Jedi Code Library − Cross Platform Strategy

Table of Contents

Jedi Code Library − Cross Platform Strategy

This paper presents the JCL teams strategy for cross platform compliance of the Jedi Code Library. It is based up on the discussions within the JCL newsgroup and JCL developer mailing lists. This document is currently **work in progress** and subject to changes with or without notice.

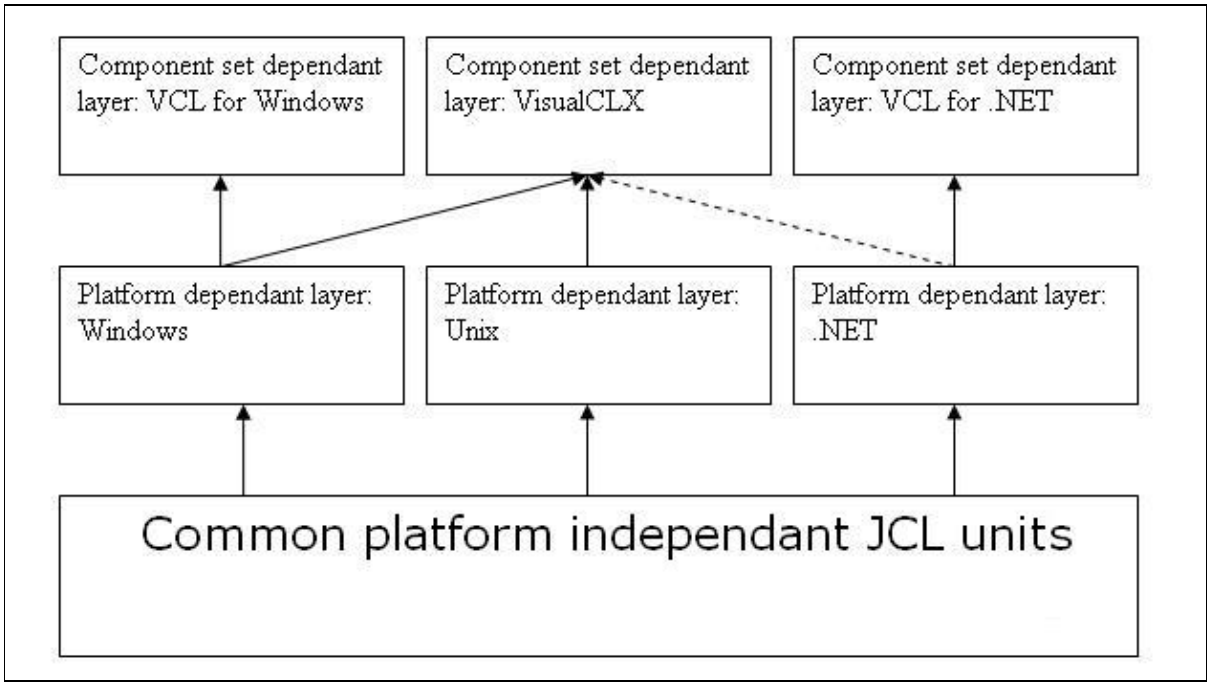
Contents

Background

The main objective is to make the Jedi Code Library VisualCLX (Kylix for Delphi/Delphi) and Delphi.NET compatible. For a detailed explanation of the currently used terminology, see the following article Overview of the VCL for .NET.

We have to cope with nearly all aspects of cross platform progamming, like different APIs, different Operating System concepts etc. Since we want to be as crossplatform compatible as possible interface compatability is the most important issue for us. Jedi Code Library users should have to opportunity the use the JCL on whatever platform they like. Figure 1 shows the three basic layers we have:

- Platform independant layer: Units which are not (or only very minor) platform specific and do not depend on a specific component set. This doesn't mean that units in this layer have to consist of no platform dependant code, but they have to be nearly 100% interface compatible and all functionality must have been ported to all supported platforms.
- Platform dependant layer: Units which depend on a specific platform (e.g. JclCLI)
- Component dependant layer: Units which depend on a specific component set


Component set dependant layer: VCL for Windows
Component set dependant layer: VisualCLX
Component set dependant layer: VCL for .NET
Platform dependant layer: Windows
Platform dependant layer: Unix
Platform dependant layer: .NET
Common platform independant JCL units
Fig 1: The Jedi Code Library crossplatform layer structure

Common platform independant layer

This layer consists of all files which are not platform dependant or need only very minor adjustations. Furthermore all units in this layer do not depend on a specific component set. Examples for common platform independant units are JclBase, JclDateTime, JclFileUtils and JclMath. The units have been ported to all platforms and are the crossplatform "core" of the Jedi Code Library. As a general rule a unit in this layer should have no platform specific ifdefs in its interface section.

Component dependant layer

When it comes to sharing code between VCL and VisualCLX−applications, some facts need to be stated:

- A unit is called *VCL−dependent*, when it uses some VCL−unit(s), e.g. Graphics.
- A unit is called *VisualCLX−dependent*, when it uses some VisualCLX−unit(s), e.g. QGraphics.

When a unit contains neither VCL− nor VisualCLX−specific code, there is no problem: It can be used by either type of application.

While it is basically possible to create VCL−dependent and VisualCLX−dependent variants of the same unit by means of conditional compilation − and use them in VCL− and VisualCLX−applications respectively −, this method fails at design time: One and the same unit cannot be installed twice in the IDE, not even as part of different packages. We would have to rename one of the variants, effectively creating a new unit. Therefor we will use a preprocessor to resolve the conditional compilation symbols related to VCL/VisualCLX−specific code and create VCL/VisualCLX units from a common codebase.

Component dependant units should be largely "interface compatible" − interface adjustments for specific component sets are unavoidable − nonetheless similar interfaces are desirable.

Platform dependant layer

Furthermore we do not have to differentiate between VCL and VisualCLX units only (the so called component set dependent layer), but also between UNIX, Windows and .NET dependent units. The platform dependent units doesn't need to be interface compatible (if there is an equivalent in one of the other suported platforms at all!). An example for a platform dependant unit is JclCLI. Nonetheless if there are equivalents in all other supported platforms as well it might be considerable to write a more general class and include that unit into the common platform indepedant layer.

Preprocessor

The preprocessor jpp is a modified version of Barry Kelly's ppp tool. In contrast to ppp, which resolves all conditional compilation directives without exception, with jpp symbols not only can be defined but also undefined. Those symbols which are neither defined nor explicitely undefined are considered as of unknown status and it and its related source code remains untouched.

The usage of jpp is not too hard. It is called via

```
jpp [options] <input files>...
```

Possible options are

```
  -i       Process includes
  -c       Process conditional directives
  -C       Strip comments
  -pxxx    Add xxx to include path
  -dxxx    Define xxx as a preprocessor conditional symbol
  -uxxx    Assume preprocessor conditional symbol xxx as not defined
  -x[n:]yyy  Strip first n characters from file name; precede filename by prefix yyy
```

The example command line below generates a file JclQGraphics.pas in subdirectory CLX from file Graphics.cb located in the current directory. Symbols "VisualCLX" and "COMPILER6_UP" are specified as defined, "Bitmap32" and "VCL" as undefined.

```
  jpp -c -dVisualCLX -dCOMPILER6_UP -uBitmap32 -uVCL -xCLX\JclQ Graphics.cb
```

Minimizing VCL dependencies

To reduce VCL dependencies in JCL, the following changes have been made:

JclFileUtils
: PathCompactPath is an overloaded function. The variant which takes a TCanvas as argument (and thus creates a dependency on VCL unit Graphics) has been removed.

JclShell
: ShellLinkGetIcon has been removed. It could get part of some genuine VCL−dependent unit (e.g. JclGraphUtils), but for now it is left out.

JclPEImage

```
 Replace "uses Consts," by

    uses
      {$IFDEF COMPILER6_UP}
      RtlConsts,         // VisualCLX-package compatible (part of rtlxx.bpl)
```

```
                    {$ELSE}
                    Consts,            // not VisualCLX-package compatible (part of vclxx.bpl)
                    {$ENDIF COMPILER6_UP}
```

Note that the first two changes have enormous impact, since many JCL units use JclFileUtils and JclSysInfo (which both use JclShell). This leaves JclGraphics and JclGraphUtils as sole units with genuine VCL/VisualCLX−dependencies. JclPrint is the only remaining pure VCL−dependent units.

Generating Jcl[Q]Graphics.pas and Jcl[Q]GraphUtils.pas

First compile Preprocessor\jpp.exe from Preprocessor\jpp.dpr. Then change to the "Source" directory and type "make" at the command line. This will create the units
VCL\JclGraphics.pas
VCL\JclGraphUtils.pas
CLX\JclQGraphics.pas
CLX\JclQGraphUtils.pas

from their prototypes _Graphics.pas and _GraphUtils.pas.

Status − Platforms

This table gives a short overview of which units are already working under three different Delphi language compilers/platforms. There are three status levels possible:

- Yes − The unit has been ported to that platform
- No − The unit has not been ported to that platform
- dependant − The unit is platform dependant and will not be ported.

Name	**Delphi (Windows)**	**Kylix for Delphi**	**Delphi.NET**
Jcl8087	Yes	No	No
JclAppInst	Yes	No	No
JclCil	Yes	No	No
JclClr	Yes	No	No
JclCom	Yes	No	No
JclComplex	Yes	No	No
JclConsole	Yes	No	No
JclCounter	Yes	No	No
JclDateTime	Yes	No	No
JclDebug	Yes	No	No
JclDotNet	Yes	No	No
JclEdi	Yes	No	No
JclExprEval	Yes	No	No
JclHookExcept	Yes	No	No
JclIniFiles	Yes	No	No
JclLanMan	Yes	No	No
JclLocales	Yes	No	No
JclLogic	Yes	No	No
JclMapi	Yes	No	No

JclMath	Yes	No	No
JclMetaData	Yes	No	No
JclMidi	Yes	No	No
JclMime	Yes	No	No
JclMisecel	Yes	No	No
JclMultimedia	Yes	No	No
JclNTFS	Yes	No	No
JclPEImage	Yes	No	No
JclPrint	Yes	No	No
JclStrHashMap	Yes	No	No
JclStatistics	Yes	No	No
JclShell	Yes	No	No
JclSecurity	Yes	No	No
JclSchedule	Yes	No	No
JclRTTI	Yes	No	No
JclResources	Yes	No	No
JclRegistry	Yes	No	No
JclStrings	Yes	No	No
Jclsvcctrl	Yes	No	No
Jclsynch	Yes	No	No
JclTask	Yes	No	No
JclSysUtils	Yes	No	No
JclSysInfo	Yes	No	No
JclTD32	Yes	No	No
JclUnicode	Yes	No	No
JclUnitConv	Yes	No	No
JclWin32	Yes	No	No
JclWinMidi	Yes	No	No

Packages