Borland® C++BuilderX™—Taking C++ Into the Future Without Abandoning the Past

The history and evolution of Borland C++: a step-by-step guide to moving forward

A Borland White Paper

March 2004

Borland®

Contents

Borland's rich history with C++

Borland is a leading name in C++ software development solutions. To date, over 1 million developers have chosen Borland for their C and C++ development needs. Borland's history with C++ goes back to its beginning in the early 1990s and Borland has made history several times over with innovative approaches to C++ development.

This white paper discusses the historical, innovative Borland approach to C and C++ language development, followed by a step-by-step guide to leveraging existing investments in software applications.

Borland® Turbo C:® where it all began

In the late 1980s, Borland released Borland® Turbo C,® an expansion from the extremely popular Borland® Turbo Pascal® product line, to support the fast growing C –based application development community. The introduction of the Integrated Development Environment (IDE) proved to be highly productive for developers. The speed of the compiler was extraordinary compared to the Microsoft C compiler, which saved significant time in developing applications because developers had a much faster trunaround for checking their work. By coupling the IDE with a top-notch compiler and debugger, for a fraction of the price of its competitors, Borland quickly became a market leader for C language-based application development tools.

Borland focused primarily on supporting Microsoft DOS, but also supported the IBM® OS/2® operating system. Borland also provided extensive standard application functionality such as screen Input-Output (IO) text functions that made User Interface (UI) development easy to implement. This proved to save developers significant development time on platform-specific code so that they could instead focus on their particular application requirements.

Turbo C 2.01 provided all of the tools in one environment—a radical approach to software development at the time. These tools provided tight integration between the editor, compiler, linker, and debugger (the first version of Turbo C to include the integrated debugger), and allowed the developer to have access from a single environment. The Professional Edition also included the standalone versions of Turbo Assembler® and Turbo Debugger®. The inclusion of these tools further strengthened the developer's toolbox.

The Turbo C compiler compiled over 16,000 lines per minute. It included hypertext online help, supported all six memory models, and inline assembly. It included more than 450 library functions and Intel386™ Processor support was included. Unlike previous offerings, project files made up of multiple source files could easily be built and maintained, saving significant development time. Borland provided full source code with the libraries so that they could be changed and recompiled if needed—and pioneered a new way to package development software in the industry.

In its day, the Turbo Debugger was very powerful. It could be run in extended memory, which left the lower 640k of memory available for the application being developed. It could browse through structures with data debugging, set conditional breakpoints, and break on memory access. It could log expressions and included 386 ICE capabilities. These advanced debugging abilities allowed the developer to quickly and concisely find defects in the software in an intuitive, visual manner.

Turbo C from Borland gave C developers their first taste of an integrated development environment. The integration of an editor, build engine, and debugger provided tangible productivity benefits to developers, including a one-stop-shop for all their development needs. This was the start of a continuing approach to innovative development solutions from Borland for C language developers.

Turbo C®++: Borland's answer to C++

In order to support C++, a nascent language with great promise due to its object-oriented language features and superb C language compatibility, Borland released Turbo® C++ in the early 1990s. Turbo C++ had all the features of Turbo C in addition to support for C++. The C++ compiler conformed to the AT&T 2.0 Specification for the C++ language, what many at the time considered to be the de facto, if not the first, standard implementation. This was Borland's first step in demonstrating its commitment to industry standards, and the flexibility that supports the customer.

The development environment and command line tools ran under DOS. Turbo C++ v1 came in two editions: Turbo C++ and Turbo C++ Professional, which included Turbo Assembler,® Turbo Debugger, and Turbo Profiler.™ With these tools, the developer was equipped with complete solution to perform the work in a productive manner.

Turbo C++ introduced TurboVision, a GUI library for the DOS operating system that provided application logic for managing menus, buttons, dialogs, borders, etc.
The Turbo Vision library exemplified the power of the promise of an object-oriented landguage like C++ to provide re-usable functionality, presented in a simplified, pre-packaged interface. With the combination of included and integrated tools in the application library, Borland continued to increase developer productivity by providing the ultimate product solution.

Borland® C++: Windows® is the way

In 1992, Borland released Borland C++ and ObjectVision® 2.0 for OS®/2 2.0. When Microsoft moved from DOS to the Windows® platform, Borland continued to provide compelling development environments for the new platforms. The product line was called Borland C++ and in the spirit of the Turbo product lines, provided a full IDE, compiler, and debugger for the Windows platform, while continuing to support the DOS platform. This demonstrated Borland's understanding of the importance of supporting existing solutions while providing future solutions to help the customer migrate.

The Borland C++ product line also popularized a new C++ framework from Borland, specifically for the Windows® platform, called Object Windows Library (OWL). OWL was an object-oriented framework that provided support for standard application functionality and GUI controls for Windows. OWL represented an evolution of the application framework and the experience gained from Turbo Vision enabled Borland to create an extremely productive framework for these new GUI platforms.

Borland C++ version 3 and later version 3.1 became a popular ~~standard~~ development environment for 16-bit Windows development under Windows 3.0 and 3.1. It was a fast IDE and compiler, which used OWL version 1.0, that utilized Dynamic Dispatch Virtual Tables (DDVTs) for handing Windows messages. It included support for streamable objects, custom controls, Dynamic-link library (DLL) creation, and global heap suballocation that did not fragment memory like normal farmalloc, farfree, farrealloc, and farcalloc could. This support allowed the developer more flexibility and generated more performant applications.

OWL was tremendously popular and Borland was a market leader for through the early to mid 1990s. By the mid 1990s, Borland was the only independent provider of C++ solutions on the Windows® platform, and Microsoft® was its only viable competitor. Around that time Borland released Borland C++ 4.0, which supported 32-bit development. Full source for OWL, the Classlib, and RTL was included so that they could be changed and recompiled if needed. The inclusion of source gave the developer even more flexbility to make changes to the core libraries to support custom application requriements and to actually identify where application defects may live, in their application source or in the Borland framework.

Borland C++ 4.0 came with a new version 2.0 of OWL that had support for more features, multiple inheritance, exception handling and OLE. There were point releases to repair bugs in the product and improve Object Linking and Embedding (OLE) support. Version 2.5 of OWL included complete encapsulation of OLE 2 using ObjectComponents, templates for ObjectComponents classes, new classes that manage UI handles, and hatched brushes.
32-bit programs could use existing Visual Basic Extention (VBX) controls, new data type definitions and new 32-bit dispatch functions. With these changes, Borland showed its commitment to supporting the latest in platform functionality which provided developers with a competitive edge.

Borland C++ 5.0 was released with support for OLE 2.0 applications, including containers, servers, and automated applications, using the ObjectComponents Framework. Full exception handing was added. Point releases 5.01, and 5.02 were released that resolved many of the quality issues of 5.0. Borland C++ 5.02 was used to create Windows® 32-bit, 16-bit, and DOS® applications, and included the release 5.01 of OWL. It was the last version released but importantly underscored Borlands understanding and support for existing application requirements in addition to supporting future application requirements.

OWL 5.01 included complete encapsulation of Windows Common Controls and additional ObjectWindows Controls, such as glyph buttons, check listboxes, and notetabs. Improved toolbar and status bar implementation with docking support and new gadgets, OCX Container support, and the TOleWindow which could host OCX controls were included. OWL 5.01 also provided emulation of the Column Header common control when running 16-bit applications on Windows® v3.1 or Windows® NT. OCF/OLE improvements included support for the Automation VARIANT type via the TAutoVal class. These improvements continued to assist the developer in fully supporting the new technology on the platform.

The Borland C++ product line innovated development on the Windows platforms by supporting the wide range of Windows platforms within a single environment. Importantly, Borland showed a commitment to supporting the past while moving developers into the future with its support for both 16-bit and 32-bit development within the same IDE product. This primarily aided developers in their forward migration while continuing to support their every day requirements.

Borland® C++Builder:™ Rapid Application Development for C++

By 1997, Borland had seen remarkable success with the Borland® Delphi™ product line, an evolution of the Turbo Pascal product line, which provided a RAD (Rapid Application Development) solution for Windows with native compilation. Delphi innovated the RAD market with the introduction of its object-oriented component framework for Windows called VCL (Visual Component Library). The main competitor to Delphi was Microsoft® Visual Basic.® Borland addressed this hot market with support for C++ developers and Borland® C++Builder™ was released early that year. C++Builder was essentially Delphi for C++, in that the C++ product line moved to the Delphi IDE and could enjoy the same RAD experience with the Delphi VCL framework. This was a revolutionary approach to development productivity and for the first time Borland enabled C ++ developers access to the RADical development methodology.

The first version of C++Builder was an object-oriented, visual programming environment for Rapid Application Development of general-purpose and client/server applications for Microsoft® Windows® 95 and Windows® NT. Using C++Builder, highly efficient Windows applications could be created with a minimum of manual coding. It provided a comprehensive library of reusable components and a suite of RAD design tools, including application and form templates, and programming experts. When C++Builder program was was opened on the computer; the developer was immediately placed within the visual programming environment. It was within this environment that C++Builder provided all the tools needed to design, develop, test, and debug applications. C++Builder also included the Borland® Database Engine (BDE). Using the BDE, and database components, access to many different types of databases was quick and easy. C++Builder showed C++ developers how productive they could be with a RAD framework and environment and provided this support for key industry technologies, such as database, to fully enable the developer to be successful and to meet application requirements.

Version 6 of C++Builder delivered better ANSI/ISO C++ conformance for power and performance and provided a high-productivity development environment to create efficient cross-platform e-business applications with the included Borland® Kylix™ development environment for the Linux® platform. The BizSnap™ Web Services development platform enabled simplified business-to-business integration by easily creating industry-standard SOAP/XML Web Services and connections. It increased the power, speed, and efficiency of the development process with WebSnap,™ the component-based Web application development platform. With DataSnap,™ unlimited royalty-free data-access middleware solutions could be built that integrate with many business processes and many business partners. With the addition of these features, C++Builder 6 (with Kylix 3) enabled developers to support multiple platforms, Windows and Linux, and to build solutions that integrate existing enterprise functionality—a new shift in direction in today's enterprises.

With C++Builder, Borland raised the bar of developer producitivy, yet again, for C++ developers on the Windows and Linux platforms. The framework had become a central role in this highly resourceful development methodology (RAD) and C++Builder provided developers key advantages over competitors through greatly minimized development time and effort.

Borland® C++BuilderX:™ the multiplatform unifying solution

C++BuilderX is a new C++ product offering from Borland that is set to address key trends and issues within the software industry. With C++BuilderX, developers can now take advantage of the latest in application lifecyle trends and IDE integration without having to give up existing investments in compiler technology or existing code. The key industry trends are detailed below:

- *Multiple platform support:* A sizable number of today's C++ applications are deployed on or are targeted at multiple platforms. As a result, writing standard ANSI/ISO C++-compliant code once and recompiling for other targets is highly desirable. Unfortunately, this is not always possible because of the disparity and incompatibilities between the numerous C++ tool chains (A tool chain is the set of tools to build an application from source to binary executable, i.e. compiler, linker, librarian, etc.) . These differences are even more prevalent amongst those applications targeting UNIX® platforms requiring migration to Linux platforms with Intel processor support.
- *Support and maintenance of existing C++ code:* Analysts predict that professional C and C++ developers will remain the largest community of developers through 2005[1]. The C and C++ languages have been used for several decades and there are millions of lines of code that need to be maintained. In some cases, this code is very fragile, original developers are no longer available and weak documentation has made extension/modifications nearly impossible. C++ developers need the latest in application lifecycle tools to manage this effectively.

[1] IDC, Professional Developer Model, 2001

- *Performance Computing:* C++ remains the choice for highly scalable and high-performance applications. Fast runtime performance, small size of code, and direct hardware accessibility still make C++ the language of choice for building the types of applications frequently found in the finance, manufacturing, telecom, defense and automotive industries. Many of these industries' developers are using non-integrated tools and are eager for solutions that allow them to build better software faster.
- *Mobile and embedded device computing:* As many other languages support the multitude of new mobile, handheld, and embedded devices, there has been a strong commitment for C++ as the language of choice for these platforms. Because small application size and high speed are essential to building these applications, the majority of mobile application developers are building applications in C++. Today, many of these developers are using low-level tools and libraries to deliver small and fast applications. The complexity, breadth, and depth of these development environments are often a barrier to entry for the large number of application developers wanting to focus on the business process to be implemented instead of the technology required to build these applications.
- *Standards and C++ portability*: After a temporary moratorium on changes to the ANSI/ISO C++ 1998 standard, the standards committee is accepting proposals for new directions to the standard. Today, many compilers are still aiming to achieve compliancy with the current standard. Compliance to the ANSI/ISO standards is key to delivering on the promise of C++ portability.

To address these issues, Borland C++BuilderX focuses on the following goals:

- *Build a development tool for the entire C++ market*: C++BuilderX focuses on delivering the right IDE solution for the majority of C++ developers. The project manager, code editor, compilers, and debuggers are all tightly integrated and designed to cater to the needs of C++ development on multiple platforms.
- *Support for application lifecycle management (ALM) tools:* In order to truly deliver the complete solution for the C++ developer, C++ BuilderX integrates with the latest and greatest Borland® ALM technologies.
- *Lead in C++ standards compliancy:* Borland is committed to building a new line of C++ compilers that offer developers the latest and most complete ANSI/ISO C++ and C99 standards compliancy on both Intel and ARM platforms. The compiler architecture is

designed to deliver both a new front-end and back-end technology. This new back-end technology has been designed to provide the ability to target new hardware platforms, deliver new optimization capabilities, and plug in to a common compiler front end, promoting code portability across platforms.

- *Provide a true C++ RAD visual development environment:* The Borland commitment to enable developers to build better software faster is fulfilled with a new visual development environment and support for multiple platform frameworks for all C++ developers. This pure C++ framework does not require the use of any compiler extensions, and allows developers to easily build and deliver cross-platform applications with any standard C++ compiler.
- *Improved maintenance:* Borland understands the importance of on-going product maintenance. Built into the product is support for submitting deficiency reports and tracking any patches and updates through an enhanced Borland Quality Central interface. Borland is committed to a regular maintenance schedule and conforming with evolving ANSI standards, including scheduled product updates.

Borland C++BuilderX provides a robust and flexible solution for developing and deploying C++ applications. The C++BuilderX IDE runs on Windows, Linux, and Solaris and is designed to provide a consistent and unified interface across these platforms, bringing them into one development management view. Operating in a consistent interface can minimize the time developers spend training for, configuring, and interacting between the platforms. C++BuilderX provides tight integration for project and build management and allows developers to manage or migrate applications across platforms more easily. This support directly address the needs of C++ developers to support multiple platforms.

Borland C++BuilderX includes built-in support for multiple popular C and C++ compilers and debuggers,. This enables developers to utilize these disparate tools to build and debug applications within a uniform, integrated interface independent of the debugger being used for increased efficiency and productivity. C++BuilderX also provides the freedom to use custom compilers within the same development environment. This allows the ability to leverage existing IT investments in C++ technology.

Borland® Enterprise Studio for C++ is a complete solution for managing the entire application lifecycle, Borland® Enterprise Studio for C++ and delivers the tools the development team needs for defining, designing, developing, testing, and deploying enterprise C++ solutions. All of the components in Borland® Enterprise Studio for C++ are integrated to work together right out of the box, so the team can spend time developing rather than trying to get their tools to work together. With a single, proven source for licensing, support, and other services, Borland® delivers all these benefits of ALM and conveniences in a single, economical package

To address the performance requirements of most C++ applications, C++BuilderX ships with the Intel® Software Development Solutions on the Windows and Linux platforms. Performance bottlenecks can be pinpointed with the Intel® VTune™ Performance Analyzer, which collects and displays system-wide views with analysis of function, threads, module, source line, and individual instructions, helping identify and address performance issues. Intel® C++ Compilers are designed to maximize the efficiency of code for optimum run time speed on Intel® architecture. Intel® Performance Libraries gives the ability to focus on building value into the applications and bringing solutions to market faster by providing pre-built functionality. The Intel® Integrated Performance Primitives (Intel® IPP) provide a rich set of features from which developers can choose while designing and optimizing an application with multimedia functionality. The Intel® Math Kernel Library provides the framework for building solutions involving complex calculations, often used in scientific, engineering and financial software. With this support Borland delivers on supporting the performance requirements of the C++ developers.

BorlandC++BuilderXMobile Edition includes mobile development that drives productivity with a robust C++ development environment with timesaving tools and wizards designed to simplify mobile application development. Remote (or “on-target”) debugging and deployment of C++ applications to the target, streamlines the testing phase. The complete development environment helps speed the process, giving the ability to create better C++ code for mobile devices, faster. Through this mobile support, C++BuilderX can help the developer extend their enterprise applications into the mobile arena to address the enterprise needs to support their field.

.C++BuilderX comes in five editions: Personal, Mobile Edition, Developer, Enterprise, and Enterprise Studio.

A step-by-step guide to moving forward

Is it possible to take advantage of the investments already made in enterprise C++ development, without having to start over? With Borland C++BuilderX, it is not only possible, it is .

There are several ways for developers to continue to using and supporting existing projects that utilize customer or legacy compiler tool chains. The easiest way is by using existing make files. Below is a step-by-step guide for leveraging projects created in legacy Borland solutions.

Project from Makefile Wizard

Turbo C, Turbo C++, Borland C++, and other compilers that support or create makefiles can be utilized in the C++BuilderX environment by creating a new project with the Project from Makefile Wizard. C++BuilderX supports makefiles since and can use other compilers to generate the code. C++BuilderX comes with a command line version of make.exe that it calls to process the makefile. The parameters for the call to make.exe are in the Properties of the makefile. The parameters can be set by right clicking on the make file node in the Project content tab and selecting Properties. To use another make.exe program, the name of the program to call for make can be changed in the properties. The make file that is used has to be modified to call the right compilers with full path locations to create the application. This allows the developer to quickly pull in existing projects to get started more quickly.

Turbo C++

Use the PRJ2MAK.EXE utility that can be found in the TC\BIN directory to create the make file. For example, if TC is installed in C:\TC, and using the C:\TC\EXAMPLES\TCALC project use these steps:

1. At a DOS prompt, change directories into the C:\TC\EXAMPLES\TCALC directory.

2. Enter c:\tc\bin\prj2mak TCALC.PRJ TCALC.MAK

3. Edit the resulting C:\TC\EXAMPLES\TCALC.MAK, and change the lines shown below:

```
CC = ..\..\bin\tcc +TCALC.CFG

TASM = ..\..\bin\TASM

TLINK = ..\..\bin\tlink
```

Change to:

```
CC = c:\tc\bin\tcc +TCALC.CFG

TASM = c:\tc\bin\TASM

TLINK = c:\tc\bin\tlink
```

4. In C++BuilderX, select **File-New**, then in the dialog click **Project** from MakeFile.

5. Enter C:\TC\EXAMPLES\TCALC\TCALC.MAK as the makefile, name the Project TCALC, and click **OK**.

6. This creates the project, right-click on the **TCALC.MAK** node and select **Make**.

7. This will build the project, but there will be a Fatal error at the end; this is because All is not defined in the make file. The project was already built before this error. There will be a new TCALC.EXE in the C:\TC\EXAMPLES\TCALC\TCALC directory.

8. The project files can now be edited in the C++BuilderX environment by clicking on the **TCALC** node, and double clicking any of the source files. Even classes are viewable by clicking on the **Class** browser tab.

Turbo C

Use the PRJ2MAK.EXE utility that can be found in the TC\BIN directory to create the make file. The process is much the same as described above for Turbo C++.

Borland® C++

This example uses Borland C++ 5.02 installed in C:\BC5, and sample project C:\BC5\EXAMPLES\OWL\GAMES\BLAKJACK. Follow these steps:

1. Open the C:\BC5\EXAMPLES\OWL\GAMES\BLAKJACK\ blakjack.ide in Borland C++ 5.02.

2. Use **Project-Generate** makefile.

3. Edit the resulting C:\BC5\EXAMPLES\OWL\GAMES\BLAKJACK\ blakjack.mak, change the lines shown below:

```
IMPLIB  = Implib

BCC32   = Bcc32 +BccW32.cfg

BCC32I  = Bcc32i +BccW32.cfg

TLINK32 = TLink32

ILINK32 = Ilink32

TLIB    = TLib

BRC32   = Brc32

TASM32  = Tasm32

IDE_LinkFLAGS32 =  -LC:\BC45\LIB
```

Change to:

```
IMPLIB  = c:\bc5\bin\Implib

BCC32   = c:\bc5\bin\Bcc32 +BccW32.cfg

BCC32I  = c:\bc5\bin\Bcc32i +BccW32.cfg

TLINK32 = c:\bc5\bin\TLink32

ILINK32 = c:\bc5\bin\Ilink32

TLIB    = c:\bc5\bin\TLib

BRC32   = c:\bc5\bin\Brc32

TASM32  = c:\bc5\bin\Tasm32

IDE_LinkFLAGS32 =  -LC:\BC5\LIB
```

4. Close Borland C++ 5.02, or close the project.

5. In C++BuilderX, select **File-New**, then in the dialog click **Project** from **MakeFile**.

6. Enter C:\BC5\EXAMPLES\OWL\GAMES\BLAKJACK\ blakjack.mak as the makefile, name the Project blakjak, and click **OK**.

7. This creates the project, right-click on the **blakjack.mak** node and select **Make**.

9. This will build the project, but there will be a Fatal error at the end; this is because All is not defined in the make file. The project was already built before this error. There will be a new blakjak.exe in the C:\BC5\EXAMPLES\OWL\GAMES\BLAKJACK directory.

10. The project files can now be edited in the C++BuilderX environment by clicking on the **TCALC** node, and double clicking any of the source files. Even classes are viewable by clicking on the **Class** browser tab.

Custom toolset implementation

C++BuilderX uses toolsets to build projects without the use of makefiles. Toolsets are necessary for tighter integration with the project manager. This tighter integration provides the ability to manage the toolset visually and set or create new build configurations (in addition to the Debug and Release configurations), set project properties on a per toolset basis or across multiple toolsets, and to easily move from one toolset to another. By writing a custom toolset, developers can support their preferred and/or custom compiler needs.

C++ Builder comes with a group of existing toolsets that support the most popular compilers on the supported platforms. These toolset files are XML and can be copied and modified to create new custom toolsets. They are found in the C++BuilderX toolsets directory and provide quick reference upon which to base your new toolsets. There are also HTML documentation files in this directory that describe the format of these files.

The following is the methodology one would follow to create a custom toolset. In this example, Borland C++ 5.02 is discussed.

1. Open the IDE project in Borland C++ 5.02.

2. Use **Project-Generate** makefile.

3. Transform the makefile into a batch file to explicitly invoke the compiler and linker.

4. The information gained by creating the batch file are the options that are needed to create the custom toolset.

5. The Borland Win32 Compiler Tools Toolset that comes with C++BuilderX is very close to the one needed for Borland C++, so one could copy the Borland Win32 Compiler Tools Toolset XML files into another set with a different name.

6. Modify the new XML files to match the options needed.

7. Validate the resulting XML files.

8. Select **Tools | Reload Toolsets** from the menu.

9. The new Toolset will now be available for use in C++BuilderX and can be managed by the Build Options Explorer.

Supporting Visual Component Library (VCL) applications

Moving C++Builder applications into C++BuilderX is a very simple process. C++BuilderX comes with the Borland C++ compiler version 5.5 The Borland Win32 Compiler Tools Toolset can be used for C++ Builder version 6.0 since both use the same compiler. To use with other versions of C++ Builder, a new Toolset may need to be created that calls the compilers from that version An example of how to use C++BuilderX with a C++Builder 6.0 VCL application is described below. (Follow-up white papers will discuss forthcoming VCL support and migration into C++BuilderX)

For this example, the CustomDraw example will be used, and C++Builder 6.0 has been installed to C:\cb6. The example is in C:\cb6\Borland\CBuilder6\Examples\CustomDraw, the include directories are C:\cb6\Borland\CBuilder6\Include and C:\cb6\Borland\CBuilder6\Include\Vcl, the lib directories are C:\cb6\Borland\CBuilder6\Lib\Release, C:\cb6\Borland\CBuilder6\Lib\Obj and C:\cb6\Borland\CBuilder6\Lib. Follow the steps below:

1. In C++BuilderX, select **File-New**, then in the dialog click **New GUI Application**.

2. Click the "…" button to browse and select the C:/cb6/Borland/CBuilder6/Examples/CustomDraw Directory. This will automatically fill in the Name with CustomDraw; if not, enter CustomDraw in the Name, and click **Next**.

3. Remove the check under Windows, next to Minimalist GNU For Windows. This will leave only Borland Win32 Compiler Tools (Active) checked.

4. Click Finish.

5. In the Project content tab right click on the **CustomDraw.cbx** and select **Build Options Explorer**….

6. Select **ILINK32 Tool**

7. In the **Paths and Defines** tab, uncheck the Merge checkbox, and click the "…" button; in the next dialog, click **Remove** to remove the $(BCB)\lib. Then enter C:\cb6\Borland\CBuilder6\Lib\release and click Add. Then enter C:\cb6\Borland\CBuilder6\Lib\obj and click Add. Then enter C:\cb6\Borland\CBuilder6\Lib and click Add. Then click **OK**. This should match what is found in the customdraw.bpr "LIBPATH value" when edited in a text editor, except that the $(BCB) has been replaced with C:\cb6\Borland\CBuilder6.

8. In the Options tab, open the **Other** options and parameters folder. Check the Dynamic link library (-aa) choice. This should match what is found in the customdraw.bpr "LFLAGS value" when edited in a text editor, and compared to the C++BuilderX ILINK32 Effective Command Line options, except that -D&quot;&quot; is not included.

9. Select the **Object files**, click on the "…" button, enter **sysinit.obj** and click **Add**, enter **rtl.bpi** and click **Add**, enter vcl.bpi and click **Add**, enter vclx.bpi and click **Add**. Then click **OK**. This should match the objs found in the customdraw.bpr "ALLOBJ value" when edited in a text editor. The bpi values should match what is found in the customdraw.bpr "PACKAGES value" when edited in a text editor.

10. Select the **BCC32 Tool**.

11. In the **Paths and Defines** tab, uncheck the Merge checkbox for the Include search path (-I), and click the "…" button in the next dialog; click **Remove** to remove the $(BCB)\include. Then enter C:\cb6\Borland\CBuilder6\Include and click **Add**. Then enter C:\cb6\Borland\CBuilder6\Include\vcl and click **Add**. Then click **OK**. This should match what is found in the customdraw.bpr "INCLUDEPATH value" when edited in a text editor, except that the $(BCB) has been replaced with C:\cb6\Borland\CBuilder6.

12. The defaults for the rest of the options in the Effective Command Line should be left as is. They should generally match what is found in the customdraw.bpr "CFLAG1 value" when edited in a text editor.

13. Click the **OK** at the bottom of the Build Options Explorer.

14. In the Project content tab right click on the **CustomDraw.cbx** and select **Add Files…**. Select the .cpp, .h, .res, and .txt files by using the Ctrl key to select more than one, and click **OK**. The only required files are the .cpp and .res files that match what is found in the customdraw.bpr “OBJFILES value” and “RESFILES value” when edited in a text editor.

15. In the **Project** content tab right click on **the CustomDraw.cbx** and select **Make or Rebuild** to compile the project. The warnings are not important, but the result is the customdraw.exe.

16. The project files can now be edited in the C++BuilderX environment by double clicking any of the source files. Classes are viewable by clicking on the **Class browser** tab. After setting break points in one of the C++ files, the application can be debugged by selecting **Run-Debug Project**.

Conclusion

C++BuilderX represents an innovation in C++ technology with its flexibility in adding any compilation toolchain to the environment and its support for modern ALM solutions. Through the mechanisms discussed in this white paper, a development team can continue to leverage existing investments and move forward to take advantage of the key features of this modern IDE; such as multiple platform and compiler support, modern ALM integration, pure C++ framework visual design, and extensions into mobile, effectively taking their C++ development efforts into the future without abandoning the past.