Borland

Migrating your Delphi™ 5 Projects to Kylix™

Author: Bob Swart (a.k.a. Dr. Bob - www.drbob42.com)

Contents

Introduction

This paper provides an overview and detailed description of migrating applications from Borland® Delphi™5 for Windows® to Kylix™ for Linux.® We first explain why it is an asset to be able to develop applications for both Windows and Linux (and to easily migrate existing applications from Windows to Linux). Then we outline the shared architecture and technologies between the tools and projects, including the differences between VCL (visual component library) and CLX™ (component library for cross-platform development). This paper also discusses topics developers should be mindful of, as well as best practices developers should keep in order to migrate code from Windows to Linux and to maintain single-source, cross-platform applications between these two platforms (something which has become much easier with Delphi 6).

Kylix™

white paper

Migrating Delphi™ Projects to Kylix™

This white paper is not about cross-platform code shared between Delphi 5 and Kylix, but rather about migrating existing Delphi 5 (VCL) projects, components, and source code to Kylix (CLX) on Linux. This is an important first step in writing cross-platform code. (While the modified project source code for the Kylix edition of your project may no longer work with Delphi 5, it is likely to work flawlessly with Delphi 6).

We will see how to write simple, single-source applications (using as few IFDEFs as possible) that can be compiled using both Delphi for Windows (VCL) and Kylix for Linux (CLX). We will also provide an overview of features that are in Delphi 5 but are not supported in Kylix, and we'll suggest alternatives.

Why migrate?

Let's consider the benefits of migrating Delphi 5 projects from Windows to Linux (using Kylix). Apart from being able to share components and sections of source code, the biggest advantage is that a whole new market can be penetrated with existing code:you need not re-invent the wheel (though you may have to fill your tires with fresh air).

I, for example, have been able to migrate a number of existing Web server applications (using Delphi 5 WebBroker™ technology) from the Microsoft Internet Information Server (IIS) running on Windows to Apache™ Shared Object (DSO) running on Linux—in less than one hour each. This means I am no longer limited to Web servers running Windows NT or Windows 2000. I can deploy on Linux Web servers (one area where Linux is particularly strong,). Of course, the same can be done for regular common gateway interface (CGI) Web server applications—in fact, it's even easier for those, as we will see.

Migrating projects

The first objective when migrating your Delphi 5 project to Kylix is to physically move your Delphi 5 project from a Windows machine to a Linux machine. This must be done because Linux uses case-sensitive filenames, and Windows does not:. Whereas Project1.dpr and PROJECT1.DPR both point to the same file in Windows, they are two different files in Linux. The names of your project and units (as mentioned on the first line of your project and unit source files) must be identical to the actual filename. And case-sensitivity becomes even more important when considering the list of units inside the uses clauses of each unit. If your unit is called "Myunit" and stored as Myunit.pas on disk, then you must also call it Myunit, not MyUnit, inside the uses clause because the unit named MyUnit in file MyUnit.pas will not be found. As a workaround, you can explicitly add to every unit in the uses clause of the main project source file the name of the file where the unit can be found. For example:

```
uses
  MyUnit in "Myunit.pas",
  unit1 in "Unit1.pas";
```

Note that these "in" clauses will appear only in your main project source file, not in unit source files.

This might seem like a trivial task, but it could take a while before everything is renamed correctly, so at least all your project units can be found by the compiler. Speaking of which, a Delphi 5 project consists of a .dpr (project source code) file, a .res (icon resource) file, an .opt (project options) file, a .cfg (command-line options) file, and a set of .pas files, usually with corresponding .dfm files. A Kylix project consists of a .dpr (project source code) file, a .res file, a .conf (configuration) file, a .kof (Kylix Option File), and a set of .pas files with corresponding .xfm files. The Delphi 5 .cfg file corresponds to a Kylix .conf file, and a Delphi .opt file corresponds to a Kylix .kof file. Inside, these files differ a bit (some compiler options are meaningless in Kylix or Windows, and directories are specified with a slash on Linux instead of a drive letter, and a backslash on Windows), hence the new file extension. You can best convert files by hand or start a new empty project and migrate your units and other source code. The latter is my preference (especially for WebBroker projects).

The .xfm files are simply .dfm files. The easiest and perhaps most reliable way to turn .dfm files into .xfm files is to make sure

the .dfm files are stored as text. Either have Delphi store them as text or convert them to text using the convert.exe utility that comes with Delphi. Then, simply rename them as .xfm files. Thus, if your .dfm files are already stored as text instead of binary files, you can simply rename them from .dfm to .xfm and you're prepared for the next phase.

Single source

If you want to maintain a single source of your project that can compile with both Delphi and Kylix, you must at least make sure that Delphi uses the .dfm file and Kylix the .xfm file. This is handled by the {$R *.DFM} statement in Delphi and the {$R *.xfm} statement in Kylix. A single-source piece of code can be written using IFDEF, where Windows is identified by the conditional symbol MSWINDOWS, and Linux by LINUX:

```
{$IFDEF MSWINDOWS}
  {$R *.DFM}
{$ENDIF}
{$IFDEF LINUX}
  {$R *.xfm}
{$ENDIF}
```

You should never use the {$ELSE} directive and assume that if it is not Windows it should be Linux, or vice versa. It's not impossible that a future version of Delphi or Kylix might run on an entirely new operating system, and then the {$ELSE} could lead to incorrect results. Also, make sure you use MSWINDOWS instead of WIN32, or your code might break when a 64-bit version of Windows ships.
Another conditional symbol defined for Kylix that might be useful is VER140, which indicates that Kylix is version 14 of the Pascal compiler (in contrast, Turbo Pascal™ was version 1, Delphi 1.0x is version 8 with VER80 defined, and Delphi 5 is version 13 with VER130 defined).

Non-portable projects

Not every Delphi 5 project can be migrated to Kylix. Specifically, the ActiveX® library cannot be ported to Linux (no COM objects, ActiveX controls, ADO, ActiveForms, or Active Server Pages). This is unsurprising, but it's best to know about this before you start to invest time and/or money in a single-source cross-platform ActiveForm solution.
With the aforementioned migration actions in mind, normal Windows GUI applications are good candidates for porting to Linux.

WebBroker™ targets

Delphi 5 WebBroker™ architecture supports CGI, WinCGI, and ISAPI/NSAPI Web server applications. NSAPI support is handled by the WebBridge that is based on the same code as the ISAPI DLL. Linux supports neither ISAPI (and hence no NSAPI via the WebBridge) nor WinCGI. But that doesn't mean all your non-CGI WebBroker applications are impossible to migrate to Linux. One good thing about WebBroker is the fact that the core functionality of your Web server application is contained within the Web module. The Web module is not concerned with the target of the application. In fact, even with a standard CGI WebBroker application, you may want to port only your Web modules to Linux and create a new WebBroker application using Kylix (which has options for CGI or Apache.shared object). Then you can add the Web modules to your new native WebBroker application on Linux. In a matter of minutes, your WebBroker application skeleton will work.

Migrating database data to Linux

Delphi 5 supports a number of different databases and table formats, such as BDE (dBASE®, Paradox®) InterBase® (native as well as via the BDE), ODBC/ADO (to Access and FoxPro,® for example), and SQL Server to Oracle,® DB2,® Informix,® and SQL Server. Kylix does not support BDE but has replaced it with dbExpress™ for open access to native Linux database engines. The Kylix desktop developer supports MyBase® (local XML), MySQL,® and InterBase, while the Kylix server developer adds support for Oracle and DB2.
As a consequence, not all database formats available in Delphi 5 will be available for Kylix. Specifically, the Borland® Database

Engine (dBase, Paradox formats) will not be ported to Linux. This means that your dBase and Paradox tables cannot be used by Kylix.

Fortunately, you can use Delphi 5 to migrate your dBase and Paradox tables to InterBase or the local ClientDataSet format (CDS or XML). Data in XML format can be used by the MyBase personal XML database for Linux.

Using Delphi 5 Enterprise, you can connect a table to a DataSetProvider in order to feed a ClientDataSet component that can be used to store the data in .cds or .XML format. If you do not have Delphi 5 Enterprise (and no ClientDataSet component), consider using the following source code (based on an article I wrote for *The Delphi Magazine*, published by iTec: www.TheDelphiMagazine.com) to assist in converting a dataset to XML:

```
unit TableXML;
// Routine DataSetXML converts a DataSet to XML
format
// Author: Bob Swart (a.k.a. Dr. Bob -
www.drbob42.com)
// Available as freeware, but use at your own
risk!
//
interface
uses
  DB;

function DataSetXML(DataSet: TDataSet; const
FileName: String): Integer;

implementation
uses
  SysUtils, TypInfo;

function DataSetXML(DataSet: TDataSet; const
FileName: String): Integer;
var
  F: System.Text;
  i: Integer;

  function Print(Str: String): String;
  { Convert a fieldname to a printable name }
  var
    i: Integer;
  begin
    for i:=Length(Str) downto 1 do
      if not (UpCase(Str[i]) in
['A'..'Z','1'..'9']) then
        Str[i] := '_';
    Result := Str
  end {Print};

  function EnCode(Str: String): String;
  { Convert memo contents to single line XML }
  var
    i: Integer;
  begin
    for i:=Length(Str) downto 1 do
    begin
      if (Ord(Str[i]) < 32) or (Str[i] = '"')
then
      begin

Insert('&#'+IntToStr(Ord(Str[i]))+';',Str,i+1);
        Delete(Str,i,1)
      end
    end;
    Result := Str
  end {EnCode};

begin
  Result := -1;
  ShortDateFormat := 'YYYYMMDD';
  System.Assign(F,FileName);
  try
    System.Rewrite(F);
    writeln(F,'<?xml version="1.0"
standalone="yes"?>');
    writeln(F,'<DATAPACKET Version="2.0">');
    with DataSet do
    begin
      writeln(F,'<METADATA>');
      writeln(F,'<FIELDS>');
      if not Active then
        FieldDefs.Update { get info without
opening the database };
      for i:=0 to Red(FieldDefs.Count) do
      begin
        write(F,'<FIELD ');
        if Print(FieldDefs[i].Name) <>
FieldDefs[i].Name then { fieldname }

write(F,'fieldname="',FieldDefs[i].Name,'" ');

write(F,'attrname="',Print(FieldDefs[i].Name),'"
fieldtype="');
        case FieldDefs[i].DataType of
          ftString,
       ftFixedChar,
      ftWideString: write(F,'string');
         ftBoolean: write('boolean');
        ftSmallint: write(F,'i2');
         ftInteger,
            ftWord: write(F,'i4');
         ftAutoInc: write(F,'i4" readonly="true"
SUBTYPE="Autoinc');
           ftFloat: write(F,'r8');
        ftCurrency: write(F,'r8"
SUBTYPE="Money');
             ftBCD: write(F,'fixed');
            ftDate: write(F,'date');
            ftTime: write(F,'time');
        ftDateTime: write(F,'datetime');
           ftBytes: write(F,'bin.hex');
        ftVarBytes,
            ftBlob: write(F,'bin.hex"
SUBTYPE="Binary');
            ftMemo: write(F,'bin.hex"
SUBTYPE="Text');
         ftGraphic,
     ftTypedBinary: write(F,'bin.hex"
SUBTYPE="Graphics');
         ftFmtMemo: write(F,'bin.hex"
SUBTYPE="Formatted');
      ftParadoxOle,
```

```
        ftDBaseOle: write(F,'bin.hex"
SUBTYPE="Ole')
        end;
        if FieldDefs[i].Required then write(F,'"
required="true');
        if FieldDefs[i].Size > 0 then write(F,'"
WIDTH="',FieldDefs[i].Size);
        writeln(F,'"/>')
      end;
      writeln(F,'</FIELDS>');
      writeln(F,'</METADATA>');
      if not Active then Open;
      writeln(F,'<ROWDATA>');
      Result := 0;
      while not Eof do
      begin
        Result := Result + 1;
        write(F,'<ROW ');
        for i:=0 to Pred(Fields.Count) do
          if (Fields[i].AsString <> '') and
            ((Fields[i].DisplayText =
Fields[i].AsString) or
             (Fields[i].DisplayText = '(MEMO)'))
then

write(F,Print(Fields[i].FieldName),'="',
                    EnCode(Fields[i].AsString),'"
');
        writeln(F,'/>');
        Next
      end;
      writeln(F,'</ROWDATA>')
    end;
    writeln(F,'</DATAPACKET>')
  finally
    System.Close(F)
  end
end;
```

After you've converted your tables, you can move them from Windows to Kylix and load the XML files using MyBase. When needed, you can then use MyBase to migrate the data even further (to Oracle, for example). Of course, you can always move your database to an InterBase database, which runs on Linux.

Finally, there is no support in Kylix for ODBC/ADO (to Access and FoxPro, for example) or SQL Server and Informix. For these database formats, it would be best to consider a migration to either InterBase or Oracle.

Third-party ODBC support

The dbExpress™ Gateway for ODBC from Easysoft (www.easysoft.com/products/kylix/main.phtml) enables Kylix applications to access any ODBC data source. It works with Linux databases that have a driver (such as InterBase, Postgres, Oracle, etc.), plus remote databases via the Easysoft ODBC bridge software. This means that Kylix applications can work with Access and SQL Server, as well as with any database with a Windows ODBC driver. The interface is also compatible with unixODBC, the open-source ODBC driver manager for Linux.

dbExpress™

Kylix uses dbExpress for cross-platform data-access technology. There is a difference between the way dbExpress accesses your database and tables and the way you might be used to doing it with the BDE, ADO, and other Delphi database mechanisms. For one thing, dbExpress offers only a unidirectional cursor to your tables. This means that you can browse forward but not backward. This is designed for efficiency, but it could be quite different from more familiar methods.

You can still browse back and forth through your data, once you connect a dbExpress data source to a ClientDataSet. This source then will be used as a local briefcase. dbExpress drivers are available for InterBase, Oracle, DB2, and MySQL, but it is also possible to write your own custom dbExpress driver (contact Borland for details).

Table 1 shows how the BDE, InterBase Express,™ and ADO Express components map to the dbExpress components. Since most of your Delphi database applications perform editing and navigation, you should be prepared to "insert" both a DataSetProvider and a ClientDataSet component for each newly ported dbExpress component. This allows your application to be connected to data-aware components such as a TDBGrid or TDBEdit. For now, there is no automatic procedure or tool to assist in this task.

The final step in migrating your database application is to make sure that modifications in your tables (using the ClientDataSet component) are indeed sent back to the original dbExpress dataset. Do this by explicitly calling the ClientDataSet. ApplyUpdates method; for example, in the OnAfterPost event of your ClientDataSet:

```
procedure
TForm1.ClientDataSet1AfterPost(DataSet:
TDataSet);
begin
  if DataSet IS TDataSet then
    TClientDataSet(DataSet).ApplyUpdates(-1)
end;
```

This step may be a bit slow, since you now call ApplyUpdates for every Post in your ClientDataSet. The alternative is to have an explicit "ApplyUpdates" button on your Form (or to fire ApplyUpdates when the user closes the Form or when a specific timer event fires). There are numerous possibilities. The safest, though slowest approach is the one described above.

For more information on using ClientDataSets, please see existing MIDAS documentation on the Borland Community Web site (http://communityborland.com). or on my own site at www.drbob42.com

Moving from VCL to CLX

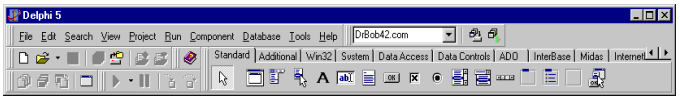
The Delphi visual component library (VCL) contains much more than visual components. In my view, this puzzle is solved by the component library for cross-platform development (CLX), which is divided into four parts (BaseCLX,™ VisualCLX,™ DataCLX,™ and NetCLX™), of which only one is visual—a much better classification!

The Kylix CLX maintains the architecture and hierarchy of the Delphi VCL, which makes the look-and-feel almost identical for Delphi developers—and your applications. Behind the scenes, CLX is making calls to the Qt library, which is available for both Windows and Linux (and hence guarantees the cross-platform nature of CLX when both Kylix and Delphi are available).

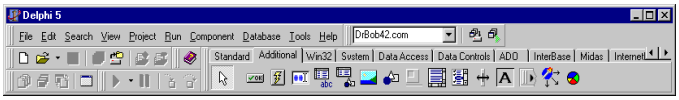
But we should first look at existing Delphi 5 VCL projects, especially the components available in the Delphi 5 component palette, to see which ones will be available in the new CLX hierarchy (and, more importantly, which ones won't). If you know beforehand that a certain component will not be available or supported by Kylix, then it's often much cheaper to create a workaround while still in Delphi 5, rather than move to Linux and potentially have to rewrite everything.

Available components

We will now examine an overview of components in Delphi 5 (Enterprise) that are available in Kylix (Server Developer). We will also list components of Delphi 5 that are not available in Kylix, suggesting alternative, third-party components or ways to avoid the use of these components altogether.



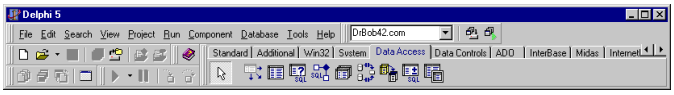
The standard page of the Delphi 5 component palette contains only components that can be found in Kylix, so everything will port over. For individual components, there may be some properties or methods that differ, but these changes often are easy to fix.



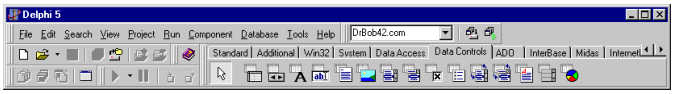
The additional page of the Delphi 5 component palette contains three components that cannot be found in Kylix, namely TStaticText, TapplicationEvents, and TChart. For TStaticText, you can always use TLabel (the difference between a StaticText and a Label is that a StaticText is derived from TWinControl and has its own Windows handle).

The Win32 page of the Delphi 5 component palette is gone in Kylix. Instead, we have a common control page on the Kylix component palette, which contains most, but not all of the components of the Win32 tab. Components that are missing on the common controls tab in Kylix are TRichEdit, TUpDown (there is a TSpinEdit, which in Delphi appears on the Samples tab), THotKey, TAnimate, TDateTimePicker, TCalendar, TCoolBar, and TPageScroller.

The system page of the Delphi 5 component palette also is gone in Kylix, and so are most of the controls on those pages.. Two of them, TTimer and TPainBox, appear in the Additional page of the Kylix component palette. The other components, TMediaPlayer, TOleContainer, TDdeClientConv, TDdeClientItem, TDdeServerConv, and TDdeServerItem, do not exist in Kylix.



The data access page of the Delphi 5 component palette, or, in other words, the BDE page, is almost entirely gone from Kylix. Only one component from this tab remains: TDataSource. The data access page in Kylix also contains the TClientDataSet and TDataSetProvider components (three in total).
Please check the "Migrating database data to Linux" section for more information regarding the migration of your database data from BDE to MyBase or InterBase.



The data controls page of the Delphi 5 component palette contains three components not found on the Data Controls tab in Kylix: TDBRichEdit, TDBCtrlGrid, and TDBChart.

The ADO tab of the Delphi 5 component palette is entirely gone and not present at all in Kylix.

The InterBase tab of the Delphi 5 component palette also is gone in Kylix. The single dbExpress tab of Kylix replaces the functionality of the components in the Data Access (BDE), ADO, and InterBase tabs.

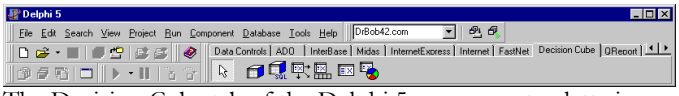
The MIDAS tab of the Delphi 5 component palette is gone in Kylix, though the TClientDataSet and TDataSetprovider components appear on the data access tab in Kylix. The other components, TDCOMConnection, TSocketConnection, TSimpleObjectBroker, TwebConnection, and TCorbaConnection are not present in Kylix. The first may never appear in Kylix (DCOM on Linux, perhaps?), but the others may become available in the forthcoming Enterprise edition of Kylix.

The InternetExpress™ tab of the Delphi 5 component palette is not present in Kylix, mainly because InternetExpress and MIDAS are not supported in the Kylix Server edition; these tabs will appear in the Enterprise edition of, however. The absence of MIDAS and InternetExpress means we cannot yet write distributed applications with Kylix.

The Internet tab of the Delphi 5 component palette contains a number of components that cannot be found on the Internet tab in Kylix: TClientSocket and TServerSocket (these two seem to be replaced with TTcpClient, TTcpServer, and TUdpSocket) and TWebBrower (which was merely a wrapper around the Internet Explorer ActiveX control). Apart from these three components, the TQueryTableProducer in Delphi has been renamed as TSQLQueryTableProducer in Kylix.

The FastNet tab of the Delphi 5 component palette has been replaced by three Indy (Internet direct) tabs, namely Indy Clients, Indy Servers, and Indy Misc. It is likely that most, if not

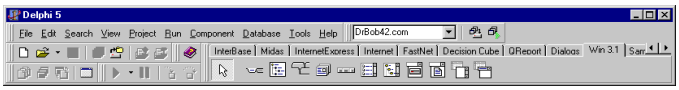
all, of the code in projects using FastNet will need to be rewritten using Indy components.



The Decision Cube tab of the Delphi 5 component palette is gone in Kylix as well.

The QReport tab of the Delphi 5 component palette is no longer present in Kylix. For reporting, I suggest looking at third-party support such as Rave Reports from Nevrona.

The dialogs tab of the Delphi 5 component palette contains four components that are missing in Kylix, namely TOpenPictureDialog, TSavePictureDialog, TprintDialog, and TPrinterSetupDialog.



The Win 3.1 tab of the Delphi 5 component palette contains no components that can be found in Kylix. This means no TTabSet, no TOutline, no TTabbedNotebook, no TNotebook, no THeader, no TFileListBox, no TDirectoryListbox, no TDriveComboBox, and no TFilterComboBox.

Finally, the samples page of the Delphi 5 component palette is not present in Kylix, although the TSpinEdit appears on the common controls tab of Kylix. Nor is there TGauge, TColorGrid, TSpinButton, TDirectoryOutline, TCalendar, nor TIBEventAlerter.

Suggested alternatives

The following table contains a list of missing components (sometimes entire tabs) and the recommended component in Kylix, or an alternative way to realize the required functionality.

*Table 1

Delphi 5 Component	Kylix Component	Comments
Additional		
TStaticText	TLabel	
TApplicationEvents		
TChart		TeeChart
Win32		
TRichEdit		Win32-specific component
TUpDown	TSpinEdit	
THotKey		Win32-specific component
TAnimate		Win32-specific component
TDateTimePicker		Win32-specific component
TCalendar		Win32-specific component
TCoolBar		Win32-specific component
TPageScroller		Win32-specific component
System		
TMediaPlayer		
TOleContainer		Win32-specific component
TDdeClientConv		Win32-specific component
TDdeClientItem		Win32-specific component
TDdeServerConv		Win32-specific component
TDdeServerItem		Win32-specific component
Data Access	dbExpress	
TTable	TSQLTable	Plus DataSetProvider and ClientDataSet
TQuery	TSQLQuery	plus DataSetProvider and ClientDataSet
TStoredProc	TSQLStored Proc	plus DataSetProvider and ClientDataSet

TDatabase	TSQL Connection	
TSession		BDE-specific component
TBatchMove		
TUpdateSQL	TSQLDataSet	plus DataSetProvider and ClientDataSet
TNestedTable		
Data Controls		
TDbRichEdit		Win32-specific component
TDbCtrlGrid		
TDbChart		
ADO	dbExpress	
TADO Connection	TSQL Connection	
TADO Command	TSQLDataSet	plus DataSetProvider and ClientDataSet
TADO DataSet	TSQLDataSet	plus DataSetProvider and ClientDataSet
TADO Table	TSQLTable	plus DataSetProvider and ClientDataSet
TADO Query	TSQLQuery	plus DataSetProvider and ClientDataSet
TADO StoredProc	TSQLStored Proc	plus DataSetProvider and ClientDataSet
TRDSConnection		
	TSQLMonitor	
	TSQLClient DataSet	
InterBase	dbExpress	
TIBTable	TSQLTable	plus DataSetProvider and ClientDataSet
TIBQuery	TSQLQuery	plus DataSetProvider and ClientDataSet
TIBStoredProc	TSQLStored Proc	plus DataSetProvider and ClientDataSet
TIBDatabase	TSQL Connection	
TIBTransaction		
TIBUpdateSQL	TSQLDataSet	plus DataSetProvider and ClientDataSet
TIBDataSet	TSQLDataSet	plus DataSetProvider and ClientDataSet
TIBSQL	TSQLDataSet	plus DataSetProvider and ClientDataSet
TIBDatabaseInfo		
TIBSQLMonitor	TSQLMonitor	
TIBEvents		
MIDAS		
TDCOMConnection		Win32-specific component
TSocketConnection		Expected in Kylix Enterprise
TsimpleObject Broker		Expected in Kylix Enterprise
TWebConnection		Expected in Kylix Enterprise
TCorbaConnection		Expected in Kylix Enterprise
InternetExpress		
TXMLBroker		Expected in Kylix Enterprise
TMidasPageProducer		Expected in Kylix Enterprise
Internet		
TClientSocket	TTcpClient, TUdpSocket	
TServerSocket	TTcpServer, TUdpSocket	
TWebBrowser		Win32-specific component
FastNet	Indy Servers	
	Indy Clients	
	Indy Misc	
Decision Cube		
QReport		Rave Reports from Nevrona are recommended
Dialogs		
TOpenPictureDialog		
TSavePictureDialog		
TPrintDialog		
TPrinterSetupDialog		
Win 3.1		

Tip: if you want to start building cross-platform applications with Delphi 5 and Kylix now, you should configure the Delphi 5 component palette in such a way that it does not show the components that are not present in Kylix. This is easy to do: just right-click with the mouse on the component palette, select "properties", and hide all components from the table.

Components and units

The discussion of CLX versus VCL is continued with the naming convention for the CLX units (such as QGraphics

versus Graphics), which heralds Delphi 6—the first development environment with design-time support integrated for two application frameworks (VCL and CLX).

The following table lists the units in Delphi 5 (from the Source\VCL directory) that are renamed in Kylix (also in the Source\VCL directory). Note that most changes involve a Q prefix.

Delphi 5 unit	Kylix unit	Comments
ActnList	QActnList	
ADOConst		Win32-specific—not in KylixWin32-specific—not in Kylix
ADODB		Win32-specific—not in KylixWin32-specific—not in Kylix
ADOInt		Win32-specific—not in KylixWin32-specific—not in Kylix
AppEvnts		Not in Kylix
AxCtrls		Win32-specific—not in KylixWin32-specific—not in Kylix
BdeConst		Not in Kylix
BdeMts		Not in Kylix
Buttons	QButtons	
CheckLst	QCheckLst	
Classes	Classes	
ClipBrd	QClipBrd	
ComCtrls	QComCtrls	
ComStrs		Not in Kylix
Consts	Consts	see also QConsts
Contnrs	Contnrs	
Controls	QControls	
CORBACon		Expected in Kylix Enterprise
CORBARdm		Expected in Kylix Enterprise
CORBAStd		Expected in Kylix Enterprise
CORBAVcl		Expected in Kylix Enterprise
CtlPanel		Not in Kylix
DataBkr		Not in Kylix
DB	DB	
DBActns	QDBActns	
DBCGrids		Not in Kylix
DBClient	DBClient	
DBCommon	DBCommon	
DBConsts	DBConsts	
DBCtrls	QDBCtrls	
DBExcept		Not in Kylix
DBGrids	QDBGrids	
DBInpReq		Not in Kylix
DBLookup		Not in Kylix
DBOleCtl		Win32-specific—not in KylixWin32-specific—not in Kylix
DBPWDlg	DBLogDlg	
DBTables		Not in Kylix
DdeMan		Win32-specific—not in KylixWin32-specific—not in Kylix
Dialogs	QDialogs	
DRTable		Not in Kylix
DSIntf	DSIntf	
ExtCtrls	QExtCtrls	
ExtDlgs		Not in Kylix
FileCtrl		Not in Kylix
Forms	QForms	
Graphics	QGraphics	
Grids	QGrids	
ImgList	QImgList	
IniFiles	IniFiles	
Mask	QMask	
Masks	Masks	See also MaskUtil
MConnect		Expected in Kylix Enterprise
Menus	QMenus	
Midas	Midas	
MidasCon		Not in Kylix
MidConst	MidConst	MIDAS is now called DataSnap™
MPlayer		Win32-specific—not in KylixWin32-specific—not in Kylix
Mtsobj		Win32-specific—not in KylixWin32-specific—not in Kylix
MtsRdm		Win32-specific—not in KylixWin32-specific—not in Kylix
Mtx		Win32-specific—not in KylixWin32-specific—not in Kylix
mxConsts		Decision Cube - not in Kylix
ObjBrKr		Expected in Kylix Enterprise
OleAuto		Win32-specific—not in Kylix
OleConst		Win32-specific—not in Kylix
OleCtnrs		Win32-specific—not in Kylix
OleCtrls		Win32-specific—not in Kylix
OLEDB		Win32-specific—not in Kylix
OleServer		Win32-specific—not in Kylix
Outline		Not in Kylix

Printers	QPrinters	
Provider	Provider	
Registry		Not in Kylix (use IniFiles)
ScktCnst		Not in Kylix
ScktComp		Not in Kylix
SConnect		Not in Kylix
SMIntf		Not in Kylix
StdActns	QStdActns	
StdCtrls	QStdCtrls	
StdVcl		Win32-specific—not in Kylix
SvcMgr		WinNT specific - not in Kylix
SyncObjs	SyncObjs	
Tabnotbk		Not in Kylix
Tabs		Not in Kylix
Toolwin		No docking supported in Kylix
TypInfo	TypInfo	
VCLCom		Win32-specific—not in Kylix
WebConst		Win32-specific—not in KylixWin32-specific—not in Kylix
Windows		Win32-specific—not in KylixWin32-specific—not in Kylix

It is to be expected that the CLX implementation in Delphi 6 use similar filenames as currently used by Kylix.

Migrating RunTime Library usage

This section focuses primarily on differences in the runtime library (RTL) and Object Pascal language features between Delphi 5 and Kylix. The Delphi runtime library (system unit as well as SysUtils) is almost the same in Kylix—almost because some typical Win32 items have been removed (such as RaiseLastWin32Error and Win32Check for example).

Interfaces

IInterface takes the place of IUnknown, but for backward compatibility, IUnknown is defined as IInterface. IDispatch requires COM and is gone in Kylix. Kylix introduces interface RTTI and portable variants.

Portable Variants

The new Variant implementation is called Portable Variants (implemented in the new Variants unit). These are variants that are binary-compatible between Win32 and Linux and are now implemented natively in the Delphi Object Pascal rather than through any use of Win32-specific APIs.

The new variant:

- includes support for variant arrays and safe arrays
- provides support for registering custom variant data types
- allows for data coercion, data copying, operator overloading, and method invocation.

Disks, directories, and files

Textfile operations and general file operations work differently in Linux as compared with Windows because Linux works on a case-sensitive file system. Areas in your application where "TABLE.DB" and "table.db" are assumed to point to the same table will have to be located and modified.

Also important is the fact that in Linux, directories use a forward slash (/) while in Windows, they use a backward slash (\). Also, there is no drive C: in Linux! These issues can be handled easily by making sure you already use the PathDelim constant which is defined in the SysUtils unit (and set to / in Kylix and \ in Delphi). For additional help, you should use the DriveDelim and PathSep constants, which are defined as follows:

```
const
  PathDelim     =  {$IFDEF  MSWINDOWS}   '\';
{$ELSE} '/'; {$ENDIF}
  DriveDelim   =   {$IFDEF   MSWINDOWS}   ':';
{$ELSE} '';  {$ENDIF}
  PathSep        =  {$IFDEF  MSWINDOWS}  ';';
{$ELSE} ':'; {$ENDIF}
```

Finally, the ExtractShortFileName function is gone in Kylix, as are DiskSize and DiskFree.

File I/O

As a related topic, where Windows uses a CR+LF pair (carriage return plus line feed) to mark the end of a line in text files, Linux uses only the LF (line feed). There is a global variable called DefaultTextLineBreakStyle in the System unit that is set to tlbsLF on Linux, which means that line breaks are simply LF.

You can override the value of DefaultTextLineBreakStyle on a file-by-file basis by calling the SetLineBreakStyle routine as follows:

```
SetLineBreakStyle(logfile, tlbsCRLF);
```

This will read or write the logfile using CR+LF pairs as end-of-line characters. Note that you cannot call this routine on an already open file!

Migrating APIs

When migrating/porting applications from one platform to another, it's unavoidable to encounter code that operates at the API level (the Windows unit versus the Libc unit). We'll look at a few examples, but it is best to avoid these kind of porting issues.

API calls

The Windows API calls are located in the Windows unit and also in related units that can be found in the Delphi5\Source\RTL\Win directory—obviously not available with Linux. The native Linux APIs are wrapped in the Libc, Xlib, and Xpm units for Kylix and in the Kylix\Source\RTL\Linux directory, which obviously is not available in Delphi 5.

Windows messages

Apart from straight Windows APIs, you should also try to minimize the number of Windows messages you send in your application. PostMessage and SendMessage will not work in Linux and will have to be replaced by calls to event handlers or notifiers.

IniFiles

The Windows IniFile is ported to Linux. However, the Windows Registry of course is not available with Linux. So while TIniFile will work just fine, TRegIniFile will not and should be replaced by TIniFile.

File time

Another platform-specific issue is the internal format in which dates and times are stored. This includes file dates, stored in the UNIX format. This is not a problem when conducting date comparisons, such as file date against file date, and against Now or Date. In fact, Date and Now are still using the offset 1 for December 31, 1899—both in Delphi and Kylix. The TDateTime representation is identical in Kylix and Delphi; the only difference is the file date stamp.

Conclusion

The ability to migrate existing Delphi for Windows applications to Kylix for Linux is very powerful. This capability is extended and enhanced with Delphi 6 and true cross-platform development using CLX.

After checking the availability of the used components in Kylix (and possibly implementing workarounds already), you should start your migrations by converting your project source files and database tables, followed by possible in-depth modifications of your Object Pascal source code. Most projects will port with very few changes; some won't port at all (such as with the ActiveForms example). WebBroker projects will port with few changes and generally work within hours, if not minutes, after you start the migration.

Bob Swart (a.k.a. Dr.Bob - drbob@chello.nl) is a Delphi consultant, trainer and Webmaster for Dr.Bob's Delphi Clinic (http://www.drbob42.com). He is a freelance technical author for *The Delphi Magazine, UK-BUG Developer's Magazine, Delphi Developer, Blaise, SDGN Magazine, DevX, TechRepublic,* and has written many chapters for programming books such as *Delphi 4 Unleashed, C++Builder 5 Developer's Guide* and the upcoming *Kylix Developer's Guide* and *Delphi 6 Developer's Guide*. Bob has been a regular speaker at the annual Borland conferences since 1993, and he writes his own training material for the "Dr.Bob's Delphi 6 Clinic" training days.

Borland

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000