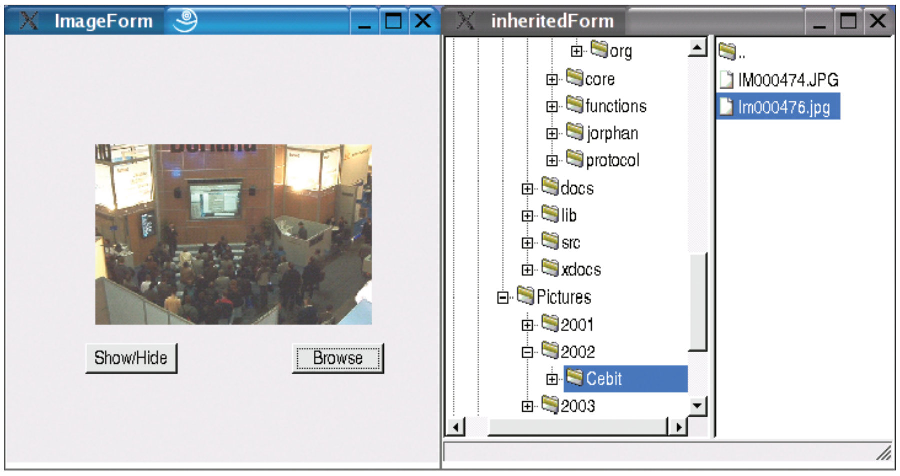


**Figure 4: The Kylix for C++ project shows an image viewer and the file browser – there is nothing left to remind the user of the program's Delphi roots**

(*Project* | *Display Source code* ) making it CLX conforming (*#include <clx.h>* instead of *#include <vcl.h>* ) to produce a fully functional CLX project.

To emphasize Kylix for Delphi's C++ capabilities, you might like to add the Delphi or Kylix for Delphi CLX Explorer to your Kylix for C++ project. After all, C++Builder has been able to speak Delphi for years.

## Polygamy

First select the *MainFrm.pas* file for the CLX Explorer in *Project* | *Add to Project*. Then select *Project* | *Update project name* a header file for the Delphi unit. The header is generated when you update the project; thus, it makes no sense to include the header file without compiling it at least once with the Pascal file.

You can then click on *File* | *New* | *Others* | *project name* | *Form1* to inherit the new header file. You might like to specify a name, such as *inheritedForm*, for the inherited form. Then add another button (with a *Browse* caption, for example) to your *ImageForm* and apply the inherited *Unit2.h* (by clicking on *File* | *Include Unit*or [Alt] + [F11]).

Specify *inheritedForm->Show();* as the event handler for *Button2*, paying attention to avoid using the original Pascal form! Of course it will work, and it looks similar in most parts, but any modifications in the inherited class will be ignored by definition.

Your Delphi CLX application is now ready for use in the Kylix for C++ application. Simply include the unit for your *ImageForm* (typically *Unit1.h*) in the file with *inheritedForm* (for example, *Unit2 .cpp*) and set the *onSelectItem* property of *FileListView1* in *inheritedForm* to the construction shown in Listing 1.

### Project Conversion Utility

This utility is available from the *eToys* section of [3]. It installs below *Tools* | *Project Conversion Utility* in the Kylix C++ IDE or C++Builder. The project conversion utility only converts the C++ Builder and/or Kylix for C++ project files without modifying sources, that is, it does not convert from VCL to CLX. The Windows version can convert Kylix for C++ project files to C++Builder 6 project files, and Linux version C++Builder 6 project files to Kylix für C++ project files. This is required at present on account of varying switches in these compilers.

## Conclusion: Simple through Complex

CLX portability of this first version with C++ shows some teething troubles, starting with the capitalization of Borland's own unit names, which differs between Windows and Linux (for example, *SoapHTTPClient*), through the dumping of the portable SQL Client dataset in Delphi 7 in favor of the simple dataset, and culminating in the need to convert C++ project files.

But Kylix 3 not only means that existing Delphi and C++ projects are portable to Linux, but that for the first time this really does mean the complete Delphi and C++ developer environment.

Both environments provide Rapid Application Development (RAD) facilities, allowing the developer to simply point and click to generate code for the user interface.

This solution is available for Linux, and is thus one of the first, if not the first, C++ RAD for Linux.

Practical porting still leaves a lot of scope – in the best of all cases, a CLX can be converted with minimal effort, within a few minutes or hours. The other extreme is a time-consuming Herculean tasks and assumes a high level of skill just to convert a single VCL application that makes lavish use of the BDE database connector, Windows messages, and direct Win32 API calls.

Programmers wanting to avoid this effort, are advised to concentrate on developing CLX applications – and to choose the right platform for the task, Linux. But even a Windows CLX development can be ported without any compromise, provided the developer avoids Windows specific features.

What ever approach you adopt, if you want to be able to recognize the kind of issues that can occur in large-scale projects, regular porting (at least once a week) of a CLX project to the target platform, is recommended. ■

### INFO

[1] Migration from Delphi 5 VCL to Kylix CLX: *http://bdn.borland.com/article/images/27534/migrating_delphi5.pdf*

[2] A case study on the Cross Platform Project: *http://www.borland.com/products/case_studies/kylix_colliers.html*

[3] Kylix Downloads (Trial, Open Edition, Patches and Project Conversion Utility): *http://www.borland.com/products/downloads/download_kylix.html*

[4] BDE to dbExpress conversion: *http://www.borland.com/products/white_papers/pdf/migrating_borland_database_engine_applications_to_dbexpress.pdf*

THE AUTHOR

*Jochen Stärk works for Borland GmbH in Germany, in technical systems support where his major focus is C++ Builder and Kylix.*