C++ Builder 4
Client/Server Foundations

Courseware Manual
Chapter 2

Version 1.0

Chapter 2: Projects, Units, and Forms

What will be covered in this chapter:

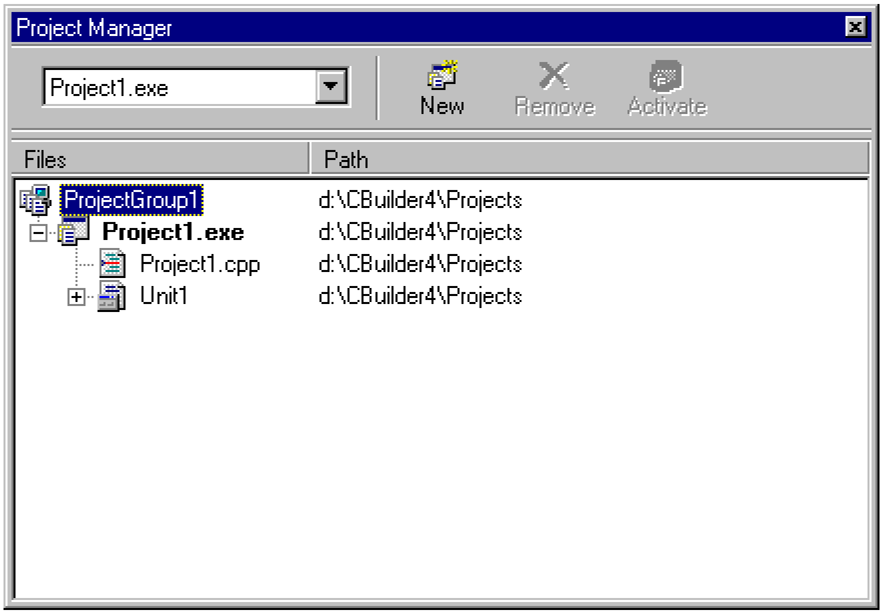
- ❑ What makes up a project in C++ Builder
- ❑ How to invoke and use the Project Manager
- ❑ The basic file types used by C++ Builder
- ❑ How units and forms are related
- ❑ The CPP file
- ❑ The DFM file
- ❑ How to set project options
- ❑ The steps involved in compiling and linking a C++ Builder project

Projects

All of C++ Builder's application development revolves around projects. When you create an application in C++ Builder, you are creating a project. To do any useful work in C++ Builder requires a basic understanding of how projects are constructed. In this chapter, we are going to take a brief look at the file types that make up a C++ Builder project. We will examine these file types in greater detail, later.

Project Manager

C++ Builder has a special tool for project management called, not surprisingly, the Project Manager. To view the Project Manager, select **View | Project Manager.** Here is a picture of the Project Manager displaying default project group, project, unit, and form names:



Files	Path
ProjectGroup1	d:\CBuilder4\Projects
Project1.exe	d:\CBuilder4\Projects
Project1.cpp	d:\CBuilder4\Projects
Unit1	d:\CBuilder4\Projects

Figure 2.1 - The Project Manager

You may have noticed a project group file in the Project Manager. C++ Builder 4 now allows you to open more than one project at a time in the Project Manager. This is extremely useful if you have two (or more) separate but related projects that you would like to work with at the same time. For example, you may have a project that produces a DLL. This DLL project is used by another project that produces an EXE. In the Project Manager, you can create a project group that associates the two projects.

The Project Manager lists, in a hierarchical view, the unit names, the form contained in the unit (if there is a form in the unit—more about this relationship shortly), and the path of the unit (shown to the right of the unit name).

The Project Manager also has some buttons at the top, which are largely self-explanatory. These make it easy to add a project, remove a project, or activate a currently selected project. If you want to add, remove, or save files in a project, simply right-click on the project name in the Project Manager and select the option from the speed menu. Actually, there are several speed menus for the Project Manager that can be displayed by right-clicking on the different files. The options in these speed menus will change based on which type of file you right-click on. Many of these options can also be found under the main **Project** menu.

Project Group Files (*.BPG)

Project Group files are useful for managing related projects in one instance of C++ Builder. However, project group files are not necessary for creating an application in C++ Builder. For example, if you open a project that is not part of a project group, C++ Builder will create a new project group with a default name (ProjectGroup1). You can save the new project group file, but it is not necessary.

If you decide to save the project group file and wish to add a new or existing project to the group, simply right-click on the project group and select the corresponding option. You can also save your project group file from this speed menu.

If you would like to view the project group's source file (.BPG), you can right-click on the project group file name in the Project Manager and choose **View Project Group source.** This will display the file on a page in the code editor. However, C++ Builder automatically maintains your project group file, so you never have to manually edit it. The only place you should manipulate a project group file is through the Project Manager window, as shown in Figure 2.1.

Project Files (*.BPR & *.CPP)

The project is actually maintained in a text file with a BPR extension (which stands for Borland Project). C++ Builder automatically maintains your project file, so you never have to manually edit it. In fact, you should rarely edit the BPR file directly. You should only manipulate a project file through the Project Manager window, as shown in Figure 2.1.

In the general architecture of C++ Builder programs, the project file is the main program file. The BPR file is really a make file at the single project level. The .BPR file stores the compiler and linker settings that C++Builder uses to build your project. When you save a project for the first time, C++Builder prompts you to save a .BPR file that defines the name of the project.

Because C++Builder maintains the .BPR file, you will not normally need to modify it manually, and it is generally recommended that you not do so. Most changes you could make to the .BPR file can be made using the Project|Options in C++Builder, and doing so ensures that C++Builder keeps all the project's files synchronized. Additional build options, however, are available as command line switches.

If you need to manually edit the .BPR file, choose Project|View Project Makefile from the main menu to open it in the Code editor.

C++Builder treats each section of the .BPR file differently. For example, you can modify some parts only from the IDE and some parts you can modify only by editing the .BPR file manually. Additionally, some .BPR file options are intended only for the command line make utility. The following examples show the sections of the .BPR file generated by C++Builder for a new application and how they are used by the IDE.

Here is the text of a BPR file that contains just the default unit and form name for a new project. If you do not understand the syntax shown here, don’t worry! Most of the details are beyond the scope of his course. We are only showing this as a reference point for existing C++ programmers:

```
# ----------------------------------------------------------------
!if !$d(BCB)
BCB = $(MAKEDIR)\..
!endif

# ----------------------------------------------------------------
# IDE SECTION
# ----------------------------------------------------------------
# The following section of the project makefile is managed by the
BCB IDE.
# It is recommended to use the IDE to change any of the values in
this
# section.
# ----------------------------------------------------------------

VERSION = BCB.04.04
# ----------------------------------------------------------------
PROJECT = Project1.exe
OBJFILES = Project1.obj Unit1.obj
RESFILES = Project1.res
RESDEPEN = $(RESFILES) Unit1.dfm
LIBFILES =
IDLGENFILES =
IDLFILES =
LIBRARIES =
SPARELIBS =
PACKAGES = Vcl40.bpi Vclx40.bpi vcljpg40.bpi Vclmid40.bpi
Vcldb40.bpi bcbsmp40.bpi \
  ibsmp40.bpi vcldbx40.bpi Qrpt40.bpi TeeUI40.bpi teedb40.bpi
tee40.bpi Dss40.bpi \
  NMFast40.bpi Inetdb40.bpi Inet40.bpi dclocx40.bpi
DEFFILE =
# ----------------------------------------------------------------
PATHCPP = .;
PATHASM = .;
PATHPAS = .;
PATHRC = .;
DEBUGLIBPATH = $(BCB)\lib\debug
RELEASELIBPATH = $(BCB)\lib\release
USERDEFINES =
SYSDEFINES = _RTLDLL;NO_STRICT;USEPACKAGES
# ----------------------------------------------------------------
CFLAG1 = -I$(BCB)\include;$(BCB)\include\vcl -Od -Hc -
H=$(BCB)\lib\vcl40.csm -w -Ve -r- \
  -a8 -k -y -v -vi- -c -b- -w-par -w-inl -Vx -tW -tWM \
  -D$(SYSDEFINES);$(USERDEFINES)
IDLCFLAGS = -src_suffixcpp -I$(BCB)\include -I$(BCB)\include\vcl
PFLAGS = -U$(BCB)\lib\obj;$(BCB)\lib;$(RELEASELIBPATH) \
  -I$(BCB)\include;$(BCB)\include\vcl -$YD -$W -$O- -v -JPHNE -M
RFLAGS = -i$(BCB)\include;$(BCB)\include\vcl
AFLAGS = /i$(BCB)\include /i$(BCB)\include\vcl /mx /w2 /zd
LFLAGS = -L$(BCB)\lib\obj;$(BCB)\lib;$(RELEASELIBPATH) -aa -Tpe -
x -Gn -v
# ----------------------------------------------------------------
ALLOBJ = c0w32.obj Memmgr.Lib $(PACKAGES) sysinit.obj $(OBJFILES)
ALLRES = $(RESFILES)
ALLLIB = $(LIBFILES) $(LIBRARIES) import32.lib cp32mti.lib
```

```
# ---------------------------------------------------------------------------
!ifdef IDEOPTIONS

[Version Info]
IncludeVerInfo=0
AutoIncBuild=0
MajorVer=1
MinorVer=0
Release=0
Build=0
Debug=0
PreRelease=0
Special=0
Private=0
DLL=0
Locale=1033
CodePage=1252

[Version Info Keys]
CompanyName=
FileDescription=
FileVersion=1.0.0.0
InternalName=
LegalCopyright=
LegalTrademarks=
OriginalFilename=
ProductName=
ProductVersion=1.0.0.0
Comments=

[Debugging]
DebugSourceDirs=$(BCB)\source\vcl

[Parameters]
RunParams=
HostApplication=
RemoteHost=
RemotePath=
RemoteDebug=0

[Compiler]
InMemoryExe=0
ShowInfoMsgs=0

[CORBA]
AddServerUnit=1
AddClientUnit=1
PrecompiledHeaders=1

!endif

# ---------------------------------------------------------------------------
# MAKE SECTION
# ---------------------------------------------------------------------------
# This section of the project file is not used by the BCB IDE.
It is for
```

```
# the benefit of building from the command-line using the MAKE
utility.
# ---------------------------------------------------------------

.autodepend
# ---------------------------------------------------------------
!if !$d(BCC32)
BCC32 = bcc32
!endif

!if !$d(CPP32)
CPP32 = cpp32
!endif

!if !$d(DCC32)
DCC32 = dcc32
!endif

!if !$d(TASM32)
TASM32 = tasm32
!endif

!if !$d(LINKER)
LINKER = ilink32
!endif

!if !$d(BRCC32)
BRCC32 = brcc32
!endif

!if !$d(IDL2CPP)
IDL2CPP = idl2cpp
!endif

# ---------------------------------------------------------------
!if $d(PATHCPP)
.PATH.CPP = $(PATHCPP)
.PATH.C   = $(PATHCPP)
!endif

!if $d(PATHPAS)
.PATH.PAS = $(PATHPAS)
!endif

!if $d(PATHASM)
.PATH.ASM = $(PATHASM)
!endif

!if $d(PATHRC)
.PATH.RC  = $(PATHRC)
!endif
# ---------------------------------------------------------------
$(PROJECT): $(IDLGENFILES) $(OBJFILES) $(RESDEPEN) $(DEFFILE)
    $(BCB)\BIN\$(LINKER) @&&!
    $(LFLAGS) +
    $(ALLOBJ), +
    $(PROJECT),, +
    $(ALLLIB), +
    $(DEFFILE), +
    $(ALLRES)
!
# ---------------------------------------------------------------------
.pas.hpp:
    $(BCB)\BIN\$(DCC32) $(PFLAGS) {$< }

.pas.obj:
    $(BCB)\BIN\$(DCC32) $(PFLAGS) {$< }

.cpp.obj:
    $(BCB)\BIN\$(BCC32) $(CFLAG1) -n$(@D) {$< }

.c.obj:
    $(BCB)\BIN\$(BCC32) $(CFLAG1) -n$(@D) {$< }

.c.i:
    $(BCB)\BIN\$(CPP32) $(CFLAG1) -n. {$< }

.cpp.i:
    $(BCB)\BIN\$(CPP32) $(CFLAG1) -n. {$< }

.asm.obj:
    $(BCB)\BIN\$(TASM32) $(AFLAGS) $<, $@

.rc.res:
    $(BCB)\BIN\$(BRCC32) $(RFLAGS) -fo$@ $<
# ---------------------------------------------------------------------
```

Additionally there is a CPP file with the same name as the BPR file except for the extension. Here is the corresponding simple project CPP for the default form that comes up with a new application.

```
//---------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("Project1.res");
USEFORM("Unit1.cpp", Form1);
//---------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   try
   {
      Application->Initialize();
      Application->CreateForm(__classid(TForm1), &Form1);
      Application->Run();
   }
   catch (Exception &exception)
   {
      Application->ShowException(&exception);
   }
   return 0;
}
//---------------------------------------------------------------
```

The real guts of a C++ Builder application reside in the unit and form files "included" by the project files(BPR and CPP). Let's take a closer look at them.

Units and Forms

Source files and forms are the basic building blocks of a C++ Builder application. You will see references to unit files, which are borrowed from Delphi. A Delphi unit corresponds to the C++ source and header files together, sometimes including the DFM file (discussed later). Therefore, if you see a reference to a unit in any documentation, it is referring to the header and source file taken together. A unit is simply a collection of header and source files.

CPP files

Notwithstanding the project file, files with a CPP extension are regular source files. Usually, your C++ applications will consist of a number of source files with a single CPP project file as a project wrapper. This “main” CPP project file can be distinguished from source CPP files because it uses the project’s name. For example, if your project is called Project1 and the project contains one source file named Unit1, the project source file would be called Project1.cpp while the source file would be labeled Unit1.cpp.

DFM files

The source code for your forms is kept in a source file (*.cpp and *.h). However, the property values that you set in the Object Inspector are not placed in either the source or header file. Instead, for each form there is an additional file with the same name as the source file, but with a DFM extension.

The DFM file is a binary file, which means it cannot be read in a regular text editor. However, if you want to look at the contents of a DFM file, you can use **File | Open** within C++ Builder to view it, and it will be displayed as a text file. In fact, you can open it in this manner and make changes to it. Alternatively, you can right-click on the form and choose **View As Text** to see and edit the contents of the DFM file. C++ Builder will automatically close the visual Form Designer when you open the form as text. Your changes will be reflected in the form’s properties when you bring it back into the Form Designer.

You never have to manually modify the DFM file! C++ Builder automatically handles all modifications for you. However, if you want to modify it, you can. Be forewarned, however, that if you do something that is syntactically incorrect, C++ Builder may not be able to bring your form back into the Form Designer until you correct that item.

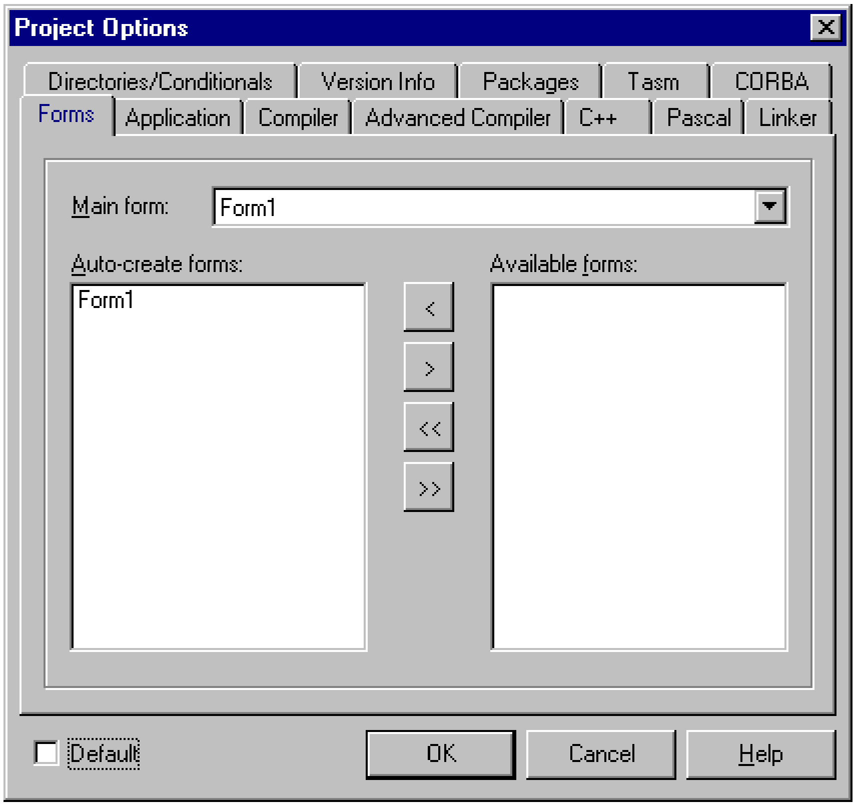
If you want C++ Builder to save the DFM as text, it will (by using the **File | Save As...** command and changing the **Save as Type** setting). In fact, there are some utilities (for version control purposes) that will import the DFM file as text and then send it back as a binary file again. The reason that the DFM file is binary is for speed purposes - binary files can be read much more quickly than text files. Also, the binary DFM file is actually bound into the final executable, so the smaller sized binary file keeps your executable file a smaller size. You can think of the DFM file as a specialized resource file that includes the component property values for the form.

Project Options

From the Project Manager, you can view the project option dialog by right-clicking on the desired project. This mimics the functionality of the **Project | Options...** menu command. This dialog allows you to change various aspects of your project. It is a tabbed dialog with pages for Forms, Application, Compiler, Advanced Compiler, C++, Pascal, Linker, Directories/Conditionals, Version Information, Packages, Tasm, and CORBA. Let's discuss each of these items one at a time.

Forms

The Forms page of Project Options allows you to specify some characteristics of how your application will handle its forms:



Main form: Form1

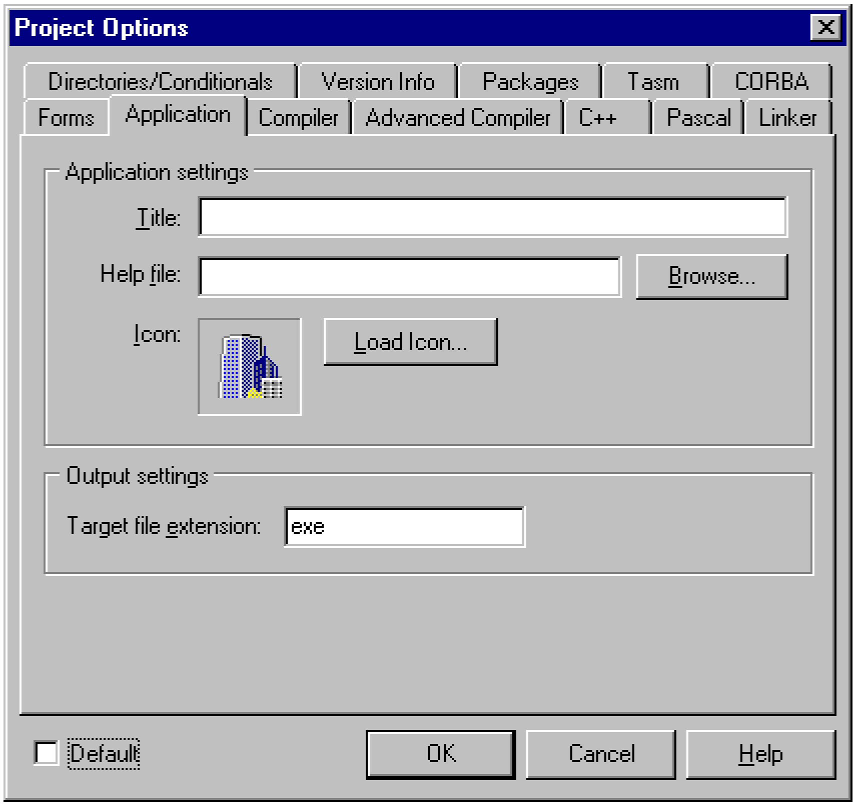
Auto-create forms: Form1

Figure 2.2 - Project Option's Forms Page

This dialog allows you to specify which form will be the main form for the application (in other words, the form that first appears when your application is run). The list box on the right displays the available forms in the project. The list box on the left allows you to specify whether or not to auto-create the forms. If you have the forms in the auto-create list, the project will automatically create them at the time the application loads. This is your best option if you know a particular form will be used the majority of the times the application is run. The other option is to not have C++ Builder auto-create your forms, meaning that you must create them within source code. This is your best option for forms that may not be used every time your application runs. Having a form not auto-create makes your application start faster and use less memory. When you need the form, you can explicitly create it in the application. For example, you might have a setup dialog box that is run very infrequently. It should be a non-auto-create form, because you do not want it instantiated every time you run the application—only when the user needs to change setup information.

Application

The application page of the project options dialog box lets you change characteristics of your application:



Target file extension: exe

☐ Default

Figure 2.3 - Application Page

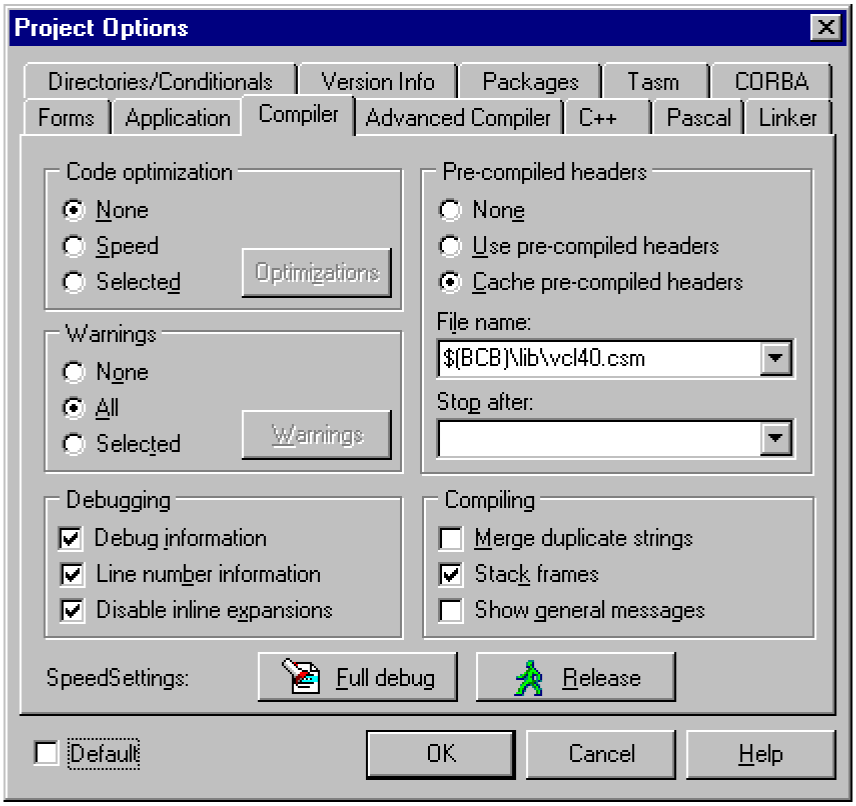
The Title text box lets you give your application a name other than the name of the project file. The Application dialog also allows you to specify what help file will be associated with your application, and allows you to specify an icon for your application. Note that your icon file is stored in the resource file (.RES) until the application has been compiled. If this resource file is deleted, C++ Builder will revert the applications icon back to the default icon. However, once the application has been compiled, the icon is bound in the executable. Such activity is in accord with C++ Builder's ability to create true Windows stand-alone executables.

In the Output Settings section of this page is a text box for a Target File extension. This allows you to set the file extension to be used for the target executable file. For example, if the project is an ActiveX application, the standard file extension could be specified as being .OCX. Typically, you would leave this alone unless you are creating an ActiveX executable instead of an application.

The Application dialog also has a default check box, which will save your current settings as the default settings for projects created in the future. You will see the default check box on all of the Project/Options pages.

Compiler

The compiler page allows you to set compiler options:



Code optimization
● None
○ Speed
○ Selected

Warnings
○ None
● All
○ Selected

Debugging
☑ Debug information
☑ Line number information
☑ Disable inline expansions

Pre-compiled headers
○ None
○ Use pre-compiled headers
● Cache pre-compiled headers

File name: $(BCB)\lib\vcl40.csm

Stop after:

Compiling
☐ Merge duplicate strings
☑ Stack frames
☐ Show general messages

SpeedSettings:

☐ Default

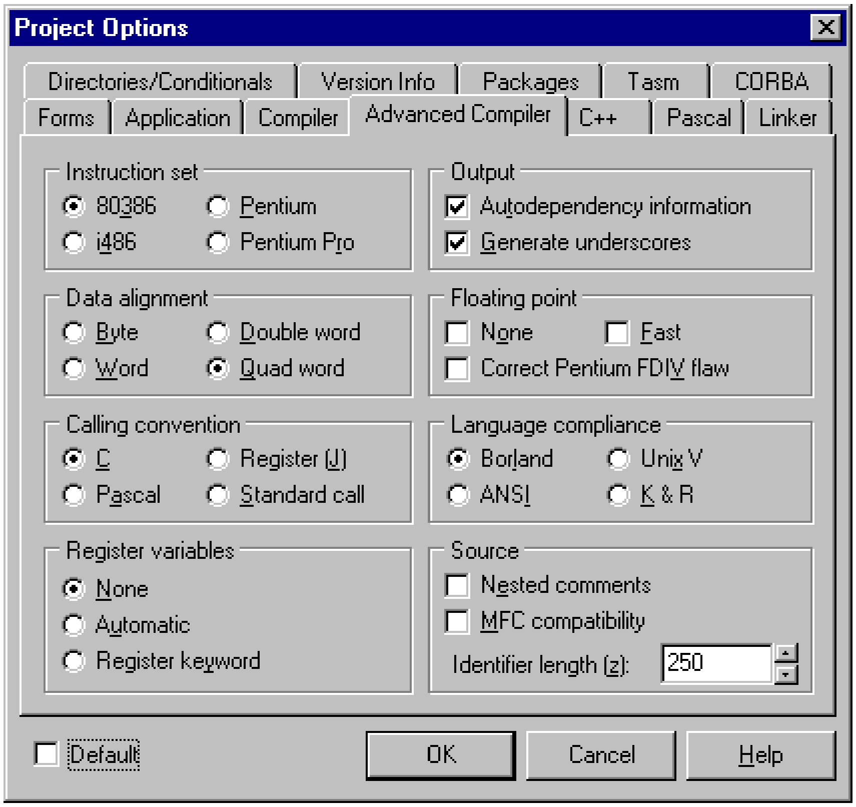
Figure 2.4 - Compiler Page

Most of these options will be discussed later, so we will not delve into them here. Note, however, that this is the place to specify compiler settings for each of your projects. Compiler settings are kept at the project level, as opposed to the environment level. More specifically, you can have projects that have different compiler settings without having to change global settings as you work on each project.

This dialog also allows you to set the compiler's optimizer. If this option is selected, the created EXE file will be compressed and laid out for most efficient loading.

Advanced Compiler

The Advanced Compiler page of the Project Options dialog box specifies additional options for the C++ compiler (modify only with non-VCL applications). The default settings on this page are necessary to build VCL applications.



Instruction set
● 80386 ○ Pentium
○ i486 ○ Pentium Pro

Output
☑ Autodependency information
☑ Generate underscores

Data alignment
○ Byte ○ Double word
○ Word ● Quad word

Floating point
☐ None ☐ Fast
☐ Correct Pentium FDIV flaw

Calling convention
● C ○ Register (J)
○ Pascal ○ Standard call

Language compliance
● Borland ○ Unix V
○ ANSI ○ K & R

Register variables
● None
○ Automatic
○ Register keyword

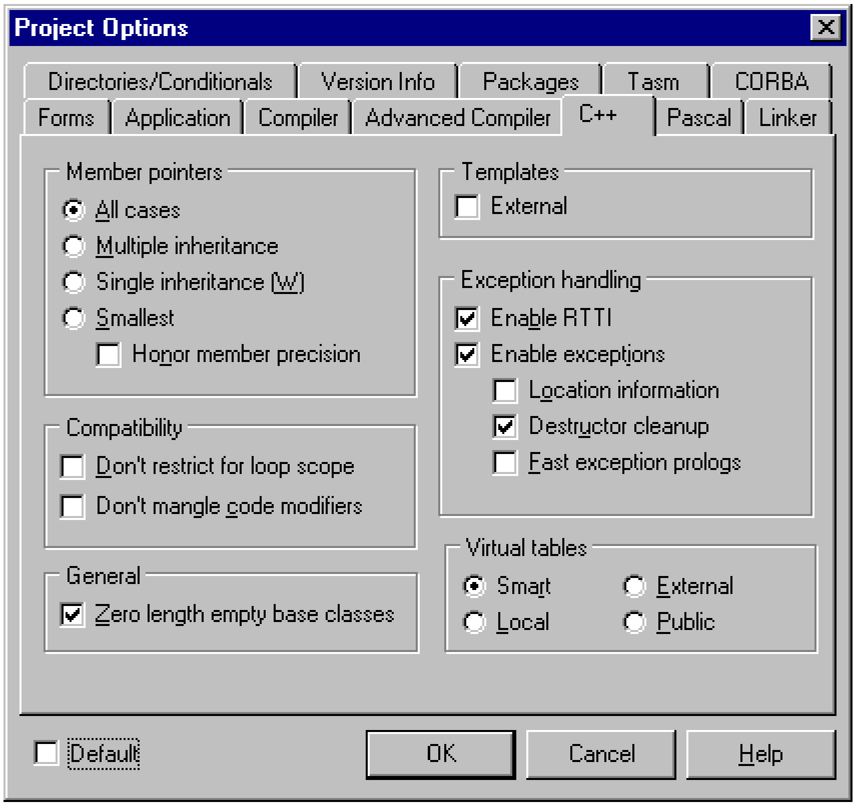
Source
☐ Nested comments
☐ MFC compatibility
Identifier length (z): 250

☐ Default

Figure 2.5 - Compiler Page

C++

The C++ page provides C++ language-specific compiler options:



Member pointers
● All cases
○ Multiple inheritance
○ Single inheritance (W)
○ Smallest
☐ Honor member precision

Compatibility
☐ Don't restrict for loop scope
☐ Don't mangle code modifiers

General
☑ Zero length empty base classes

Templates
☐ External

Exception handling
☑ Enable RTTI
☑ Enable exceptions
☐ Location information
☑ Destructor cleanup
☐ Fast exception prologs

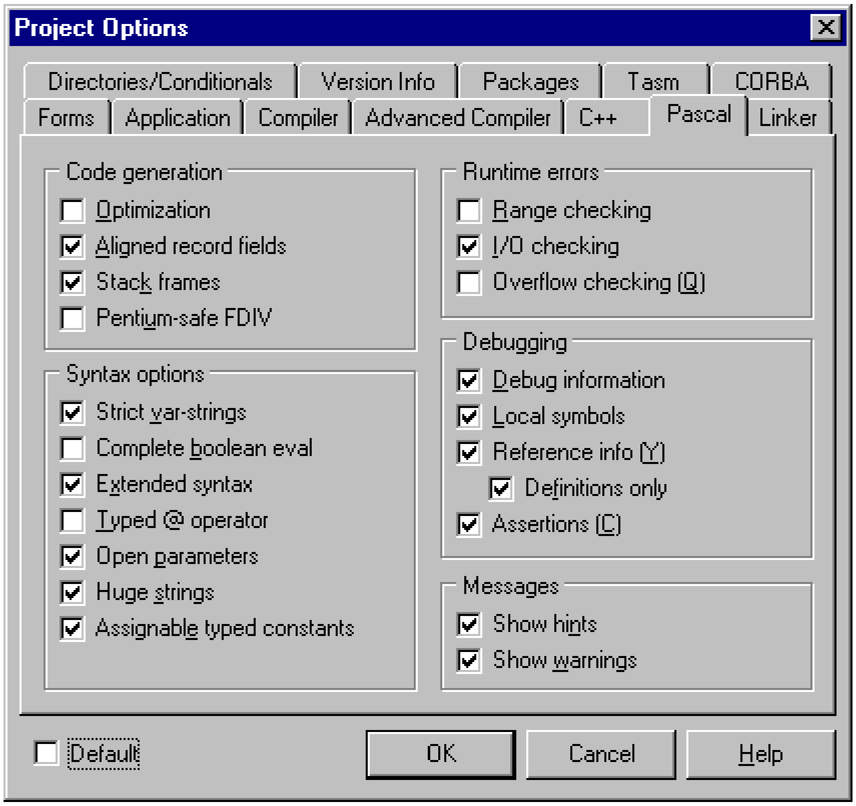
Virtual tables
● Smart
○ External
○ Local
○ Public

☐ Default

Figure 2.6 - Compiler Page

Pascal

This is a screen shot of the Pascal page of the project options dialog:



Code generation

☐ Optimization
☑ Aligned record fields
☑ Stack frames
☐ Pentium-safe FDIV

Syntax options

☑ Strict var-strings
☐ Complete boolean eval
☑ Extended syntax
☐ Typed @ operator
☑ Open parameters
☑ Huge strings
☑ Assignable typed constants

Runtime errors

☐ Range checking
☑ I/O checking
☐ Overflow checking (Q)

Debugging

☑ Debug information
☑ Local symbols
☑ Reference info (Y)
  ☑ Definitions only
☑ Assertions (C)

Messages

☑ Show hints
☑ Show warnings

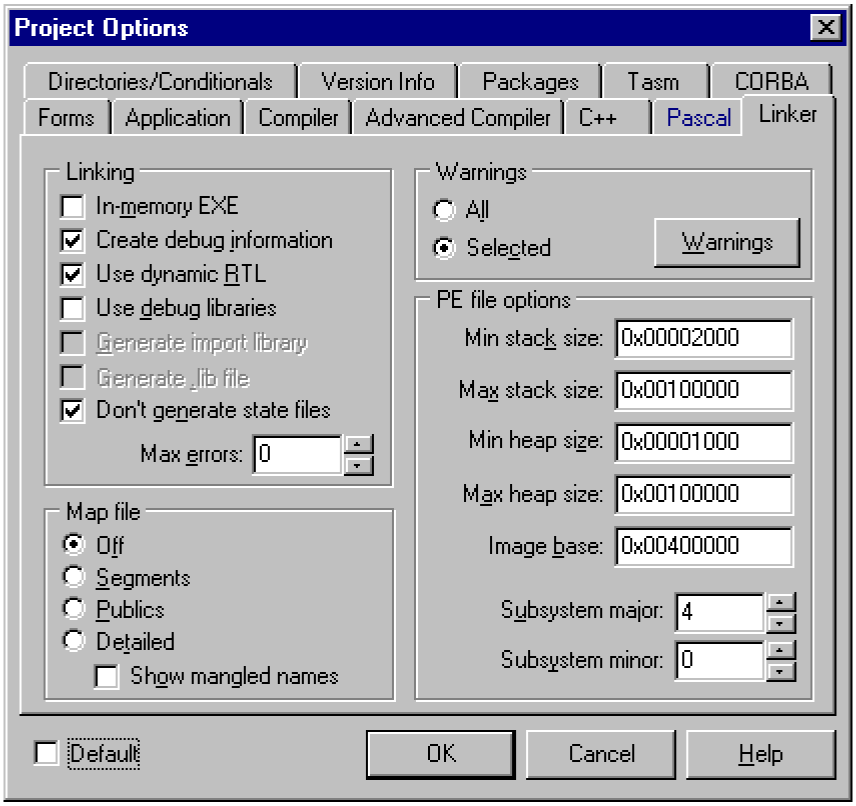
☐ Default

Figure 2.7 - Linker Page

Because C++ Builder uses Object Pascal code for components and some forms, it includes the Delphi Object Pascal compiler in addition to the C++ compiler. Thus, if you have existing Delphi forms and/or units, you can use them in C++ Builder with no modifications. This page of the project options controls compiler settings for Object Pascal.

Linker

The linker page allows you to set various memory and linker options:



Linking
☐ In-memory EXE
☑ Create debug information
☑ Use dynamic RTL
☐ Use debug libraries
☐ Generate import library
☐ Generate .lib file
☑ Don't generate state files
Max errors: 0

Map file
● Off
○ Segments
○ Publics
○ Detailed
☐ Show mangled names

Warnings
○ All
● Selected

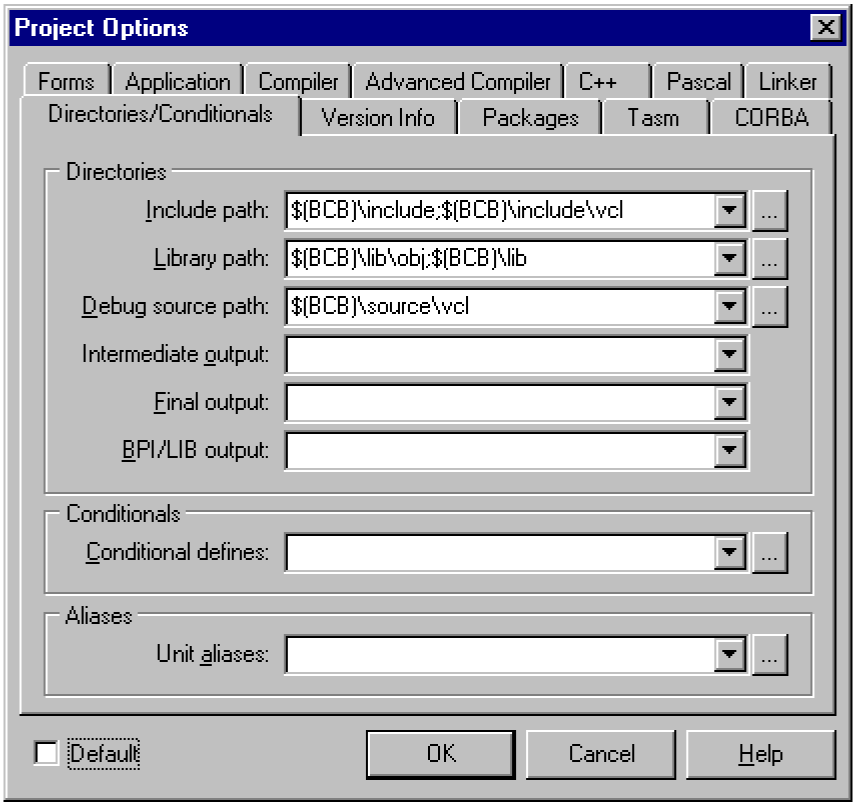
PE file options
Min stack size: 0x00002000
Max stack size: 0x00100000
Min heap size: 0x00001000
Max heap size: 0x00100000
Image base: 0x00400000
Subsystem major: 4
Subsystem minor: 0

☐ Default

Figure 2.8 - Linker Page

Directories/Conditionals

The next page of the Program/Options dialog specifies Directories and Conditionals:



Include path: $(BCB)\include;$(BCB)\include\vcl
Library path: $(BCB)\lib\obj;$(BCB)\lib
Debug source path: $(BCB)\source\vcl

Figure 2.9 - Directories/Conditionals Page

Here, you can specify the output directory for all of the system-generated project files (compiled units and generated executable files). In some cases, you might want to keep your source files and your generated files in separate directories. The Unit Output Directory specifies a separate directory to contain the .OBJ files.

The search path setting allows you to include units in your project that do not reside in the project's home directory. For example, if you have some common library routines that reside in another place on your hard drive, you can indicate the path to them here. You can include multiple search by separating them with semi-colons (";").

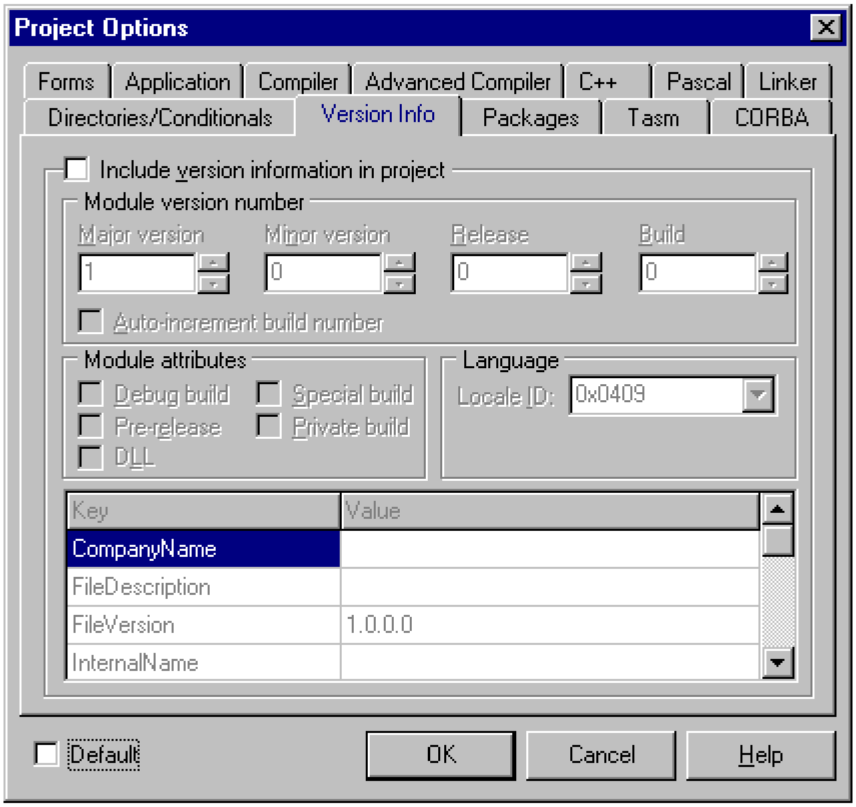
The Debug Source Path setting is used by the Debugger. By default, the Debugger searches the paths defined by the compiler. However, a path can be entered here to include a file in the debugging session.

The BPI and LIB output directory settings allow you to specify an output directory for these files generated after compiling a package. Packages will be discussed in greater detail in a future chapter.

The Conditional Defines setting allows you to list symbols referenced in conditional compiler directives. The Unit Aliases setting is mainly used for backwards compatibility. In this setting, we can specify alias names for units that may have changed names or were merged into a single unit.

Version Information

The Version Information page makes it easy to embed version information in your application.



☐ Include version information in project

Module version number

Major version: 1
Minor version: 0
Release: 0
Build: 0

☐ Auto-increment build number

Module attributes

☐ Debug build
☐ Special build
☐ Pre-release
☐ Private build
☐ DLL

Language

Locale ID: 0x0409

Key	Value
CompanyName	
FileDescription	
FileVersion	1.0.0.0
InternalName	

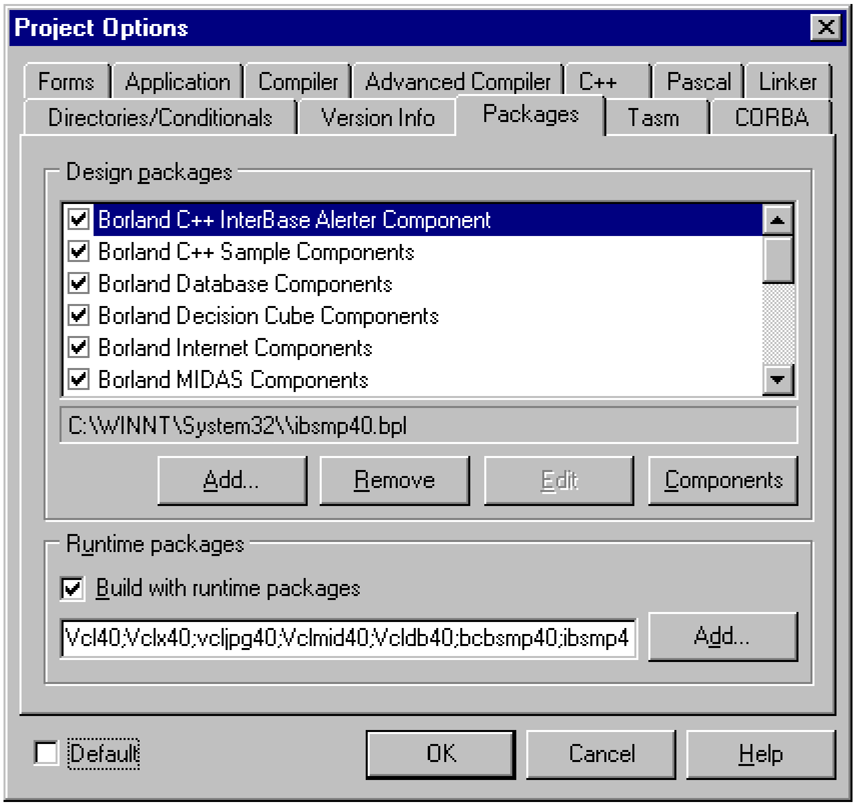
☐ Default

Figure 2.10 – Version Information Page

This information can be accessed in Windows 95 by right-clicking on the EXE and choosing properties.

Packages

The Packages page allows you to change package related options for your project:



☑ Borland C++ InterBase Alerter Component
☑ Borland C++ Sample Components
☑ Borland Database Components
☑ Borland Decision Cube Components
☑ Borland Internet Components
☑ Borland MIDAS Components

C:\WINNT\System32\\ibsmp40.bpl

☑ Build with runtime packages

Vcl40;Vclx40;vcljpg40;Vclmid40;Vcldb40;bcbsmp40;ibsmp4

☐ Default

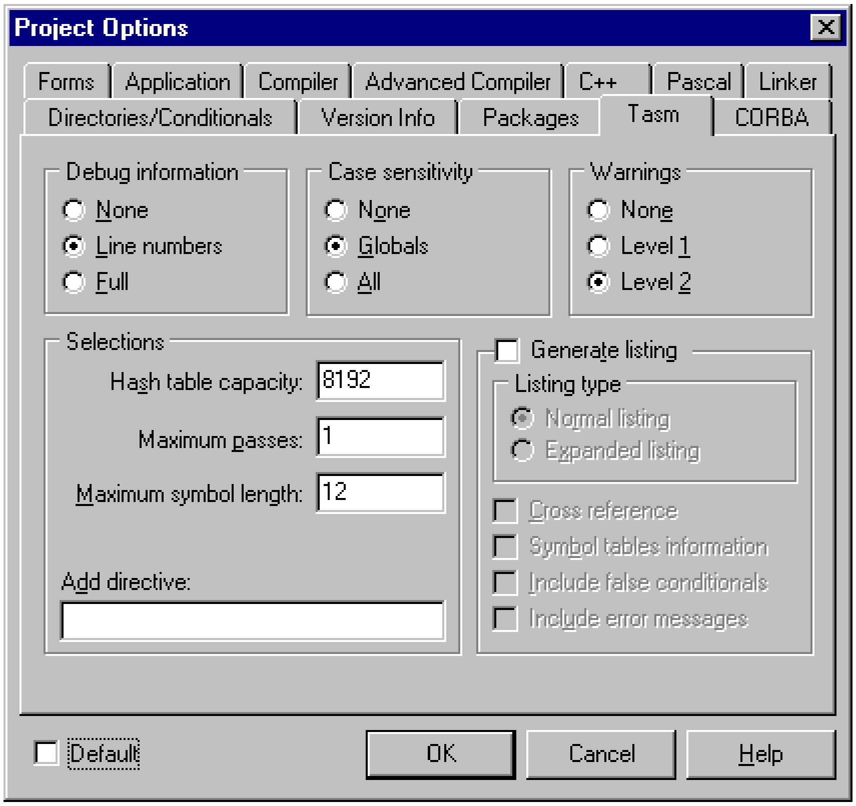
Figure 2.11 – Packages Page

Packages allow you to control your distribution options when you deploy the application. A package enables you to put some common code in DLL's that are installed with your application so that multiple C++ Builder applications can access it. For more information about packages, consult the chapter on Packages found later in this course.

Note: All of the preferences for the Compiler, Linker and Directories/Conditionals pages are stored in the .BPR file. Although changes can be made directly to this file, it is not recommended. If default settings are changed a file, default.bpr, will be created.

Tasm

The Tasm options set the parameters and scope for the turbo assembler.



Debug information
○ None
● Line numbers
○ Full

Case sensitivity
○ None
● Globals
○ All

Warnings
○ None
○ Level 1
● Level 2

Selections
Hash table capacity: 8192
Maximum passes: 1
Maximum symbol length: 12
Add directive:

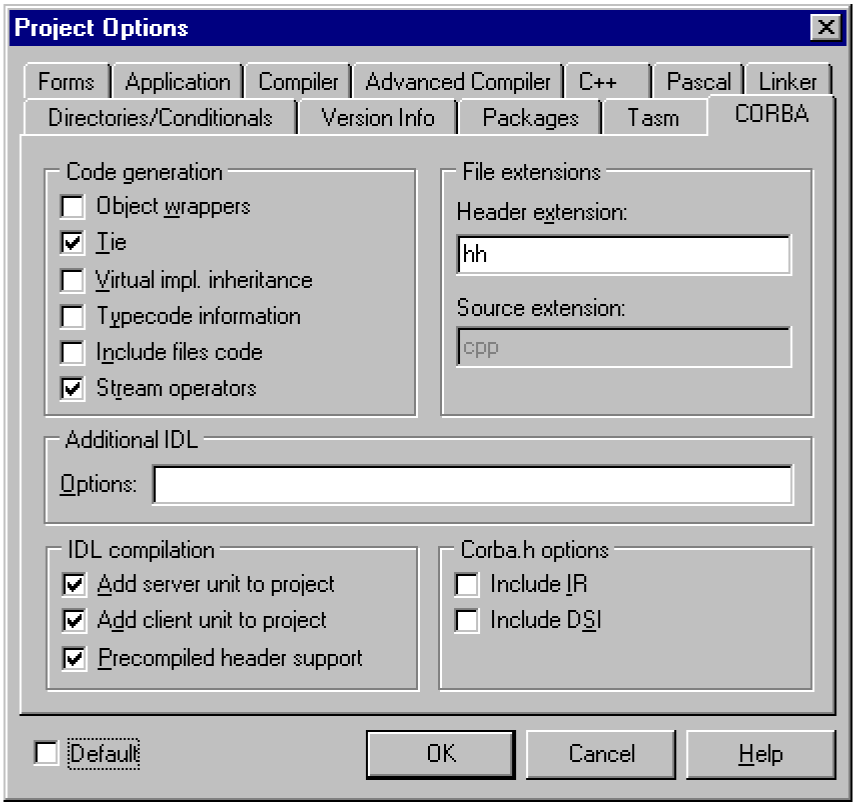
☐ Generate listing
Listing type
● Normal listing
○ Expanded listing
☐ Cross reference
☐ Symbol tables information
☐ Include false conditionals
☐ Include error messages

☐ Default

Figure 2.12 – TASM Page

CORBA

Use the CORBA page to set the flags that control how IDL files are compiled to create server files (that contain skeleton classes), and client files (that contains the stub classes). The CORBA page also allows you to indicate what CORBA library support should be linked into the project.



Code generation
☐ Object wrappers
☑ Tie
☐ Virtual impl. inheritance
☐ Typecode information
☐ Include files code
☑ Stream operators

File extensions
Header extension: hh
Source extension: cpp

Additional IDL
Options:

IDL compilation
☑ Add server unit to project
☑ Add client unit to project
☑ Precompiled header support

Corba.h options
☐ Include IR
☐ Include DSI

☐ Default

Figure 2.13 – CORBA Page

Compiling and Linking

To run and distribute your application, you must compile it. Note that there is no interpreter running in the background, so code must be compiled in to an executable file before it can be run.

Compiling

To compile your application, you can choose the **Project | Make <Project Name>** menu option. You can just choose the **Project | Compile Unit** option to compile the current unit but not build the entire project.

The **Make** command automatically compiles only the modules that need to be compiled. In other words, C++ Builder only compiles the source files that have changed and any that depend on files that have changed. This limited compilation will speed things up, considerably.

There may be some situations when you want all of the modules in a project to be compiled. In such cases, choose the **Project | Build <Project Name>** menu option. This forces the compiler to re-compile every file in the project.

If you are working with multiple projects in a project group, you can compile or build all projects by selecting the **Make All Projects** or **Build All Projects** menu choices found under the **Project** menu.

Linking

Linking in C++ Builder could not be easier, because you never have to do it! The linker is automatically invoked when you tell C++ Builder to compile and create an application. Unlike some other linkers, C++ Builder's linker is so fast that you often don't even notice the link step. Combined with the blazing speed of the compiler, C++ Builder makes it possible to compile and link programs at an unbelievable rate.

Running

Rather than explicitly compiling your application and then running it manually (from the **Run** menu or by pressing **F9**), you will usually use the Compile and Run speed button:

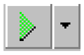


Figure 2.14 - Run Speed button

This button compiles, links, and runs your project. It is a handy shortcut to quickly test your program while it's in development. You can also select which project you wish to run by using the drop down list of projects displayed by selecting the down arrow next to the Run button.

Summary

What was covered in this chapter:

- ✔ We looked at how projects are constructed in C++ Builder, and how to use the Project Manager.
- ✔ We looked at the main file types in C++ Builder: the BPG project group file, the BPR project file, the CPP, the H and the OBJ unit files, and the DFM form file. We also saw how units and forms are related to each other.
- ✔ We looked at how to set project options on each of the pages of the Project Options dialog box.
- ✔ We saw how to compile and link C++ Builder projects. We also saw how the Run button is an easy shortcut to compile, link, and run a C++ Builder application.

CBUILDER 4 FOUNDATION COURSEWARE NO-NONSENSE LICENSE STATEMENT AND DISCLAIMER

IMPORTANT – READ CAREFULLY
This license statement and disclaimer constitutes a legal agreement ("License Agreement") between you (an individual) and Inprise Corporation ("Inprise") for the Cbuilder 4 Foundation Course ("Courseware"), including any software, media, and accompanying on-line or printed documentation. This License Agreement and your rights are only for each of the first twelve chapters of the Courseware that are being provided at no charge under this special program.

BY INSTALLING, COPYING, OR OTHERWISE USING THE COURSEWARE, YOU AGREE TO BE BOUND BY ALL OF THE TERMS AND CONDITIONS OF THE LICENSE AGREEMENT.

Upon your acceptance of the terms and conditions of the License Agreement, Inprise grants you the right to use the Courseware in the manner provided below.

This Courseware is owned by Inprise or its suppliers and is protected by copyright law and international copyright treaty. Therefore, you must treat this Courseware like any other copyrighted material (e.g., a book), except that you may either make one copy of the Courseware solely for backup or archival purposes or transfer the Courseware to a single hard disk provided you keep the original solely for backup or archival purposes.

You may transfer the Courseware and documentation on a permanent basis provided you retain no copies and the recipient agrees to the terms of the License Agreement. Except as provided in the License Agreement, you may not transfer, rent, lease, lend, copy, modify, translate, sublicense, time-share or electronically transmit or receive the Courseware, media or documentation.

WARRANTY DISCLAIMER
THE COURSEWARE ARE PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, BORLAND AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, WITH REGARD TO THE COURSEWARE. SOME STATES/JURISDICTIONS DO NOT ALLOW LIMITATIONS ON DURATION OF AN IMPLIED WARRANTY, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

LIMITATION OF LIABILITY
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL BORLAND OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE COURSEWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF BORLAND HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, BORLAND'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS LICENSE AGREEMENT SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE COURSEWARE PRODUCT OR U.S. $25. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

U.S. GOVERNMENT RESTRICTED RIGHTS

The Courseware and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraphs (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Inprise Corporation., 100 Enterprise Way, Scotts Valley, CA 95066.

GENERAL PROVISIONS
This statement may only be modified in writing signed by you and an authorized officer of Inprise. If any provision of this statement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed for its essential purpose, all limitations of liability and exclusions shall remain in effect.

This statement shall be construed, interpreted and governed by the laws of the State of California, U.S.A. This statement gives you specific legal rights; you may have others which vary from state to state and from country to country. Inprise reserves all rights not specifically granted in this statement.

Form 05/13/00