Copyright © 1997. Borland International, Inc.

Using Borland C++Builder Forms in OWL and MFC Applications

Written by Brian Myers and Greg Cole

Contents

I. Introduction

This document describes how to build OWL or MFC programs that invoke forms designed visually in C++Builder using the Visual Component Library. The Visual Component Library (VCL) is the class framework that ships with C++Builder and Delphi. Programs you design visually as forms in C++Builder and Delphi use components built from VCL.

This paper explains how to combine OWL and MFC applications with forms built using C++Builder. It's certainly possible to combine C++ applications with forms built using Delphi, but it is easier with C++Builder because of the common language, C++. For more information on blending C++ with Delphi code, look on *http://www.borland.com/borlandcpp/papers* for "Sharing Code and Objects Between Delphi and C++" and "Application Development with Borland C++ and Delphi."

Pros and Cons

The main reason to consider combining the OWL or MFC framework with the VCL framework is to take advantage of VCL while leveraging all your existing code. For example, the RAD visual design capabilities of Borland C++Builder make it easy to design many self-contained dialog boxes very quickly. You can then invoke those forms directly from your OWL application.

The smallest unit that transfers easily from VCL to another framework is a form. Taking individual controls or components out of VCL individually would be more difficult. And you can get the effect of transferring a single control alone by creating a form that fits the control exactly, invisibly, and bringing that form to OWL.

You can use a VCL form as a dialog or as a client window in your OWL application, and you can put all the controls you want onto a form.

Note: Section V, Late breaking developments, details an alternative integration strategy which is both quicker and easier to implement.

Combining OWL or MFC with VCL involves the following steps:

- Install both C++Builder and Borland C++ 5.0. The Service Release version of Borland C++ 5.0 is required. This release will be available shortly after the release of Borland C++Builder.
- Put BC5\BIN first on your path and define BCBROOT as an environment variable pointing to your C++Builder installation.
- Copy the examples under bc5\examples\cbuilder from your CD to your hard drive.
- Link to a version of the OWL or MFC library that has been rebuilt for VCL compatibility.
  - For OWL, this means either OWL52V.DLL or OWLWV.LIB. Both are available on the Borland C++ 5.0 Service Release CD.
  - For MFC, you'll have to rebuild the MFC library. See the instructions below.

Copyright © 1997. Borland International, Inc.

- Use #define STRICT everywhere. (OWL always does anyway, but VCL does not.)

Also, combining both OWL and VCL results, not surprisingly, in larger .EXE files than would be expected when working with a single class library. A non-debugging version of the simple OWLDLG example linked dynamically and built with the -O1 switch (optimize for size) is about 266K.

If you rely on string tables for editing or internationalizing your user interface, be aware that VCL application UI strings are written in the binary DFM file. VCL forms can be viewed and edited as text and, in addition, a Borland C++Builder Translation Suite will be available after the release of C++Builder based upon the Delphi Translation Suite. This tool supports string extraction, table-based translation, context-sensitive translation for forms and menus, and more.

II. Examples

Borland C++ 5.0 Service Release includes three new examples demonstrating ways to combine VCL forms with OWL and MFC applications. All three examples are located on the CD under \BC5\EXAMPLES\CBUILDER.

EXAMPLES\CBUILDER\OWLDLG: located on your CD

In this example, an OWL application invokes a dialog box that was created with C++Builder. The dialog box is a VCL form, not an OWL TDialog object. You will want to create your own bc5\examples\cbuilder\owldlg directory on your hard drive and copy the example files from your CD to your hard drive.

OWLDLG demonstrates simple interaction between to the OWL and VCL parts. A menu command invokes the dialog. The dialog asks the user to pick a color. When the dialog closes, it passes the selected color back to the application. Each time the user invokes the dialog, the OWL application tells the dialog what the current color setting is, and the dialog starts by showing the current color.

Copy the OWLDLG example from the Borland C++ CD

The Borland C++ 5.0 Service Release ships with new a example that demonstrate how to combine OWL and VCL. The new example is not copied to your hard drive during installation. You'll need to copy it from the CD manually.

- Create a bc5\examples\cbuilder\owldlg directory on your hard drive. It is important to create this in the right place in order for the .ide file copied over to work. From your bc5\examples directory on your hard drive:

  ```
  mkdir cbuilder
  cd cbuilder
  mkdir owldlg
  ```

- From your CD go to the bc5\examples\cbuilder\owldlg directory and issue the command:

  ```
  copy /s <installed-drive>:bc5\examples\cbuilder\owldlg
  ```

EXAMPLES\CBUILDER\HTMLFORM: located on your CD

HTMLForm uses a VCL form to host an ActiveX control in an OWL application. You will want to create your own bc5\examples\cbuilder\htmlform directory on your hard drive and copy the example files from your CD to your hard drive. The VCL form contains a tool bar and the NetManage HTML browser OCX control, available with the Client/Server and Professional versions of C++Builder. The OWL frame has a menu and a status bar. The client window of the OWL frame is the VCL form. The user can type a URL in the VCL tool bar and browse that site. The OCX control generates events when it opens a URL and when it finishes downloading from the URL. The VCL handlers for these events place messages on the OWL status bar. The user can select sites from an OWL menu and OWL calls OCX methods to open the appropriate URL.

Copy the HTMLFORM example from the Borland C++ CD

Copyright © 1997. Borland International, Inc.

The Borland C++ 5.0 Service Release ships with new a example that demonstrate how to combine OWL and VCL. The new example is not copied to your hard drive during installation. You'll need to copy it from the CD manually.

- Create a bc5\examples\cbuilder\htmlform directory on your hard drive. It is important to create this in the right place in order for the .ide file copied over to work. From your bc5\examples directory on your hard drive:

  mkdir cbuilder
  cd cbuilder
  mkdir htmlform

- From your CD go to the bc5\examples\cbuilder\htmlform directory and issue the command:

  copy /s <installed-drive>:bc5\examples\cbuilder\htmlform

EXAMPLES\CBUILDER\MFCDLG: located on your CD

This example resembles OWLDLG in that it invokes a VCL form as a dialog box, but this time from an MFC application. You will want to create your own bc5\examples\cbuilder\mfcdlg directory on your hard drive and copy the example files from your CD to your hard drive. The program in the MFCDLG directory is a modified version of the CTRLTEST program from the standard MFC examples. MFCDLG is described in more detail below, under the heading "Combining MFC and VCL."

Copy the MFCDLG example from the Borland C++ CD

The Borland C++ 5.0 Service Release ships with new a example that demonstrate how to combine MFC and VCL. The new example is not copied to your hard drive during installation. You'll need to copy it from the CD manually.

- Create a bc5\examples\cbuilder\mfcdlg directory on your hard drive. It is important to create this in the right place in order for the .ide file copied over to work. From your bc5\examples directory on your hard drive:

  mkdir cbuilder
  cd cbuilder
  mkdir mfcdlg

- From your CD go to the bc5\examples\cbuilder\mfcdlg directory and issue the command:

  copy /s <installed-drive>:bc5\examples\cbuilder\mfcdlg

III. Combining OWL and VCL

This section gives step-by-step instructions for creating programs like OWLDLG and HTMLFORM. It also explains how to avoid some possible pitfalls. If you want to try the OWLDLG and HTMLFORM examples follow the instructions above and copy them from your CD to your hard drive.

Instructions

To combine OWL and VCL in your programs, follow these steps.

Set up your machine

- Install both Borland C++Builder and Borland C++ 5.0 Service Release on your machine.
- Set the path and environment so that the BC Integrated Development Environment (IDE) and OWL makefiles can locate the source, library, include, and tools directories for both OWL and VCL.
  - Put \BC5\BIN first on your path (before C++Builder, if it is there too.)
  - Define BCBROOT as an environment variable pointing to the directory where C++Builder is installed, for example:

```
SET BCBROOT=C:\CBUILDER
```

Copy the VCL-Compatible OWL Libraries from the Borland C++ CD

Copyright © 1997. Borland International, Inc.

The Borland C++ 5.0 Service Release ships with new libraries that are essential for combining OWL with VCL. The new VCL-compatible libraries are not copied to your hard drive during installation. You'll need to copy them from the CD manually.

- Copy these files into the BIN and LIB directories of your Borland C++ 5.0 tree.

CD directory	Description
\BC5\BIN\OWL52V.DLL	OWL DLL
\BC5\BIN\BDS52V.DLL	BIDS DLL
\BC5\LIB\OWLWVI.LIB	OWL DLL import library
\BC5\LIB\BIDSVI.LIB	BIDS DLL import library
\BC5\LIB\OWLWV.LIB	OWL static-link library
\BC5\LIB\BIDSV.LIB	BIDS static-link library

The CD's BIN and LIB directories also contain VCL-compatible OCF libraries and diagnostic versions of all the VCL-compatible libraries.

Building the VCL-Compatible Libraries Yourself

You can also build the VCL-compatible libraries yourself, if you prefer. The makefiles under \BC5\SOURCE recognize a new -DVCL switch and will pick the right options for you:

```
cd c:\bc5\source\classlib
make BIDSNAME=bids BIDSVER=52 -DVCL
cd c:\bc5\source\ocf
make -DVCL
cd c:\bc5\source\owl
make -DVCL
```

To make the DLL versions, add the -DUSEDLL flag to each command line. To make debugging or diagnostic versions, add the -DDEBUG and -DDIAGS flags. Don't use the -DCODEGUARD flag – CodeGuard support for the Delphi and C++Builder VCL is not yet available.

To use the incremental linker instead of TLINK32, add the -DILINK flag.

Why is a New OWL Library Necessary for Working with VCL?

The "V" versions of the OWL libraries use alignment and vtable compiler options that make them compatible with VCL . They also consider enums to be the size of a byte rather than the size of an int, as does VCL.

VCL uses a new version of the run-time library, the one that ships with Borland C++Builder (CP32MT.LIB). The new RTL to has a different memory manager. To work with VCL, OWL must be linked to the VCL's RTL. Because the VCL's RTL is multithreading, the VCL-compatible version of OWL is really the MODEL=t version of OWL, now built with different command-line options.

Finally, the VCL and OWL frameworks happen to use a lot of the same names for different classes. Both define a TApplication, a TStatusBar, a TRect, etc. When you turn on the -DVCL flag you also turn on a namespace wrapper around OWL. This lets you tell the compiler which version of TRect you mean by writing OWL::TRect or Windows::TRect (where *Windows* is a namespace defined by VCL).

Create a C++Builder Form

- Launch C++Builder and create a new project. Design your form visually and program it completely—adjust properties, add events, and extend your TForm-derived class in any way that will be useful.

Copyright © 1997. Borland International, Inc.

It's C++, just like OWL. You can do whatever you like with the form. You will still be able to edit the form even after building it into an OWL app, but making it as complete as possible the first time helps.

Import the Form Into Your OWL Project

- The .H, .CPP, and .DFM files from your C++Builder project are needed for the OWL project. Copy them to the OWL project directory.
  Command line Users: you can just point your OWL makefile to the C++Builder project files.
  IDE Users: add these files to your project using the project views local menu | Add node.

  - Define STRICT in every unit. (You can omit it if the first Windows header you include is an OWL header. Otherwise, be sure to #define STRICT explicitly.)

- Call AdoptVCLAppWindow in your SetupWindow function. The adoption function is defined for you in ADOPT.H and ADOPT.CPP, part of the VCLDLG example. See ADOPT.H for more details about what it's doing and why.

Build Your Program

You've installed the VCL-compatible OWL libraries, set your environment variables, and designed your C++Builder form. Depending on the logic of your program, you may also have to add a few function calls so the OWL code can interact with the VCL code. They're just C++ function calls. VCL objects can call methods on OWL objects and vice versa without any difficulty.

The next step is to build all the files together.

Directory Options

Command line Users:
Be sure to add this switch for locating the VCL OBJ files:

```
-j$(BCBROOT)\lib\obj
```

IDE Users:
To locate the C++Builder include files, the following must be appended to your projects include path:

```
$env(bcbroot)\include;$env(bcbroot)\include\vcl
```

To locate the VCL OBJ files, the following must be appended to your projects Library path:

```
$env(bcbroot)\lib;$env(bcbroot)\lib\obj
```

Linker Options

If you rebuild your project often, using ILINK32 instead of TLINK32 will speed up your builds noticeably.

Compiler Options

Setting the right options is necessary for a successful compile and link because OWL and VCL normally define vtables differently and use different sizes for enums. Here are the most important option switches to use:

Command line Users:

-a4	align on 4-byte boundaries (OWL defaults to 1)
-b-	don't treat enums as ints (let them be smaller)
-Vx	allow 0-length empty member functions
-Ve	allow 0-length empty base classes

Since -WM (multithreading) is now the default for BCC32 in both Borland C++Builder and Borland C++ 5.0 Service Release, you don't need to set it explicitly.

Copyright © 1997. Borland International, Inc.

IDE Users:

off	Compiler | Code Generation, Allocate enums as ints (OWL defaults to on)
on	32-bit Compiler | Processor, Data alignment - Double word (4-byte)
on	C++ Options | General, Zero-lenth empty base classes
on	C++ Options | General, Zero-length empty class member functions

Even though -WM (multithreading) is now the default for BCC32 in both Borland C++Builder and Borland C++ 5.0 Service Release, this is not true for the IDE. You will need to turn on the Multithread option explicitly using TargetExpert. While in TargetExpert, choose Dynamic if you want to use the OWL DLLs, or Static, to use the static libraries. Note, since TargetExpert resets the libraries, make these changes before manually updating/adding libraries.

Defines

When built for VCL-compatibility, the OWL libraries wrap themselves in three namespaces: ClassLib, OCF, and OWL. Because the namespaces are conditionalized in the header files, when using the OWL namespaces you must always define the BI_NAMESPACE symbol for the compiler.

Command line Users: `-DBI_NAMESPACE`
Command line Users using OWLMAKE.GEN: `-DNAMESPACE`
IDE Users: add `BI_NAMESPACE` to the project's Defines. Also, If you are building with the OWL DLL, confirm these are defined: _RTLDLL;_BIDSDLL;_OWLDLL. If you are building statically, be sure they are not defined.

Libraries

VCL applications require a number of supporting libraries. You'll probably need all of these:

```
owlwv bidsv cp32mt vcl ole2w32 oc30 mpr user32 import32
```

To use the DLL versions of the libraries, replace the first three libraries with owlvi, bidsvi, and cp32mti.

IDE Users: To see what libraries are currently being included by your project, turn on Show run-time nodes (see Options | Environment, Project View). To add the missing BC++ Libraries, use TargetExpert. You will then have to modify the BIDS and OWL libraries to be the correct VCL compatible libraries. Do this by using the project views local menu | Edit node attributes, and changing the name field. Next add the missing C++Builder Libraries explicitly using the project views local menu | Add node.

Using OWLMAKE.GEN

All the OWL examples in Borland C++, including OWLDLG and HTMLFORM, have short makefiles that generate their rules by including a larger and more general makefiles, OWLMAKE.GEN and MAKEFILE.GEN. These have been updated so that they too understand the -DVCL switch. If you're familiar with using MAKEFILE.GEN, then writing OWL/VCL makefiles is easy—just add -DVCL whatever you already have. (Also be sure to set BCBROOT as an environment variable—see above.)

Common Build Errors

Depending on what OWL types you use, you are likely to see compiler warnings such as "ambiguity between OWL::TColor and Graphics::TColor." Because VCL and OWL use the same names for a handful of classes it is sometimes necessary to be completely explicit about which you mean. The VCL-compatible version of the OWL framework uses three namespaces: ClassLib, OCF, and OWL. VCL is split up into many smaller namespaces such the as Graphics, Forms, BDE, Mapi, Controls, and others. Resolve ambiguities by adding an explicit namespace qualifier to each ambiguous class name.

OWL Classes	**VCL Classes**
`ClassLib::TColor`	`Graphics::TColor`

Copyright © 1997. Borland International, Inc.

```
OWL::TApplication      Forms::TApplication
ClassLib::TRect        Windows::TRect
```

Both OWL and VCL have "using" clauses in their headers so you don't need to write *using namespace OWL* or *using namespace Forms* in your own code. They namespaces are always already open for you.

If your code contains forward references to classes defined inside the namespace, then you should wrap those references in the namespace, too. For example, if your code contains this line now

```
class _OWLCLASS TStatusBar;
```

You should change it this:

```
namespace OWL {
class _OWLCLASS TStatusBar;
}
```

Finally, if the linker complains about unresolved references to symbols that include namespace scopes, be sure that you have defined BI_NAMESPACE for the compiler (see Compiler Options, above.)

Debugging Versions

To build debugging versions, link the executable to VCLD.LIB instead of VCL.LIB. (And as always you can rebuild OWLWV.LIB and OWLWVI.LIB with the DEBUG and DIAGS options.)

Potential Difficulties Using OWL with VCL

VCL and OWL use the same identifier value for different resource strings. Furthermore, due to the way VCL apps are built, the VCL strings always get priority. EXEs that combine OWL and VCL always get the VCL string, not the OWL string, when there is a conflict. HTMLForm has to turn off the display of status bar hint text for system menu items because otherwise the wrong strings appear. Other resource ID collisions may still be discovered.

OWL cannot renumber these strings because their numbers are determined by the command ID for the relevant system menu item. One workaround: build the VCL modules of an OWL/VCL app into a separate DLL. All the VCL resources will end up there. OWL will look for its resources in the EXE, VCL in the DLL.

IV. Combining MFC and VCL

This section describes how to modify an MFC example so that it will host a form generated by Borland C++Builder. Specifically, it tells how to add a VCL form to the standard MFC example, CTRLTEST (found in BC5\EXAMPLES\MFC\GENERAL\CTRLTEST.) The instructions detail all the modifications needed to add a dialog box designed as a form in C++Builder. The modified example can be found on your CD in BC5\EXAMPLES\CBUILDER\MFCDLG. If you want to try the MFCDLG example follow the instructions in the EXAMPLES section and copy them from your CD to your hard drive.

CTRLTEST shows an MFC window invoking different types of dialog boxes. MFCDLG adds a command to invoke a C++Builder form as though it were just another dialog box..

These instructions assume that both Borland C++Builder and Borland C++ are installed. It also assumes you have copied the example from the CD to your hard drive. MFC is only included in Borland C++, not C++Builder. VCL is in C++Builder, not in Borland C++.

Instructions

VCL applications are always multithreaded and linked statically to the MFC library (no MFC DLLs). The corresponding MFC library is NAFXCW.LIB.

Copyright © 1997. Borland International, Inc.

Rebuild NAFXCW.LIB

NAFXCW.LIB is normally built with compiler switches that conflict with VCL. To make a compatible version, you'll need to rebuild NAFXCW.LIB with the proper compiler and linker switches. Here are the steps:

- Copy the MFC source code from BC5\SOURCE\MFC directory on the CD to your hard disk. Preserve directory names and structure so that the makefiles will work. For example, copy it to C:\BC5\SOURCE\MFC.

- From the MFC subdirectory enter the command:

```
make -B -l -fborland.mak DEBUG=0 DBGINFO=0 LIBDIR=c:\bc5\lib
"OPT=-a4 -b- -Vx -Ve"
```

This results in a new version of NAFXCW.LIB being created in the C:\BC5\LIB directory

Create the Form in C++Builder

Using Borland C++ Builder, create the desired form. In our modifications to CTRLTEST, we named the form "SimpleCP." C++ Builder creates a .CPP file containing the form's code, a .H file to describe the form, and a DFM file which holds the resource data necessary to create and display the form at runtime.

Add a Menu Choice to CTRLTEST

CTRLTEST already has a menu for selecting different kinds of dialog boxes. We added an item to it.

Add a command ID to RESOURCE.H:

```
#define IDM_TEST_CPPBUILDER             413
```

Add a corresponding menu item to the AFX_IDI_STD_FRAME menu defined in CRTLTEST.RC:

```
MENUITEM "&C++Builder Form...",         IDM_TEST_CPPBUILDER
```

Extend the CTestWindow class defined in CTRLTEST.H by adding a method to handle the menu selection:

```
afx_msg void OnTestCppBuilder();
```

Extend the message map in CTRLTEST.CPP so that selection of the menu at runtime will cause the flow of execution to be passed to the handler:

```
ON_COMMAND(IDM_TEST_CPPBUILDER, OnTestCppBuilder)
```

Create a new file to hold the implementation of the command handler. Ours is BLDRTEST.CPP.

```
#include "stdafx.h"
#include "ctrltest.h"

// C++Builder and MFC both have BEGIN_MESSAGE_MAP and
END_MESSAGE_MAP
// macros.  They are performing analogous tasks but in
// different ways.  The defines made by MFC must be removed prior
to
// including VCL.H to avoid
```

Copyright © 1997. Borland International, Inc.

```
// compiler errors

#undef BEGIN_MESSAGE_MAP
#undef END_MESSAGE_MAP
#include <vcl.h>

#include "SimpleCPPBuilder.h"

void CTestWindow::OnTestCppBuilder()
{
  TForm1 *aForm = new TForm1(NULL);
  aForm->ShowModal();
}

// The purpose of the following code is to call the VCL
// initialization code once during application startup.
//
static void foo(){
  Application->Initialize();
}
#pragma startup foo
```

Reparent the main VCL window

During VCL initialization a VCL TApplication window is created. This is a hidden top level window used to represent the application to the system (its icon shows up in the task bar). In a normal VCL application, this is the desired behavior. However, when the main window is an MFC window the result is *two* top level windows. If you skipped this step, you would end up with two icons in the Task Bar when you ran CTRLTEST.EXE, both referring to the same application.

To avoid the duplication, modify CTRLTEST.CPP (the main module) to #include the file ADOPT.H. Then call AdoptVCLAppWindow to in the CTestApp::InitInstance() method:

```
AdoptVCLAppWindow(*pMainWnd);
```

The code in ADOPT.CPP makes the invisible VCL window a child of the main MFC window so only one icon appears in the Task Bar for CTRLTEST.

Modify the Makefile

Modify Symbol Definitions

The MAKEFILE for CTRLTEST needs to be extended. Add these lines to it:

```
DEBUG=0
CFLAGS=-a4 -b- -Vx -Ve -D_VCL_LEAN_AND_MEAN
LINKFLAGS=-V4.0
CFLAGS=$(CFLAGS) -I"c:\program
files\borland\cbuilder\include\vcl"
OTHERLIBS=vcl
OTHERLIBPATH="c:\program files\borland\cbuilder\lib;c:\program
files\borland\cbuilder\lib\obj"
```

The line "CFLAGS=-a4 -b- -Vx -Ve -D_VCL_LEAN_AND_MEAN" sets the appropriate compiler options and omits portions of the VCL which are not strictly necessary when combining MFC and VCL.

Copyright © 1997. Borland International, Inc.

The line "LINKFLAGS=-V4.0" is not strictly necessary. It will tag the resulting .EXE as being a Windows 4.0 application. Using this tag makes the dialogs in the application look better when run under Win95 or WinNT because they use the CTRL3D extensions.

The line "CFLAGS=$(CFLAGS) -I"c:\program files\borland\cbuilder\include\vcl"" assumes that Borland C++ Builder has been installed to the C:\PROGRAM FILES\BORLAND\CBUILDER directory. The line ensures that header files needed by VCL are locatable.

The line "OTHERLIBS=vcl" specifies that the VCL.LIB file should be linked in to the resulting .EXE. This library resolves the references which will be made to VCL supplied functionality.

The line "OTHERLIBPATH="c:\program files\borland\cbuilder\lib;c:\program files\borland\cbuilder\lib\obj"" will be used to expand the library search path so that VCL.LIB can be located.

Add the New Modules to the Build Process

The MAKEFILE contains the line:

```
OBJS=ctrltest.obj paredit.obj \
```

we need to add our new files to this list so that they will be compiled and linked. Change the line to be:

```
OBJS=SimpleCPPBuilder.obj adopt.obj bldrtest.obj \
  ctrltest.obj paredit.obj \
```

Tell Make What Libraries to Use

The MFCSAMPS.MAK file sets the libraries to be linked with for a statically linked MFC app with the line:

```
LIBRARIES=nafxcw$(DEBUG_SUFFIX).lib $(OTHERLIBS) cw32mt.lib\
import32.lib
```

CP32MT.LIB is a new version of the runtime library (CW32MT was used in past versions) designed to be fully VCL compatible.

```
LIBRARIES=nafxcw$(DEBUG_SUFFIX).lib $(OTHERLIBS) cp32mt.lib\
import32.lib
```

We need to specify a number of other libraries (which we can do by setting OTHERLIBS in our makefile)

Tell Make Where to Look for Libraries

We need to set the library search path to include the C++Builder libraries as well as the standard Borland C++ libraries. MFCSAMPS.MAK sets LINKFLAGS with a line like:

```
LINKFLAGS=/n /m /s /w-inq $(LINKDEBUG) /L$(BORLIB) $(LINKFLAGS)
```

Since BORLIB is used to specify startup code, we can't change it. So we'll extend the line to this:

```
LINKFLAGS=/n /m /s /w-inq $(LINKDEBUG) /L$(BORLIB);$
(OTHERLIBPATH)\
$(LINKFLAGS)
```

Help MAKE Locate Its Include File

The MAKEFILE contains the line

```
!include <$(MAKEDIR)\..\include\mfc\mfcsamps.mak>
```

Copyright © 1997. Borland International, Inc.

MAKEDIR is an internal macro which expands to be the directory where MAKE was invoked. If C++Builder is installed **after** BC5.0 then typing MAKE on the command line invokes the MAKE.EXE from CBUILDER\BIN. That MAKE does not have a ..\INCLUDE\MFC\MFCSAMPS.MAK file.

Either put BC5\BIN first on your path, or use an absolute path in the !include line of your makefile so MAKE can find the MFCSAMPS.MAK installed with Borland C++.

How to Use Debugging Versions of the Libraries

The line "DEBUG=0" makes the linker use the non-debugging version of the MFC library. (To use the debugging version, you could build NAFXCWD.LIB using the command

```
make -B -l -fborland.mak DEBUG=1 DBGINFO=0 LIBDIR=c:\bc5\lib
"OPT=-a4 -b- -Vx -Ve"
```

There is also a debugging version of the VCL. To use it, change OTHERLIBS=vcl to OTHERLIBS=vcld.

V. Late breaking developments - an alternative integration strategy

This section describes an alternate way of combining VCL with OWL or MFC. This approach is simpler, but has not been tested as well as the above method.

This method recognizes the differences in vtable alignment, enum size and structure packing between VCL and MFC/OWL and addresses it by ensuring that the appropriate compiler options are in effect when VCL related classes are encountered. This is done by wrapping references to VCL within header files which explicitly set/reset these options.

The following instructions describe how to modify one of the MFC example programs CTRLTEST (found in BC5\EXAMPLES\MFC\GENERAL\CTRLTEST), so that it will host a VCL based form.

These instructions assume that both Borland C++Builder and Borland C++ are installed. MFC is only included in Borland C++, not C++Builder. VCL is included in C++Builder, not in Borland C++.

Instructions

Copy new VCL include files

Release 1.0 of C++Builder shipped with several header files which require upgrading to resolve compile time conflicts. The new header files have no incompatibilities with the other headers shipped with C++Builder and so there is no need to rebuild any VCL based code that relies on them. This needs to be done only once. The existing files:

SYSDEFS.H
WINDOWS.HPP
CONTROLS.HPP
RICHEDIT.HPP

and the new files:

VCLON.H
VCLOFF.H

in the directory BC5\EXAMPLES\CBUILDER\INCLUDE on your CD need to be copied into the CBUILDER\INCLUDE\VCL subdirectory of your existing CBUILDER installation.

Create C++ Builder form

Use C++Builder to create the desired form - call it myform. Copy the 3 form files (myform.cpp, myform.h, myform.dfm) to the project's directory.

copy myform.* c:\bc5\examples\mfc\general\ctrltest

Copyright © 1997. Borland International, Inc.

Load the project file

Run BCW and open the project file (c:\bc5\examples\mfc\general\ctrltest\ctrltest.ide).

Extend the project's directories

From the Project View window, select the CTRLTEST.EXE node and use the right mouse button to invoke the node's context menu. Select "Edit local options" to see the Project Options Dialog. Select the "Directories" category and append C++Builder's directories to the existing directories. To the Include path add:

```
$env(bcbroot)\include;$env(bcbroot)\include\vcl
```

and to the Library path add:

```
$env(bcbroot)\lib;$env(bcbroot)\lib\obj
```

Add VCL form to project

From the Project View window, select the CTRLTEST.EXE node and use the right mouse button to invoke the node's context menu. Select "Add node", and select the MYFORM.CPP file.

Switch runtime libraries

From the IDE's main menu select Options | Environment | Project View and turn on the "Show run-time nodes" option.

In the Project View window, select the node named "cw32mt.lib". Remove it from the project by pressing the Delete key.

This is an important step - your project will crash if this runtime library is left in the project because VCLON.H brings in an alternate (VCL friendly) runtime library named by CP32MT.LIB.

Incorporate VCL functionality

Make the following modifications to the file CTRLTEST.CPP:

a. add the following after the last #include:

```
#include "vclon.h"
#include "myform.h"
#include "vcloff.h"
```

b. in the function "CTestApp::InitInstance()", after Create has been called on "pMainWnd", add the following code (init VCL stuff correctly):

```
Application->Initialize();
SetParent(Application->Handle, *pMainWnd);
```

c. in the "CTestApp::OnAppAbout()" function (at the bottom of the file), comment out the single line of code and add the following:

```
TForm1 *form = new TForm1(Application);
form->Left = 100;
form->Top = 100;
form->ShowModal();
delete form;
```

Copyright © 1997. Borland International, Inc.

Test the results

Make & run the project; use the "Help | About" menu to invoke the C++Builder dialog box.

Linker issues

VCL.H contains a compiler pragma that causes a library symbol to be emitted into the object file. When this object file is encountered during the link it will cause the file referenced by the pragma to also be included. An advantage of this technique is that the linked libraries can be changed without affecting the project definition. A disadvantage is that control over exactly where (with respect to other object modules) the referenced libraries will be encountered is difficult for the user to determine. The result may be success, warnings or errors during link.

If your link succeeds with no warnings, the order of your linked objects requires no tweaking.

If your link encounters either warnings or errors, it is because the order in which files are being linked is confusing the linker. Borland C++ version 5.02 comes with both an incremental linker (the default) and a standard linker (controlled by the Options|Project|Linker|32-bit linker|Use incremental linker option).

Messages like:

```
Linking D:\BC5\EXAMPLES\MFC\GENERAL\CTRLTEST\ctrltest.exe
  Error: Unable to open file 'GRIDS.OBJ'
  Error: Unable to open file 'OUTLINE.OBJ'
```

Come from the incremental linker and can be fixed by ensuring that a source file containing VCLON/VCLOFF is encountered prior to a file which simply includes VCL.H.

Messages like:

```
Linking D:\BC5\EXAMPLES\MFC\GENERAL\CTRLTEST\ctrltest.exe
  Warning: Export '_DebugHook' is duplicated
  Warning: Export '_ExceptionClass' is duplicated
  Warning: Export '__lockDebuggerData()' is duplicated
  Warning: Export '__unlockDebuggerData()' is duplicated
  Warning: Export '_DebugHook' is duplicated
  Warning: Export '_ExceptionClass' is duplicated
  Warning: Export '__lockDebuggerData()' is duplicated
  Warning: Export '__unlockDebuggerData()' is duplicated
```

Come from the standard linker and can safely be ignored (the duplicates aren't really duplicates at all, they are the exact same code from the same place just referenced differently).

Copyright © 1997. Borland International, Inc.