Borland®

Cross-platform Development with Borland® RAD Tools

By Borland Software Corporation
September 2002

Contents

Introduction

An important choice when writing applications is which operating system is used for servers and workstations. Today, that frequently means choosing between Microsoft® Windows® and Linux® platforms.

PCs running Windows are often used at the desktop. The Linux platform, on the other hand, is used mostly for servers because of its reputation for low cost of ownership, high reliability, and excellent security.

Traditionally, developers create and maintain separate programs for Windows and Linux platforms. Borland® development tools now make it possible to target both Windows and Linux platforms with a single application. The component-based development environments compile to high-speed, native code, so programs run without compromise on both platforms. Without learning a new tool, Windows developers can easily create applications for Linux, and Linux developers can target Windows.

Borland gives businesses the freedom to choose the platform which best meets their development needs.

RAD Tools white paper

Borland® solutions: Delphi,™ C++Builder™ and Kylix™

Borland offers the software development industry a wide range of choice for application development. Building upon nearly 20 years experience in programming languages, today's family of Borland tools is consistently regarded as a fast, productive and dependable way to create applications. Borland understands development and has consistently created innovative solutions that make it easier for developers to create better systems faster.

Borland® Delphi™ and Borland C++Builder™ are award-winning Windows development solutions that increase developer productivity and speed. Borland also delivers the benefits of these technologies to Linux with Borland Kylix,™ a rapid application development (RAD) tool that provides both ANSI/ISO C++ and Delphi™ language programming—two powerful object-oriented languages in one Linux development solution.

The key to developer productivity is a well-designed component library. Delphi, C++Builder, and Kylix share a common Component Library for Cross-platform (CLX™), which simplifies development.

As well as enabling cross-platform user interface development, CLX has powerful features for server-side development for the Internet, middleware, and Web Services. The CLX library includes the following sublibraries:

DataCLX™—for scalable data access. This provides a single, consistent programming model for accessing relational databases.
DataSnap™—for construction of multi-tier applications. The DataSnap components support transactions, multiple threads of execution, fail-over across multiple servers, and other features that are essential for enterprise-class server development.
WebSnap™—for building Web server applications. These can work with data sources through the DataSnap architecture, creating multiple-tier systems with thin-client Web browser interfaces.

BizSnap™—to create and use Web Services using WSDL and SOAP. Web Services are now established as the standard way to exchange information between different applications and across different platforms, even between corporations and their partner organizations.

These components work together. For instance, the components in BizSnap can be combined with the components in DataSnap to create a multi-tier application that makes data available across Web Services interfaces. The approach to development is consistent across the components, reducing the time to learn how they work.

CLX can be extended by the developer or by third parties to create new capabilities. The source code for the library is included so experienced developers can understand the programming techniques that are used. It is possible to create a repository of new components that are based on the standard library but which include further extensions for a particular development.

These parts of the CLX library are the same for both Windows and Linux development, so developers only need to create and maintain a single shared set of program code. For example, a single program can be recompiled to work with Apache™ on Linux or on Windows. Developers therefore have a choice of deployment platforms as well as the benefits of great productivity.

Two additional sublibraries are provided for building user interfaces:

VisualCLX™—Controls for Linux which wrap standard Qt widgets.
VCL—includes controls for Windows which wrap Windows controls.

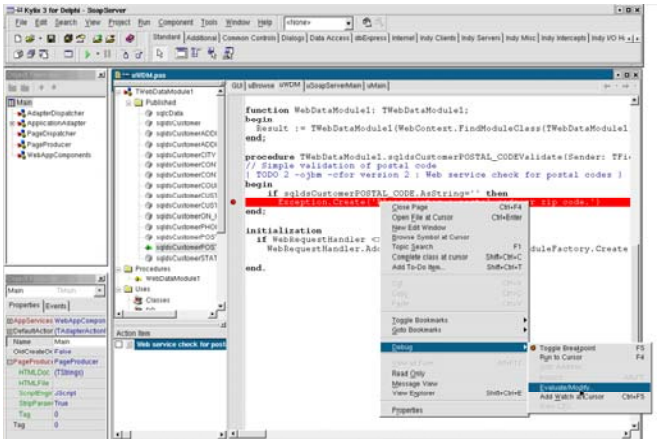


Figure 1: Full integrated debugging is included in Kylix, even across multiple processes on separate computers

As far as possible the controls in these two libraries are compatible with each other: they have the same names and the same properties. There are some differences in appearance and behavior so that applications take on the appropriate look and feel of their platform.

Developing for the Windows® platform

Borland Delphi and Borland C++Builder each provide comprehensive development capabilities for the Windows platform.

Creating applications using Delphi or C++Builder is very straightforward. Components are chosen from a palette and their initial properties are set. Business logic is then written, using either the Delphi language—based on ObjectPascal—or C++. The completed system is compiled natively for optimal performance.

This makes development extremely productive. The components are designed in such a way that in most cases they handle what the application needs to do without any extra code. The code that the developer does write is typically the business logic that their particular system requires to respond to events.

When writing code, features such as code completion and code templates speed the process even further. Compilation takes only a few seconds, allowing iterative development of even complex applications.

Developing for the Linux® platform

Borland Kylix is an integrated development environment similar to Delphi and C++Builder but for the Linux platform. Similar tools are provided to graphically build applications, write code, compile and debug. Kylix 3 supports both the C++ and Delphi languages.

Although it provides the same core development capabilities, Kylix is not simply a port of the Windows development tools. It includes additional features that are specifically tailored for Linux developer productivity, such as access to low-level libraries.

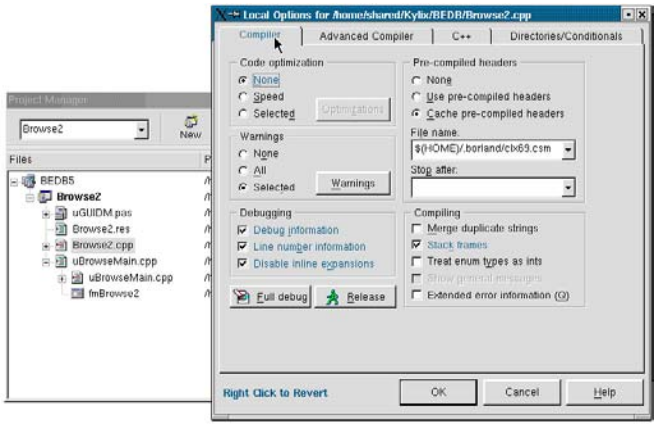


Figure 2: Kylix provides a fine degree of control over project build options

Developing cross-platform applications

Writing programs with each of the development tools is done in the same way, so those familiar with one system will find it very straightforward to move to one of the others.

Applications written using the CLX for Delphi can be loaded directly into Kylix, since much of the CLX is identical between Kylix, Delphi, and C++Builder. The differences between the two platforms are not necessarily apparent to the developer as they are hidden in the lower levels of the CLX library.

There are some Windows-specific capabilities in Delphi and C++Builder that are not available in Kylix. The Visual Component Library (VCL) components for ADO database servers, Windows 3.1 style interfaces, and Windows API calls are not appropriate for

use on Linux. In their place, Kylix offers regular CLX database components and access to Qt,® libc and other system libraries. With this cross-platform capability in CLX, a single application can be transferred between Linux and Windows and recompiled to create a native application on the other platform. Usually, very few changes are necessary.

CLX™ components and controls

The CLX library is made up of components, represented by icons on the tool's component palette. Any of these components can be placed on a form, a visual design surface that is available in a CLX application. The tool automatically creates the code necessary to make an instance of the component available in the application.

Some of the components provide a visual user interface when the application is run. These controls are used for conventional labels, buttons, text edit areas, images, and other familiar display items. Other components do not have a user interface, for example components to handle timers, database queries, and Web server page data. These are represented by icons on the form.

The instance of the component on a form can be selected in the designer, and the Object Inspector™ used to set its initial properties. Many components are interlinked through properties: for example, a data-aware grid control has a DataSource property that can refer to a TDataSource component instance.

The components all respond to events, which represent some outside change such as a button press, a data item changing, a form opening, or a clock timer expiring. Program code is written to handle these events. Again, the development environments automatically insert and handle the brief amounts of housekeeping code required for this.

Graphical client/server applications with DataCLX™

The DataCLX architecture provides a powerful mechanism to create client/server applications that work across platforms and which do not have a strong tie to a database from a particular vendor.

For the user interface, data aware versions of many of the CLX controls are available. When compiled into the application these controls appear like other controls on that platform—so buttons on Windows look and behave as they should, as do the same controls when compiled for Linux.
As the applications look and behave in the same way on Linux as they do on Windows, there is little to retrain users who move from one platform to the other—the system will continue to work as they expect.

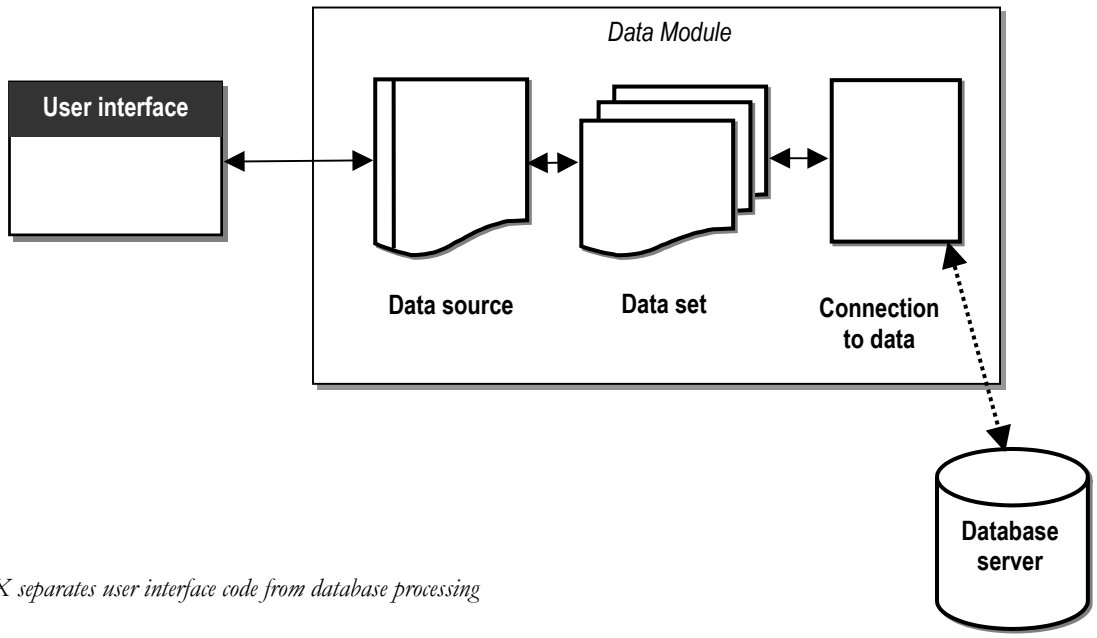


Figure 3: DataCLX separates user interface code from database processing

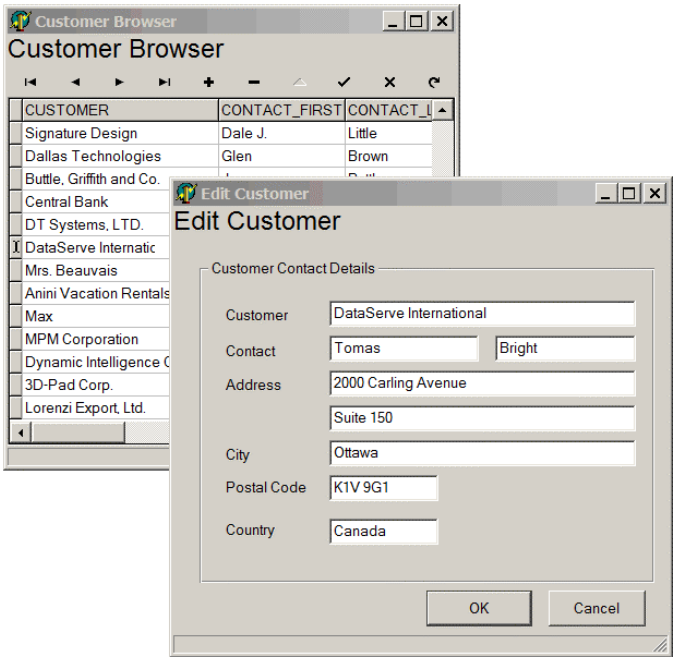


Figure 4: Delphi and C++Builder can construct fast, attractive user interfaces for Windows

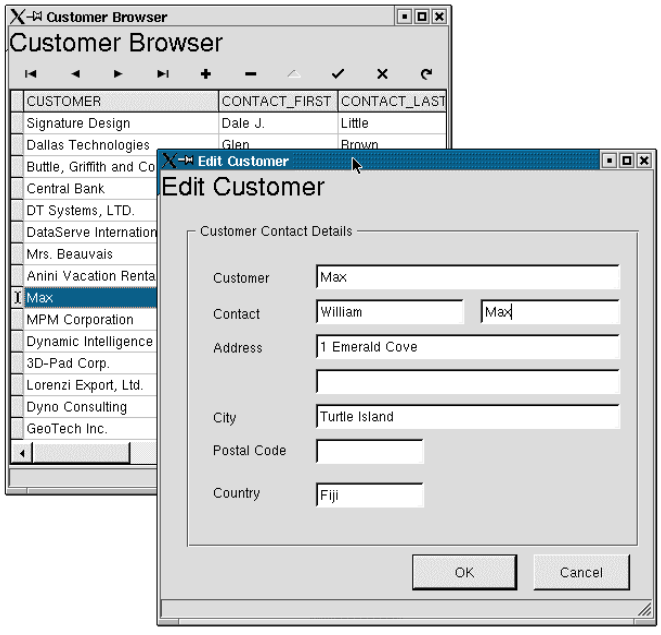


Figure 5: Kylix can create similarly fast and attractive user interfaces for Linux

The tools each include a special design surface known as a DataModule. These specialized forms provide a single point where components for data access can be collected together for an application. This centralization of components and their code makes it straightforward for developers to implement design patterns such as the model-view-controller approach, where data access is separated from data presentation and application control.

There are further levels of abstraction in the DataCLX components. Application developers need to ensure that their systems can easily move from one database to another, so the DataCLX components are designed to make it easier to create applications that are portable between databases as well as being cross-platform portable.

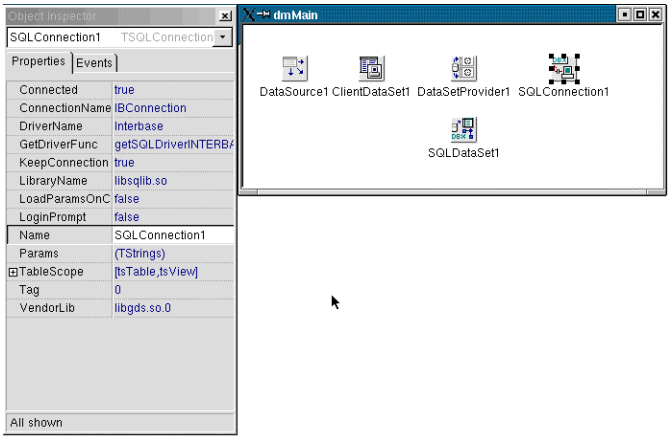


Figure 6: Data access can be centralized in one point in an application using a DataModule

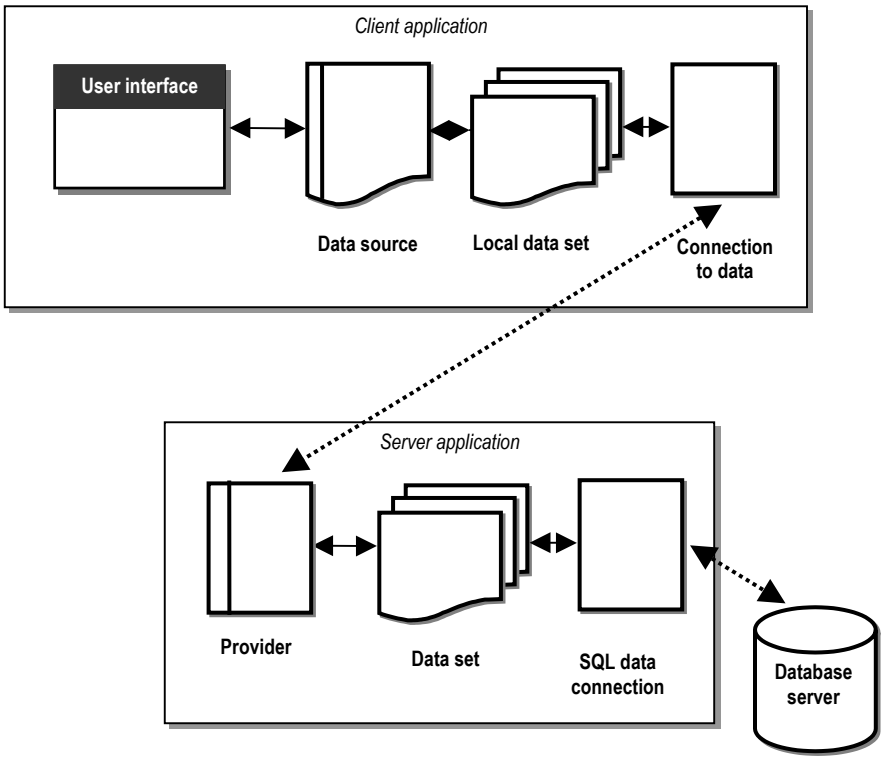


Figure 7: DataSnap creates multi-tier database applications

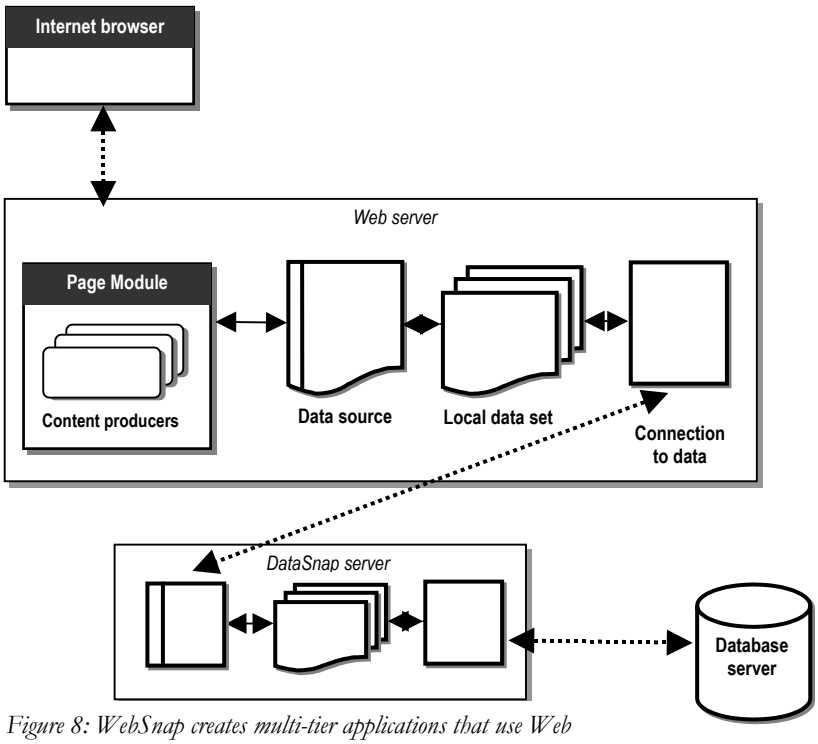


Figure 8: WebSnap creates multi-tier applications that use Web browsers as clients

Multi-tier application development with DataSnap™

Many applications need to have the middle-tier business logic and database processing centralized rather than running on every user's PC. Only the centralized servers need to be maintained, and in many cases performance can increase.

Through the DataSnap components in the CLX library, Delphi, C++Builder, and Kylix support multi-tier applications of this kind. Instead of the data processing running at a DataModule at the users' PC, a server application can be used to perform the processing centrally.

Windows systems and Linux systems can each freely interoperate through the cross-platform nature of the CLX. For example, a developer may choose to implement Windows clients that communicate with Linux servers. The same program code can be recompiled to later support Linux clients or Windows servers.

The DataSnap CLX components automatically support enterprise features such as high availability through automatic fail-over and high performance through multiple threading.

Web applications

Many enterprise applications are best delivered through a Web browser to the users' desktops. This can dramatically reduce the costs of deploying applications as well as increasing security and simplifying administration.

These applications run at the Web server and can be built using the WebSnap components in CLX. The components can be grouped together in a PageModule giving one central point where the application creates Web pages.

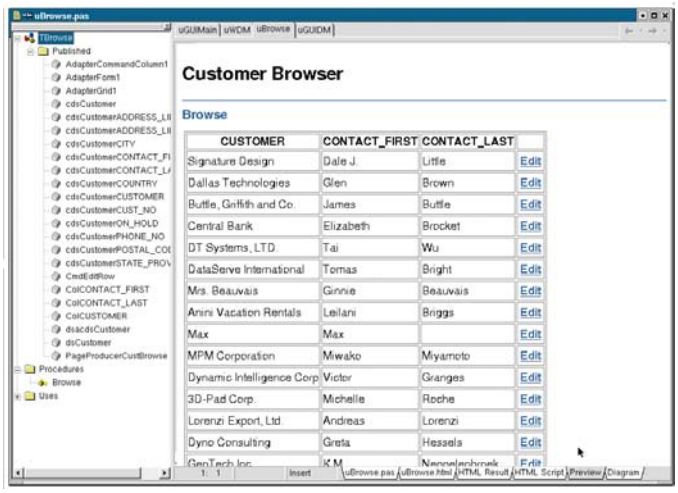


Figure 9: A Web page designer is included in Kylix (and Delphi and C++Builder) to simplify development of WebSnap applications

The components automatically handle complex issues like maintaining state and include pre-built modules that handle authentication.

Both the Linux and Windows tools support a range of Web servers through CGI, the Common Gateway Interface. They also both support direct links with the popular Apache Web Server.

The WebSnap technology is concerned with the presentation of the data to the user through a Web server. To build highly scalable Internet applications, the WebSnap components can use the DataSnap multi-tier technology for guaranteed availability and high performance. Indeed, the WebSnap system can share the same DataSnap data provider as applications with a user interface.

Through the cross-platform nature of CLX, developers can choose the most appropriate server architecture. The majority of Web servers run with Apache on Linux as this combination is secure, reliable, and fast. A Kylix application using WebSnap components can run on this server and communicate with DataSnap to a Windows system, for example.

Integration through Web Services

Few, if any, enterprise applications work in isolation. Most need to work with data from other systems either within your business, your partners, or customers. To ease this integration Web Services standards have been created for secure, reliable communication between systems.

Web Services can be used to build composite applications. For example, the Web Services provided by a freight forwarding company can be combined with a user support application to

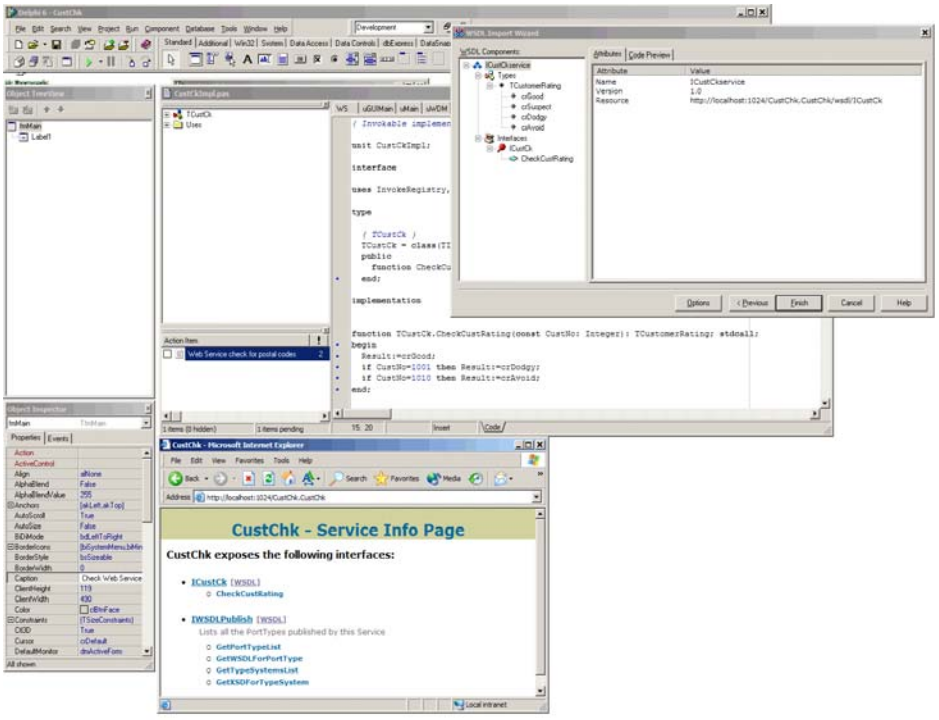


Figure 10: Web Services can be created using BizSnap

automatically dispatch and track spare parts.

The BizSnap technology in CLX makes it possible to use Web Services in applications, and also to publish them to make data available to other systems.

Development of applications that use Web Services is simplified, as BizSnap maps data provided through a standard WSDL interface to an object that can be used in program code. The programmer works with the object as normal, and BizSnap handles the communication using SOAP and XML to the Web Services host.

A set of tools and wizards is provided to create Web Services that can be accessed through a Web server. Once a Web Service is defined, program code can be written to connect it with other parts of an application. As for the WebSnap technology, these Web Services can work with data either directly through DataCLX or through DataSnap servers, possibly on different platforms.

Case Study: Colliers International

A multinational real estate agency with more than 300 offices, Colliers needed a system to extract and format data from their existing legacy systems and database.

The initial application was written for a Windows server using Borland® Delphi™ 6. This made the data available as a Web Service so it could deliver data to Colliers' brokers directly in many different formats.

With the Windows version proven for internal use, Colliers and their development partner have now created an enhanced version of the application for access over the Internet. This uses Linux® servers and is written using Borland Kylix.™ The port of the basic application from Delphi 6 took less than two hours.

Since the new Kylix version of the system uses the CLX, WebSnap, and BizSnap technologies, it supports many different formats, including access by wireless devices. The result is a fast, efficient system that can be used by brokers while sitting in a car with their client.

More information on Colliers International and other companies that use Kylix for development, is available on the Borland Web site at http://www.borland.com/kylix

Alternative approaches

No other major software vendor publishes a cross-platform integrated development environment that creates native applications. The closest alternative is to use a programming language along with a cross-platform library such as Qt or GTK.

While libraries such as GTK are rich and provide good low-level control, they do not provide the same integrated component-based development as the Borland tools. This makes it harder to create reusable objects and means the tools are not as productive. The Borland tools each provide similar low-level control from either the Delphi language or C++.

The Qt system extends C++ to support properties by providing a preprocessor, the Meta Object Compiler. Qt includes classes that provide links to databases but does not include as comprehensive a class library as CLX.

Another alternative is to use a solution like Java.™ This provides an effective way for applications to work across different platforms but does not give the same performance or low-level access to the operating system as a natively compiled application.

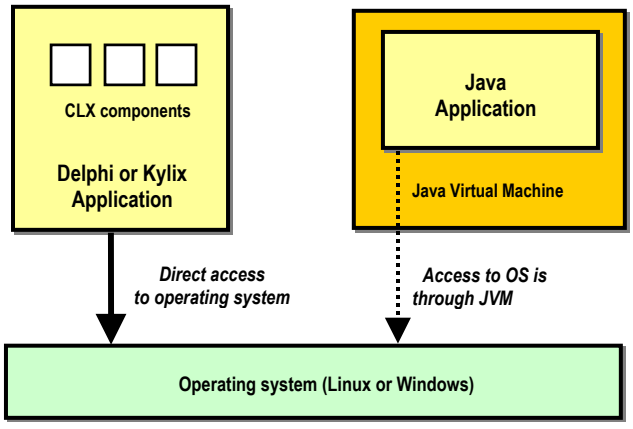
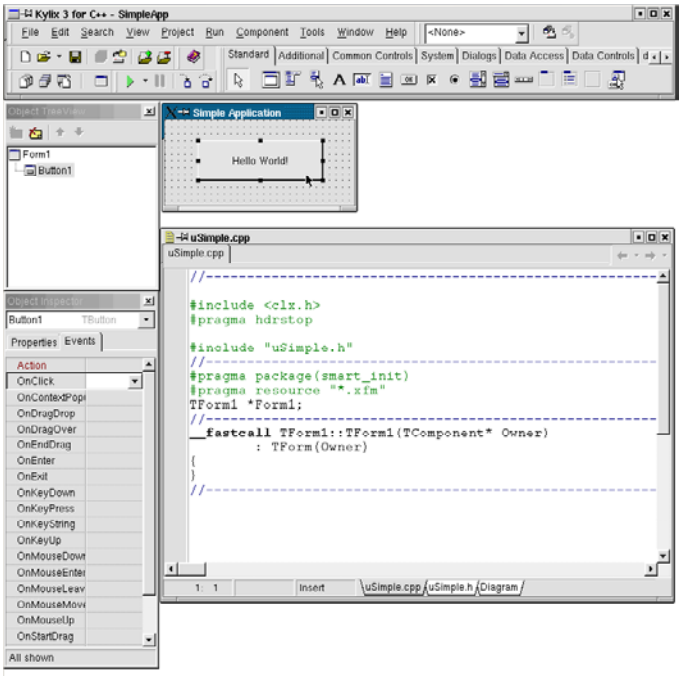


Figure 12: Unlike Java, CLX applications run directly on Windows or Linux, giving high performance

Comparing Kylix™ and other cross-platform tools and libraries

There is a great contrast between development using a fully integrated tool such as Kylix and stand-alone editors and compilers.

As an example, consider a simple application that paints a window on the display with a button labeled "Hello World". Using Kylix or C++Builder, this can be done using the IDE without creating any code at all.



Making this button do something is as simple as choosing the appropriate event in the Object Inspector™ and entering code.

It is more complex to use a graphics library such as GTK. The C code to create a similar program might be:

```
#include <gtk/gtk.h>

/* Callback function. */
void hello( GtkWidget *widget,
            gpointer   data )
{
    g_print ("Hello World\n");
}

gint delete event( GtkWidget *widget,
                   GdkEvent  *event,
                         gpointer   data )
{
    g_print ("delete event occurred\n");
    return TRUE;
}

/* Another callback */
void destroy( GtkWidget *widget,
              gpointer   data )
{
    gtk_main_quit ();
}

int main( int   argc,
          char *argv[] )
{
    GtkWidget *window;
    GtkWidget *button;

    gtk init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    g_signal_connect (G_OBJECT (window), "delete_event",

    g_signal_connect (G_OBJECT (window), "destroy",
                      G CALLBACK (destroy), NULL);

    gtk_container_set_border_width (GTK_CONTAINER
                      (window), 10);

    button = gtk_button_new_with_label ("Hello World");

    g signal connect (G OBJECT (button), "clicked",
                      G_CALLBACK (hello), NULL);

    g_signal_connect_swapped (G_OBJECT (button), "clicked",
                      G_CALLBACK
                      (gtk_widget_destroy), window);

    gtk container add (GTK CONTAINER (window), button);

    gtk_widget_show (button);

    gtk_widget_show (window);

    gtk main ();

    return 0;
}
```

As can be seen, almost all of this code is concerned with the housekeeping of creating windows and handling events. Using a toolkit like GTK means that the developer needs to write this additional code. With Borland tools, all of the window management code is handled by CLX. Events are still available for the programmer to override if necessary.

The Borland® community

Together with the technology, a key strength of Borland tools is the strong community of developers. The Borland® Developer Network hosts thousands of components, hundreds of applications, and dozens of white papers, which are relevant to development for both Kylix and Delphi. The Borland community assembles in a conference each year to share experiences and discuss new developments.

Summary

Borland tools are known for their productivity and ease of use. With the release of Kylix, the maturity and depth of Delphi and C++Builder is now available for the Linux platform.

For Windows developers, this gives an alternative deployment platform with considerable strengths in security and reliability; for Linux developers, this provides a flexibility to create Windows versions of applications if demanded.

For all developers, it means that the operating system need not be a constraint as an application is developed, deployed, and maintained.

To learn more about Borland development products, please visit: http://www.borland.com.

White papers and other resources for cross-platform development

The Borland Developer Network at **http://bdn.borland.com** has a wide range of resources on development using Delphi, C++Builder, and Kylix.

A quick tour of Kylix™

Borland Staff

A straightforward introductory explanation of how to develop applications with Kylix. The development process is very similar for Delphi.

Bring the power and speed of RAD to your Web application development with WebSnap

Nick Hodges

Outlines the structure of a WebSnap application. Includes a comparison with other technologies, including ColdFusion,® Active Server Pages and the CGI.

Creating Kylix Database Applications

Bill Todd

Explains how the database architecture of Kylix functions. The same dbExpress and components are available in Delphi.™

Kylix versus gcc development

William Roetzheim

A direct comparison of the cost of developing a Linux application with gcc against Kylix. Dramatic savings in cost are identified across the development life cycle.

Migrating to Kylix from Delphi 5

Bob Swart

Programmer reference for those planning to migrate from earlier versions of Delphi to Kylix

Apache Development with Kylix Server Developer

Colin A. Ridgewell

Illustrates the WebBroker™ technologies for Apache in Kylix. Most of the development techniques in this paper also apply to Apache development for Windows with Delphi 6.

Introduction to Web Services

Hartwig Gunzer

Explains the relationship of Web Services to other systems such as CORBA,® RMI and DCOM. Gives a developer-level explanation of how Web Services are constructed, showing the low-level structure of Web Services protocols and systems.

Generating XML and Delivering It Across the Web

Keith Wood

Illustrates how WebBroker can generate formatted XML rather than HTML files

The Kylix Object Model

Ray Konopka

Kylix and Delphi both allow the developer to construct their own components, and both products include the complete source code for the CLX class library. This paper outlines the techniques used to create objects.

Technical Overview of the Kylix Compiler

John Ray Thomas

Underlying Kylix is a high-speed optimizing native code compiler. This paper outlines the low-level capabilities and strengths of the compiler.

Borland

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