User's Guide

VERSION
4.0

Borland® C++

User's Guide

Borland® C++

Version 4.0

Redistributable files

You can redistribute the following files in accordance with the No Nonsense License Statement:

- BC40RTL.DLL
- BIDS40.DLL
- BIDS40F.DLL
- BIVBX10.DLL
- BW320007.DLL
- BW320009.DLL
- BW32000C.DLL
- BWCC.DLL
- BWCC0007.DLL
- BWCC0009.DLL
- BWCC000C.DLL
- BWCC32.DLL
- COMPRESS.EXE
- CTL3D.DLL
- CTL3D32.DLL
- CW32.DLL
- CW32MT.DLL
- CX32.DLL
- CX32MT.DLL
- DIB.DRV
- GAUGE.VBX
- LOCALE.BLL
- MARS.DLL
- MARS.MOB
- MSMOUSE.DRV
- OWL200.DLL
- OWL200F.DLL
- PICT.VBX
- REGLOAD.EXE
- STRESS.DLL
- SWITCH.VBX
- TOOLHELP.DLL
- VGAP.DRV
- YESMOUSE.DRV

Borland may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1987, 1993 Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Borland International, Inc.
100 Borland Way, Scotts Valley, CA 95067-3249

PRINTED IN THE UNITED STATES OF AMERICA

1E0R1093
9394959697-9876543
W1

Contents

Tables

Figures

Introduction

See the *Roadmap* for information on the manuals and online Help included with Borland C++.

Borland C++ is a powerful professional tool for creating and maintaining DOS, Windows, Win32s, and Windows NT applications using the C and C++ languages. Part 1 of this manual introduces you to the Integrated Development Environment (IDE) and the command-line tools needed to create applications. Part 2 teaches you how to use Resource Workshop to build Windows resources for your applications.

What's new in Borland C++

Borland C++ 4.0 has many more features than previous releases. The following is a brief list of major additions and changes:

- The 32-bit compiler and tools generate 32-bit targets for Win32s and Windows NT.
- You can generate DOS programs from the Windows IDE.
- The IDE has a graphical integrated debugger for debugging 16-bit Windows applications.
- The IDE has an enhanced editor that lets you record keystroke macros, work in multiple panes in one editor window, and search for text using regular expressions. You can configure the editor to use Brief or Epsilon keystrokes or you can create your own keystrokes.
- The right mouse button brings up *SpeedMenus* that list commands specific to the object you click. For example, some common editing commands are on the SpeedMenu of all editor windows. (To access old functions of the right mouse button, press *Ctrl+click* right mouse button.)
- The IDE has a new multiple-target project manager that visually shows file dependencies and lets you manage more than one program.
- The IDE has a new multiple-window ObjectBrowser that displays class relationships.
- Using AppExpert you can quickly generate ObjectWindows 2.0 Windows programs. ClassExpert helps you modify and organize your AppExpert application.

Manual conventions

This manual uses special fonts and icons as follows:

`Monospaced type`	This font represents text that you type or onscreen text.
Italics	Italics are used to emphasize certain words and indicate variable names (identifiers), C++ function names, class names, and structure names.
Bold	Reserved keywords words, format specifiers, and command-line options appear bold.
Keycap	This font represents a particular key you should press—for example, "Press *Del* to erase the character."
ALL CAPS	All caps are used to represent disk directories, file names, and application names.
Menu | Choice	Rather than use the phrase "choose the Save command from the File menu," this manual uses the convention "choose File | Save".

16 16-bit Windows 32 32-bit Windows

Contacting Borland

Borland offers a variety of services to help you with your questions. Be sure to send in the registration card: registered owners are entitled to receive technical support and information on upgrades and supplementary products. North American customers can register by phone 24 hours a day by calling 1-800-845-0147. Borland provides the following convenient sources of technical information.

Service	How to contact	Available	Cost	Description
TechFax	1-800-822-4269 (voice)	24 hours daily	Free	Sends technical information to your fax machine. You can request up to 3 documents per call. Requires a Touch-Tone phone.
Automated support	408-431-5250 (modem)	24 hours daily	The cost of the phone call	Requires a Touch-Tone phone or modem.

Online Services				
Borland Download BBS	408-431-5096	24 hours daily	The cost of the phone call	Sends sample files, applications, and technical information via your modem. Requires a modem (up to 9600 baud); no special setup required.
CompuServe online service	Type GO BORLAND. Address messages to Sysop or All.	24 hours daily; 1-working-day response time.	Your online charges	Sends answers to technical questions via your modem. Messages are public unless sent by CompuServe's private mail system.
BIX online service	Type JOIN BORLAND. Address messages to Sysop or All.	24 hours daily; 1-working-day response time.	Your online charges	Sends answers to technical questions via your modem. Messages are public unless sent by BIX's private mail system.
GEnie online service	Type BORLAND. Address messages to Sysop or All.	24 hours daily; 1-working-day response time.	Your online charges	Sends answers to technical questions via your modem. Messages are public unless sent by GEnie's private mail system.

For additional details on these and other Borland services, please refer to the *Borland Support and Services Guide* that was included with your product.

P A R T I

Using Borland C++ for Windows

This section of the *User's Guide* describes how to install and use Borland C++. It teaches you how to use the components in the Integrated Development Environment (IDE), including the integrated debugger, the browser, AppExpert, ClassExpert, and the project manager. It also documents the command-lines tools, including the compiler, linker, librarian, and MAKE. Borland C++,

- Integrates development of DOS, Windows, Win32s, and Windows NT applications. You can build more than one application type from a single project file.
- Creates ObjectWindows applications quickly and easily using AppExpert. After you create an application, ClassExpert helps you maintain that application by tracking classes and events and works with Resource Workshop to manage the resources you create and use in your application.
- Helps you debug and browse your applications without having to use a separate debugger.
- Contains a customizable editor. You can use the keyboard shortcuts provided with Borland C++, or you can customize your own.

This section also describes two Windows programs that help you debug your applications: WinSight and WinSpector.

There are two online files for Borland C++ that contain additional material not in the manuals or online Help:

■ INSTALL.TXT	Contains complete installation information for both floppy and CD ROM installations.
■ UTILS.TXT	Describes command-line tools and utilities not found in the manuals or online Help.

C H A P T E R 1

Getting started

Borland C++ is a development package containing Windows tools, command-line tools, and libraries that help you develop applications for DOS, Windows, Win32s, and Windows NT. This chapter gives you a working description of the Borland C++ product—the IDE, project manager, AppExpert, tools, and utilities.

Read this chapter to learn how to

- Install and configure Borland C++
- Use the editor
- Use syntax highlighting for code
- Use the SpeedBar
- Use the Message window
- Browse your code
- Start and use other tools from the IDE

Other utilities and command-line tools are described briefly in this chapter. For information on other parts of the Borland C++ product, see the *Roadmap*, which points you to topics in the documentation.

Installing Borland C++

Borland C++ contains both DOS and Windows applications. Before you can install Borland C++, make sure your computer has the minimum hardware and software requirements.

Hardware and software requirements

To use Borland C++, your computer must have:

- DOS version 4.01 or higher
- Windows 3.1 or higher running in 386-enhanced mode
- A hard disk with 40 MB of available disk space for a normal installation (80MB for a full installation)
- A 1.44 floppy drive or CD ROM (for installation)
- At least 4MB of extended memory
- A Windows-compatible mouse

Although the following items aren't required, they can greatly increase your computer's performance:

- 8MB RAM.
- An 80x87 math coprocessor (if you're writing programs that use floating-point math). Borland C++ emulates a math chip if you don't have one.

Installation steps

The Borland C++ install program installs the Borland C++ product (the IDE, command-line tools, ObjectWindows, and Turbo Debuggers) and it installs Win32s (Win32s lets you run 32-bit programs under 16-bit Windows). The installation program works under Windows, Win32s, and Windows NT; however, not all programs run under Windows NT.

If you use a disk-compression utility, you should read INSTALL.TXT before you install Borland C++.

Before you install, make sure your computer meets or exceeds the hardware and software requirements. If you need more information about installing Borland C++, read the online file INSTALL.TXT, located on Disk 1 (this file isn't compressed, so you can view it using any text editor).

The installation instructions for floppy and CD are basically the same, but you should read the INSTALL.TXT file or the CD liner notes if you're installing from CD ROM.

To install Borland C++ from floppy disks,

1. Put Disk 1 in your floppy drive (usually A or B).
2. Start Windows and choose File | Run from the Program Manager.
3. Type `a:\install` (or `b:\install` if your floppy is in drive B), then press *Enter*. The install dialog box appears. At the bottom of this dialog box you'll see the amount of hard-disk space needed for a complete install (Target Requirements). You'll also see the amount of disk space available on your machine. Make sure you have more than enough room available for installation before continuing. If your computer uses disk compression, read the INSTALL.TXT file; you might need more room than what is listed as available.
4. Click the Customize BC4.0 Installation button if you want to select only specific files for installation. Another dialog box appears with buttons and descriptions for areas of the product. Click a button for an area you want to customize. A dialog box for that area appears where you can uncheck files you don't want installed (the default installation installs all files to your machine). Click OK and repeat this process for any areas you want to customize. Click OK to return to the first install dialog box.

5. The install program lists default directories where it will install files. Type another path only if you want Borland C++ installed to a different directory.

   - **Borland C++ destination directory** is the main directory under which all other files are installed (by default the directory is `C:\BC4`).
   - **Borland C++ Configuration File directory** is where the installation program puts the Borland C++ configuration files (usually `C:\BC4\BIN`).

6. By default, the installation program creates a Windows group where it places all the Borland C++ icons. If you don't want to create a group, uncheck Create Borland C++ Group.
7. Win32s also installs by default. If you don't want Win32s, uncheck this option. Win32s is required for running 32-bit applications.
8. Check LAN Windows Configuration only if you're installing on a machine with LAN Windows.
9. Click Install to begin copying files to your machine. When installation is complete, you can read the README.TXT file. This file contains last-minute changes to the product, the documentation, and the online Help. For a description of all the icons that install creates, read the file INSTALL.TXT located on disk one and C:\BC4.

After you install, make sure your CONFIG.SYS file has `FILES` and `BUFFERS` equal to 40 or more (see your DOS documentation for information on the CONFIG.SYS file).

The install makes the following changes to existing files on your machine:

- AUTOEXEC.BAT now includes the path to Borland C++ (`C:\BC4\BIN` by default).
- WIN.INI includes a section `[BCW4.0 INSTALL]` that is used by the TASM install program to locate where Borland C++ is installed on your machine. Also, in the `[EXTENSIONS]` section, the extension IDE is associated with the IDE (BCW.EXE).
- SYSTEM.INI includes two device lines:
  ```
  device = c:\bc4\bin\tddebug.386
  device = c:\bc4\bin\windpmi.386
  ```
- If you run under Windows NT, `NTCMDPROMPT` is added to CONFIG.NT.

FILELIST.TXT lists every file that ships with Borland C++. If you need to free disk space, read this file before you delete any Borland C++ files.

Starting Borland C++

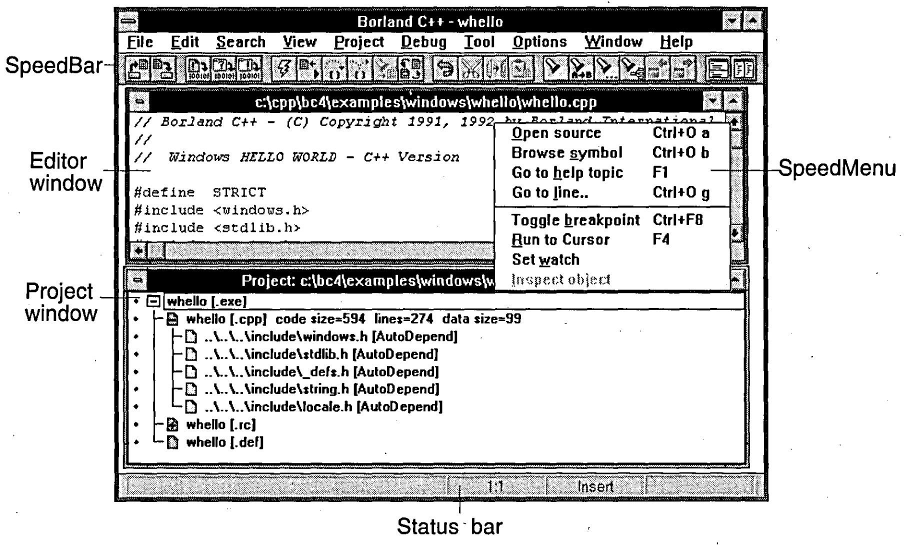
Borland C++

To start the IDE, double-click the Borland C++ icon in Windows (shown at left). The IDE lets you write, edit, compile, link, debug, and manage your programming projects. The IDE has

- An editor and browser described in this chapter
- A project manager, described in Chapter 2, "Using the project manager"
- A debugger, described in Chapter 6, "Using the integrated debugger"

Figure 1.1 shows some of the components of the IDE.

Figure 1.1
Elements of the IDE



```
// Borland C++ - (C) Copyright 1991, 1992
//
// Windows HELLO WORLD - C++ Version

#define  STRICT
#include <windows.h>
#include <stdlib.h>
```

```
whello [.exe]
├ whello [.cpp] code size=594 lines=274 data size=99
│ ├ ..\..\..\include\windows.h [AutoDepend]
│ ├ ..\..\..\include\stdlib.h [AutoDepend]
│ ├ ..\..\..\include\_defs.h [AutoDepend]
│ ├ ..\..\..\include\string.h [AutoDepend]
│ └ ..\..\..\include\locale.h [AutoDepend]
├ whello [.rc]
└ whello [.def]
```

The IDE has context-sensitive SpeedMenus that let you modify objects quickly. To view a SpeedMenu, right-click in a window (the window must be selected first with the left mouse button) or on an item in a window, or press *Alt+F10* (the SpeedMenu changes depending on what is selected). For example, to jump to a line in an editor window, right-click in the editor window, choose Go to line, then type the number for the line you want to view. The menu item "Go to line" appears only when an editor window is selected. If you open a SpeedMenu in a project window, you'll view a completely different set of menu items.

The SpeedBar also changes depending on what window you select. There is a configurable SpeedBar for the editor, browser, debugger, project manager, message window, desktop, and ClassExpert (to configure a SpeedBar, see page 12). When you place the mouse pointer over a button on the SpeedBar, a help line describing the button appears in the status line at the bottom of the IDE.

Some of the buttons on the SpeedBar are dimmed at times. This means that the command the button represents isn't available to you in the current context. For example, if an edit window is open, the Paste Text from Clipboard button is dimmed if there is no text in the Clipboard.

Getting Help

The Borland C++ Help system gives you online access to detailed information about Borland C++. You can find most product information in both the manuals and the online Help. The following topics, however, are only in online Help:

- IDE menu commands
- Editor KEYMAPPER
- Run-time library example code
- Windows API

To get online Help:

- In the IDE, choose Help from the menu or press *F1*.
- In dialog boxes, click the Help button or press *F1*.
- For menu commands, select the menu command and then press *F1*.

Configuring the IDE

You can configure the IDE to do tasks automatically (such as saving a backup of files in editor windows) or to handle events. This section describes what you can configure.

The Options | Environment dialog box lets you configure the editor, browser, debugger, project windows, and other elements of the IDE (these options are saved in a file called BCCONFIG.BCW).

You can also press + or – to expand and collapse the list of options.

To open the Environment Options dialog box, choose Options | Environment. The dialog box appears with a list of topics on the left. Some topics contain subtopics under them. For example, the Editor topic has subtopics called Options, File, and Display. When a topic has subtopics that aren't displayed, the topic contains a + next to the name. When you click a topic's + sign, its subtopics appear under it and the + turns to a – (you can then click the – to collapse the list). Topics that don't contain subtopics appear with a dot next to their name. When you click a topic, its characteristics appear to the right in the dialog box.

Not all Options | Environment topics are discussed in this chapter. See the online Help (click the Help button in the dialog box) for complete reference material on all topics and options.

Some topics associated with tasks or parts of the IDE are discussed elsewhere in this manual (for example, project options are discussed in Chapter 2). Check the index of this manual for entries on specific topics.

Changing the SpeedBars

The IDE has SpeedBars for Editor, Browser, Debugger, Project, Message, Desktop, and ClassExpert windows. When you select one of these types of windows, the corresponding SpeedBar appears. You can customize each of the SpeedBars so that they include only the buttons you need.

To add or delete buttons from any of the SpeedBars,

1. Choose Options | Environment from the main menu.
2. Choose the SpeedBar topic on the left. The right side of the dialog box displays general options for all SpeedBars.

   The options here let you choose how you want the SpeedBar to appear (top or bottom of the IDE), and how you want it to behave (check Use flyby help to view help hints on the status line when you pass the mouse pointer over a button).
3. Choose Customize, the topic under SpeedBar. The options on the right display information about the SpeedBars.
4. Choose the type of SpeedBar you want to modify (Editor, Browser, Debugger, Project, Message, Desktop, or ClassExpert) from the Window drop list.

   The column on the left (Available Buttons) displays all the available (unused) buttons with names next to them that describe the button's function. The column on the right (Active Buttons) displays only the buttons for the selected SpeedBar.
5. To add a button, double-click the button icon in the Available Buttons list, or select it and click the right-pointing arrow. The button moves to the Active Buttons list.
6. To remove a button from a SpeedBar, double-click the button icon in the Active Buttons list, or select it and click the left-pointing arrow. The button moves to the Available Buttons list.

To reorder the button positions for a SpeedBar, use the up and down arrows. The selected button in the Active Buttons list moves up or down the list (the top button appears on the far left of the SpeedBar; the last button in the list appears on the far right).

You can also make all SpeedBars identical by selecting a SpeedBar in the Window list, then pressing the Copy Layout button. A dialog box appears in which you check all the SpeedBars you want to make identical to the selected SpeedBar. For example, if you first choose the Editor SpeedBar, then click Copy Layout, the dialog box appears with Editor dimmed. If you then check Project and Message, those SpeedBars will be exactly the same as the Editor SpeedBar.

You can restore any SpeedBar to its original defaults by selecting the SpeedBar in the Window list then clicking the Restore Layout button.

Separators put space between two buttons. You can add separators to any SpeedBar by selecting a button from the Active Buttons list then clicking the Separator button. The separator is added *before* the selected button.

Setting IDE preferences

Preferences let you customize what you want saved automatically and how you want some windows to work.

To set preferences,

1. Choose Options | Environment | Preferences.
2. Check and uncheck the options you want. See the online Help (press the Help button) for an explanation of each option.
3. Choose OK.

Saving your IDE settings

The IDE automatically saves information when you exit the IDE, build or make a project, use a transfer tool, run the integrated debugger, or close, open, or save a project. You can control the automatic saving by choosing Preferences from the Environment Options dialog box (choose Options | Environment from the main menu) and setting options for automatic save.

To save your settings manually,

1. Choose Options | Save.
2. Check Environment to save the settings from the Editor, Syntax Highlighting, SpeedBar, Browser, and Preferences sections of the Environment Options dialog box. These settings are saved in a file called BCCONFIG.BCW.
3. Check Desktop to save information about open windows and their positions. This information is saved to a file called *<prjname>*.DSW. If you don't have a project open, the information is saved to a file called BCWDEF.DSW.
4. Check Project to save changes to your project (.IDE) file, including build options and node attributes.

Using the Editor

You can open up to 32 editor windows in the IDE.

Editor windows are where you create and edit your program code. When you're editing a file, the IDE status bar displays the line number and character position of the cursor. For example, if the cursor is on the first line and first character of an editor window, you'll see `1:1` in the status bar; if the cursor is on line 68 and character 23, you'll see `68:23`. The IDE status bar also indicates whether the cursor will overwrite or insert characters (press *Insert* to toggle this option) and displays the word `Modified` if you've made any changes to the file in the selected edit window.

The editor lets you undo multiple edits by choosing Edit | Undo or pressing *Alt+Backspace*. For example, if you delete a line of text, then paste some text, you can undo these edits: the pasting, which was the last edit, is undone first, then the deletion. You can set the number of undo actions allowed by choosing Options | Environment | Editor | Options and setting the Undo Limit.

Configuring the IDE editor

You can configure the editor so that it looks and behaves similar to other editors (like Brief and Epsilon). The IDE editor uses keyboard mapping files (.CKB) that set the keyboard shortcuts for the editor (these files also change the keystrokes for other windows).

The .CKB files also configure other windows in the IDE. See the online Help for more information.

You can use one of the four default .CKB files by choosing Options | Environment | Editor and clicking a SpeedSetting (Default keymapping, IDE classic, BRIEF emulation, or Epsilon). To learn how to edit or create your own .CKB file, see the online Help (search on "Keymapper").

Syntax highlighting

Syntax Highlighting lets you define a color and font attribute (like bold) for certain elements of code. For example, you could display comments in blue and strings in red. Syntax Highlighting is on by default. To turn off highlighting,

1. Choose Options | Environment | Syntax Highlighting.
2. Uncheck Use Syntax Highlighting.

Syntax Highlighting works on files whose extension is listed in the Syntax Extensions list (.CPP, .C, .H, and .HPP by default). You can add or delete any extension from this list, but you must separate extensions with semicolons.

The Syntax Highlighting section displays the default color scheme and four predefined color settings (buttons) you can use.

To use a predefined color scheme,

1. Choose Options | Environment | Syntax Highlighting.
2. Choose one of the four color schemes by clicking its button; the sample code changes to use the color scheme you select. You can use a color scheme as a starting point for customizing syntax highlighting.

To manually select syntax highlighting colors,

1. Choose Options | Environment | Syntax Highlighting | Customize. Elements and sample code appear on the top right of the Environment Options dialog box.
2. Select an element you want to modify from the list of elements (for example, Comment), or click the element in the sample code (this selects the name in the Element list). You might need to scroll the sample code to view more elements. The sample code uses the font selected in the Editor | Display section of the Environment Options dialog box.
3. Select a color for the element. The element color in the sample code reflects your selection. Use the left mouse button to select a foreground color for the element (FG appears in the color). Use the right mouse button to select a background color (BG appears in the color). If FB appears in the color, the color is used as both a background and a foreground color.
4. Choose an Attribute such as bold, if you want.
5. You can check Default FG (foreground) or BG (background) to use the Windows default colors for an element.
6. Repeat steps 2-4 for the elements you want to modify.

Working with the Message window

You can customize some of the functionality of message windows by using Preferences in the Environment Options dialog box.

The Message window displays errors and warnings when you compile programs. When you select a message in the Message window, the editor places the cursor at the spot in your code where the error or warning occurred. If the file containing the error isn't loaded in an editor window, press *Spacebar* to load it (you can also press *Alt+F10* and choose View source from the SpeedMenu). The message window remains selected so you can move from message to message.

To view the code associated with an error or a warning, either select the message in the message window and press *Enter*, double-click the message, or press *Alt+F10* and choose Edit source from the SpeedMenu. The cursor appears on the line and column in your source code where the error is most likely to have occurred (the message window moves to the background). Use *Alt+F7* to move to the next error message or *Alt+F8* to go to the previous error message.

You can also cursor through messages in the message window. As you select a message, the cursor in the editor window moves to the place where the error occurred (this is called automatic error tracking). Automatic error tracking works only if the file containing the errors is displayed in an editor window. If the next message you select references another source file (not the one in the current editor window), you must select the editor window that displays the source file associated with the message before you can continue automatic error tracking.

You can clear the message window by choosing Remove all messages from the message-window SpeedMenu (right-click or press *Alt+F10* to view the SpeedMenu).

Browsing through your code

The browser lets you view the object hierarchies, classes, functions, variables, types, and constants your program uses. The browser also lets you

- Graphically view the hierarchies in your application, then select the object of your choice and view the functions and symbols it contains and inherits.
- List the variables your program defines, then select one and view its declaration, list all references to it in your program, or edit its declaration in your source code.

Before you use the browser, be sure to set these options in the Project Options dialog box (choose Options | Project) and compile your application:

- Choose Compiler | Debugging and check Debug information in OBJs
- Choose Compiler | Debugging and check Browser reference information in OBJs
- Choose Linker | General and check Include debug information.

The Browser has a customizable SpeedBar (see page 12 for more information on customizing SpeedBars).

To start the browser, choose Search | Browse Symbol, View | Classes, or View | Globals. You can also place your cursor on a symbol in your code and choose Search | Browse symbol to bring up the browser. If the program in the current editor window hasn't been compiled yet, you must compile and link your program with debugging information before you can use the browser. If you try to browse a class definition (or any symbol that doesn't have symbolic debug information), you'll get an error message.

You can set several browser options using the Environment Options dialog box. Choose Options | Environment, click the Browser topic and select the options you want to use. Single window means you can only have only one browser window up at a time; Multiple windows opens a new browser window each time you perform a browsing action (such as choosing View | Globals from the main menu). Visible symbols are described on page 18.

Browsing through objects (class overview)

Choose View | Classes to see the "big picture," the object hierarchies in your application, as well as the small details. When you choose View | Classes, the browser draws your objects and shows their ancestor-descendant relationships in a horizontal tree. The red lines in the hierarchy help you see the immediate ancestor-descendant relationships of the currently selected object more clearly. Figure 1.2 shows the structure of the WHELLO application.

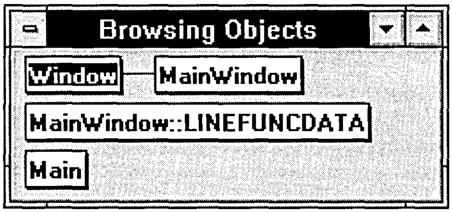


Figure 1.2
Viewing classes in an application

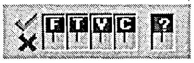
To see more information on a particular object, double-click it. If you aren't using a mouse, select the object by using your arrow keys and press *Enter.*

Filters

When you browse a particular symbol, the same letters that identify the symbol appear in a Filters matrix at the bottom of the browser window. You can use filters to select the type of symbols you want to see listed. (See Table 1.1 for a list of letters and their meaning.)

You can also check the Browser options in the Environment Options dialog box to select the type of symbols, but you must set these options before opening browser windows.

The Filters matrix has a column for each letter. Click the top or bottom row to move the letter (a letter in the top row means the browser shows symbols with that identification; a letter on the bottom means the browser excludes symbols with that identification).



To restrict views of a particular type of symbol, click the bottom cell of the letter's column as shown at left. For example, to remove all the variables displayed in the currently selected object, click the bottom cell in the **V** column.

In some cases more than one letter appears next to a symbol. The second letter appears just after the letter identifying the type of symbol and further describes the symbol. See Table 1.1 for a list of filter identifiers.

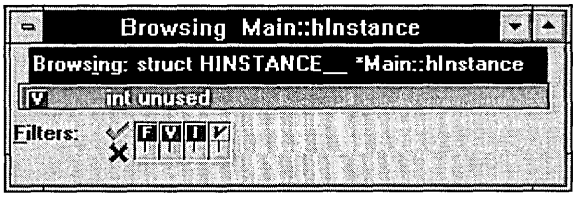
Viewing declarations of listed symbols

Use one of these methods to see the declaration of a particular symbol displayed in a list:

- Double-click the symbol.
- Select the symbol and press *Enter*.
- Select the symbol, press *Alt+F10* to view the SpeedMenu, then choose Browse Symbol.

The symbol declaration appears in a window, as shown in Figure 1.3.

Figure 1.3 Symbol declaration window



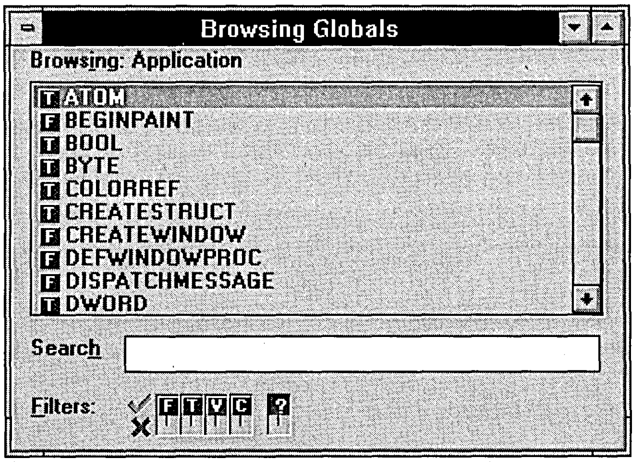
Browsing: struct HINSTANCE__ *Main::hInstance

V int unused

Browsing through global symbols

Choose View | Globals to open a window that lists every global symbol in your application in alphabetical order. The browser lists the symbols (the functions, variables, and so on) used in the object. Figure 1.4 shows the globals for the WHELLO program.

Figure 1.4 Viewing globals



Browsing: Application

T ATOM
F BEGINPAINT
T BOOL
T BYTE
T COLORREF
T CREATESTRUCT
F CREATEWINDOW
F DEFWINDOWPROC
F DISPATCHMESSAGE
T DWORD

One or more letters appear to the left of each symbol in the object. The letters describe what kind of symbol it is. You can filter out symbols using the filter list at the bottom of the browser window. See the previous section "Filters" for more information.

Table 1.1
Letter symbols in the Browser

Letter	Symbol
F	Functions
T	Types
V	Variables
C	Integral constants
?	Debuggable
I	Inherited from an ancestor
v	Virtual method

You can also type regular expressions for searching (for example, you can use ?, * and +).

To get more information on a particular symbol, either click the symbol or use your cursor keys to select it. A Search input box at the bottom of the window lets you quickly search through the list of global symbols by typing the first few letters of the symbol's name. As you type, the highlight bar in the list box moves to a symbol that matches the typed characters. You can view the symbol declaration by selecting the symbol and pressing *Enter*. See the previous section, "Viewing declarations of listed symbols," for more information.

Using regular expressions in the browser

You can use expressions in the search box in some browser windows. See Table 1.2 for a list of the symbols allowed.

Table 1.2
Browser search expressions

Character	Function
.	Matches one of any character.
*	Matches zero or more of the previous character. For example, * is an error because there is no previous character `fo*` matches anything starting with an "f" `fo*x` matches "fx", "fox","fooox"
+	Matches one or more of the previous character. For example, + is an error `fo+` matches anything starting with "fo" `fo+x` matches "fox", "fooox"
?	Matches zero or one of the previous character. For example, ? is an error `fo?` matches anything starting with "f" `fo?x` matches only "fx" or "fox"

Browsing symbols in your code

You can also browse any symbol in your code without viewing object hierarchies or lists of symbols first. Choose from these methods:

- Highlight the symbol in your code and choose Search | Browse Symbol.
- Click the right mouse button or press *Alt+F10* when an editor window is selected to display the SpeedMenu, then choose Browse Symbol.

Using command-line tools

Borland C++ contains several command-line tools that let you do the same tasks you can do in the IDE. Borland C++ includes a command-line compiler, a linker, a resource compiler, a librarian, a project builder (called MAKE), and other tools. Most of these tools are documented in this manual. Some are documented in online files. All tools are documented in the online Help.

You can use either the IDE or the command-line tools, because they produce the same results, but you might choose to use the command-line tools if you program using a DOS editor such as Brief. Here's a list of the command-line tools, what they do, and where they are documented:

- BCC.EXE and BCC32.EXE are the 16-bit and 32-bit compilers. They are documented in Chapter 3.
- TLINK.EXE and TLINK32.EXE link .OBJ files and .LIB files to form .EXEs and .DLLs. They are documented in Chapter 9.
- IMPLIB.EXE, and TLIB.EXE help you work with and create libraries. They are described in Chapter 11.
- HC31.EXE compiles files for online Help and creates the .HLP file that most Windows applications can use. It is documented in the online Help.
- BRCC.EXE, BRCC32.EXE, BRC.EXE, BRC32.EXE, and RLINK.EXE are resource tools that compile resources for your applications. They are described in Chapter 10.
- MAKE.EXE and MAKER.EXE help manage your projects by building only the files that have changed since the last build. They are documented in Chapter 12.

DPMI and the command-line tools

The command-line compiler uses DPMI (Dos Protected Mode Interface) to run in protected mode on 286, 386, i486, or Pentium machines with at least 640K conventional RAM and at least 1MB extended memory.

Although Borland C++ runs in protected mode, it still generates applications that run in real mode. The advantage to using Borland C++ in protected mode is that the compiler has *much* more room to run than if you were running it in real mode, so it can compile larger projects faster and without extensive disk-swapping.

Memory and MAKESWAP.EXE

If you get "Out of Memory" errors from DOS (not running DOS from Windows) when running the 32-bit command-line tools, create a swap file with the MAKESWAP utility. MAKESWAP takes the size of the file to create in KBytes, for example:

```
MAKESWAP 12000
```

MAKESWAP applies to DOS only, not to DOS boxes opened under Windows. See the online file INSTALL.TXT for information on running the tools from DOS boxes.

creates a 12MB swap file called EDPMI.SWP in the current directory, which the command-line tools use when they need additional memory. To set up a swap file, use the DPMIMEM environment variable at the DOS prompt or add this line to your AUTOEXEC.BAT file:

```
set DPMIMEM=SWAPFILE <location of swap file>\EDPMI.SWP
```

You must clear this environment variable before running Borland C++ 3.1 command-line tools or other 16-bit DPMI-hosted executables, such as Paradox. To clear the variable, type at the DOS prompt:

```
set DPMIMEM=
```

The run-time manager and tools

Borland C++ protected-mode applications (such as BCC and BCC32) use the run-time managers RTM.EXE and 32RTM.EXE. The tools that use the run-time manager first load the run-time manager, then do their work, and then unload the run-time manager. If you're doing lots of calls to 32-bit command-line tools that use the run-time manager (perhaps from a makefile), you could speed up the process by loading the run-time manager once, then calling the tools. To load the run-time manger, type `32RTM` at the command line. To unload the run-time manager, type `32RTM -u`.

Controlling the memory RTM uses

By default, the run-time manager consumes all available memory for itself when it loads. It then allocates memory to its clients when they request it through the memory manager API routines.

To control how much memory the run-time manager can use, at the DOS command line add the RTM environment variable to your system's DOS environment. Here is the syntax:

```
SET RTM=[option nnnn]
```

The following table lists the options you can use, where nnnn can be a decimal number or a hex number in the form of xAB54 or xab54.

Table 1.3 Environment variables for RTM's memory allocation

Option	Description
EXTLEAVE nnnn	Always leave at least nnnn kilobytes of extended memory available. The default value is 640K.
EXTMAX nnnn	Don't allocate more than nnnn kilobytes of extended memory. The default value is 4 gigabytes. In Windows, the default value is one-half the available memory.
EXTMIN nnnn	If fewer than nnnn kilobytes are available after applying EXTMAX and EXTLEAVE limits, terminate with an Out of Memory message. The default value is zero.
REALLEAVE nnnn	Always leave at least nnnn paragraphs of real memory available. The default value is 64K or 4096 paragraphs.
REALMAX nnnn	Don't allocate more than nnnn paragraphs of real memory. The default value is 1 megabyte or 65,535 paragraphs.
REALMIN nnnn	If fewer than nnnn paragraphs are available after applying REALMAX and REALLEAVE, terminate with an Out of Memory message. The default value is zero.

Running other programs from the IDE

You can run other programs, tools, and utilities without leaving the IDE. The IDE lets you run Turbo Debugger, Resource Workshop, GREP, WinSight, WinSpector, and Keymapper. To run a program from the IDE, choose Tools | *ProgramName* (for example, Tools | GREP).

To add programs to the Tools menu,

1. Choose Options | Tools. The Tools dialog box appears, listing Tools, Viewers, and Translators.
2. Click New. If you want to add an existing tool (listed in the Tools dialog box), click Edit.
3. Type the name of the program, its path, and any command-line options you always want to pass to it. (You can use transfer macros on this command line; see the online Help for more information.)
4. Type Menu text. This text can appear on SpeedMenus and on the Tools main menu. If you want to assign a shortcut key to your menu text, precede a letter with an ampersand—this letter will appear underlined in the menu. For example, the shortcut key for File is F. In the menu

text, File would appear as `&File`. If you want an ampersand in your menu text, use two ampersands (`&&Test` appears as `&Test` in the menu).

5. Type any help text you want. Help hint text appears in the status line when you select the menu item.
6. Click Advanced. The Tool Advanced Options dialog box appears.
7. Check Translator if the program uses one file type to create another file (like a compiler). Check Viewer if the program is used to view a file (like an editor).
8. Check Place on Tools menu. Check Place on SpeedMenu if you want the program to appear on the SpeedMenu for the project window (see Chapter 2 for more information on projects).
9. If your program is a Translator, type an extension for the files you want to associate with the program. For example, BCC is a translator for .C and .CPP files, so Translate From would show `.c;.cpp:`. Use a semicolon to separate file extensions and a colon to designate the end of the list.
10. Type an extension for the resulting translated file. For example, BCC converts .CPP files to .OBJ, so `.obj` appears in the Translate to box.
11. Choose OK with all the open dialog boxes.
12. Choose Tool from the main menu to see that your program name was added correctly to the Tool menu.

CHAPTER 2

Using the project manager

Borland C++ 4.0 features a new project manager with expanded functionality. This chapter describes how to use Borland C++ to build applications and projects from your source code files and how to use .PRJ files from previous versions of the project manager. If you have used earlier versions of the project manager, read this chapter carefully.

The project manager handles applications that are built from many components. Applications can have several source modules that must be compiled with different options. For example, to create an .EXE, resource scripts must be compiled with the resource compiler, import libraries must be created, and .OBJs must be linked.

What is project management?

The project manager reads .PRJ files from previous releases.

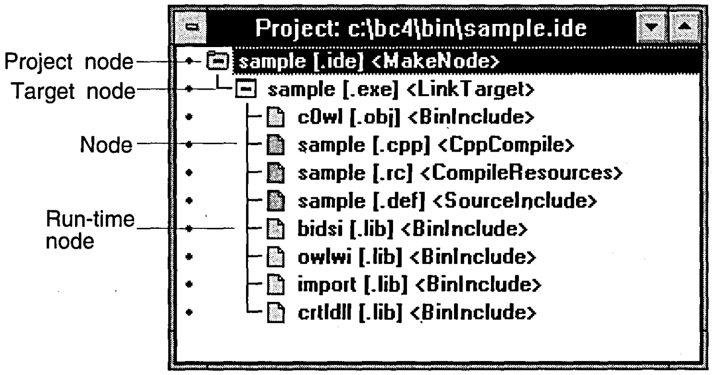
The project manager organizes and updates complex applications by keeping track of all the files and their interdependencies in a project file with the extension .IDE.

Using the project manager is an efficient way to build projects because it only translates the files that have changed since the last build of the project. The term *translate* refers to using one file type to create another. For example, the C++ compiler is a translator for .CPP files because it uses them to generate .OBJ files (see page 40 for more information on translators).

A project can be viewed as a list of files dependent on each other. Some files are source code you create; others, like .OBJ files, .EXE files, and .DLL files are produced by the compiler, linker, or other tools and are dependent on your source code files.

In the project manager, dependencies are shown graphically (this is the project tree). On each level, the files shown in a project are dependent on the files indented beneath them, as shown in Figure 2.1.

Figure 2.1
The project tree



```
sample [.ide] <MakeNode>
  sample [.exe] <LinkTarget>
    c0wl [.obj] <BinInclude>
    sample [.cpp] <CppCompile>
    sample [.rc] <CompileResources>
    sample [.def] <SourceInclude>
    bidsi [.lib] <BinInclude>
    owlwi [.lib] <BinInclude>
    import [.lib] <BinInclude>
    crtldll [.lib] <BinInclude>
```

In the project tree, different nodes have different icons.

- A project node represents the entire project. All the files used to build that project appear under it (a project node is similar to a symbolic target in a makefile). A project can contain many target nodes. For example, you might have one project that you use to build two applications and a DLL (three targets).
- A target node represents a file that is created when its dependent nodes are built (a target is usually the .EXE or .DLL that you're creating from source code). You can collapse a target node so that the dependent nodes aren't displayed.
- A node generally refers to a file used to build a target. Files such as .C, .CPP, .H, and .RC are source files associated with nodes.
- A run-time node refers to common files used at run time, such as startup code (.LIB files). You can choose not to view these files (see page 29).

Creating a project

To create a project,

1. Choose Project | New project. Type a path and a name (eight characters or less) for the project, then press *Tab*. You can also use the Browse button to select a path to the project file.
2. Type a name for the first target in your project. This is usually the name of the program you want to create (the .EXE or .DLL).

You can use TargetExpert to change these attributes.

3. Choose a target type:

   - Application is a normal .EXE file.
   - Dynamic Library is a .DLL file.
   - EasyWin is a character-mode application that runs under Windows.
   - Static Library is a .LIB file.
   - Import Library is a .LIB file.
   - Windows Help is a help file (.HLP) that you usually access from a Windows application (.EXE).

4. Choose a platform for your target:

   - Windows 3.*x* is a 16-bit Windows application.
   - Win32 is a 32-bit Windows NT application.
   - DOS Standard is a 16-bit DOS application.
   - DOS Overlay is a 16-bit DOS application that uses overlays.

See the *DOS Reference* for more information on these libraries.

5. If your application is for DOS, check

   - Floating point to link in FP87.lib.
   - Emulation to link in EMU.LIB.
   - No Math to link in the DOS math libraries.
   - Alternate startup to link in C0F*x*.OBJ, which makes SS==DS for all memory models.
   - Check any standard libraries you want to use in your application. Some libraries are checked by default when you choose a target type (you can't uncheck some of these because they are required for the type of target you're creating). If dynamic and static libraries exist, you can choose which type you want to use (Dynamic is usually the default).
   - OWL uses the ObjectWindows libraries. See the *ObjectWindows Programmer's Guide* for more information.
   - Class Library uses the Borland container class libraries discussed in the *Programmer's Guide.*
   - Runtime uses the run-time libraries listed in the *Library Reference.*
   - BWCC uses the Borland Windows Custom Control libraries. See Appendix B.
   - BGI uses the Borland Graphics Interface (available for DOS applications only). See the online file UTILS.TXT.

6. Check Diagnostic if you want to use a diagnostic version of the libraries (this is available for Class Libraries and ObjectWindows; see the

ObjectWindows documentation for more information on diagnostic versions of its libraries).

7. Check Multithread if you want to use the multithread version of the run-time library. Multithread is available only if your platform is Win32.
8. Choose a memory model for your target (Target Model). Models change depending on the target type.
9. Click OK to create the project. A graphical representation of your project appears in a project window. You can change the target attributes you set in steps 2-8 by using the project manager's SpeedMenu (right-click a node or press *Alt+F10*).

The project manager creates a target with one or more dependents—the default dependents depend on the target type. To view which dependents are added for a target type, click the Advanced button in the New Project dialog box. You can select other dependent nodes, then click OK. For DOS application, you can select how you want the stack and data segments to work.

After you create the initial target for a project, you can add, delete, or rearrange nodes and targets to your project. See page 31 for more information on editing projects.

Creating a multiple-target project

Creating multiple-target projects is similar to creating projects with one target:

1. Create a project using the steps described on page 26.
2. Choose Project | New target to add a second target to your application. The New Target dialog box appears.
3. Type a name for the second target and choose a target type (Standard is the default). Choose OK. The project manager adds a new target to your project just as it does for the first target in a project.

To include a DLL for an application in a project, place the DLL node under the .EXE node.

To view a sample project with two targets, open the file MULTITRG.IDE in the EXAMPLES\IDE\MULTITRG directory. This project file builds two versions of the WHELLO program (one that is 16-bit and one that is 32-bit). The project file contains a text file that describes how to use two or more targets in one project file.

With more than one target in a project, you can choose to build a single target, multiple targets, or the whole project. See page 30 for information on building projects.

Converting old projects

The project manager for this release can load and use projects from previous versions of Borland C++ for Windows. Choose Project | Open project, then type the name of the old project file. You can also change the search attributes from `*.IDE` to `*.PRJ` to list the old 3.0 and 3.1 projects.

The project manager converts the old project to a new one. Be sure to save the new project if you want to keep using it with this version of Borland C++. To save the project, choose Options | Save. Make sure Project is checked, then click OK. The new project is saved with the old name and the new .IDE extension.

Converting projects to makefiles

You can convert Borland C++ project files (.IDE) to makefiles (.MAK) from the IDE. To convert a project file to a makefile,

1. Open the project file (.IDE) you want to convert.
2. Choose Project | Generate Makefile. The IDE generates a makefile with the same name as the project file, but with the extension .MAK. The IDE displays the new makefile in an editor window.

Changing the Project View

The project window, by default, displays the project node, target, and dependents. You can control the display by using the Options | Environment dialog box.

1. Choose Options | Environment. The Environment Options dialog box appears.
2. Choose Project View. A list of options appears.
3. Check or uncheck the options you want. A sample node called `WHELLO` changes as you select or deselect options. This sample shows you how all nodes will appear in the project window. The following list describes each option:

   Build translator displays the translator used on the node.

   Code size displays the total size in bytes of code segments. This information appears only after the node has been compiled.

   Data size displays the size in bytes of the data segment. This information appears only after the node has been compiled.

   Description displays a description of the node. You type the description using the Edit node attributes dialog box from the SpeedMenu.

   Location lists the path to the source file associated with the node.

   Name displays the name of the node.

Number of lines displays the number of lines of code in the file associated with the node (note that this displays only after you compile the code).

Node type describes the type of node (for example, `.cpp` or `.c`).

Style Sheet names the Style Sheet attached with the node.

Output names the file (and the path to that file) that the node creates when it is translated. For example, a .CPP node creates an .OBJ file.

Show runtime nodes displays the nodes the project manager uses when the project is built. For example, it lists startup code and libraries.

Show project node displays the node for the entire project. The project node is built when you choose Project | Build all. Note that all targets are dependents of the project node.

Building a project

To build a project,

1. Open the project you want to build using Project | Open project from the main menu.
2. Choose Project | Build all from the main menu to build all the nodes in the project, even if they're up-to-date. Or, choose Project | Make all to build only the nodes whose dependents have changed since the last project build.

   The project manager builds the project using the Default Project Options Style Sheet unless you have attached a different Style Sheet to a node or overridden the options locally. See the section "Using Style Sheets" on page 37 for more information.

The project manager starts at the first target and works down the project until it comes to a node with no dependents. The project manager builds that node first (and other nodes on the same level), then works back up the project tree.

For example, if you have a project with an .EXE target that is dependent on a .CPP file, the project manager builds the .CPP file to an .OBJ, and then the project uses the new .OBJ file to create the .EXE.

If you choose Make all, the project manager checks a file's date and time to see if the file has been updated. If so, the project manager rebuilds that file, then moves up the project tree and checks the next node's file date and time. The project manager checks all the nodes in a project and builds all the out-of-date nodes.

Building part of a project

There are three ways you can build part of a project:

- To build a node and its dependents,

1. Select the node you want to build.
2. Right-click the node (or press *Alt+F10*) and choose Build node from the SpeedMenu. All the dependent nodes are built regardless of whether they're out-of-date.

- To build a project using MAKE,

1. Select the node you want to build.
2. Right-click the node (or press *Alt+F10*) and choose Make node from the SpeedMenu. MAKE builds only the nodes that aren't current. For more information about MAKE and how it chooses which files to build, see Chapter 12, "Using MAKE."

- To translate the individual node,

1. Select the node you want to translate.
2. Choose Project | Compile from the main menu or select the default translation command from the SpeedMenu. For example, if you've selected a .CPP file, the project SpeedMenu contains the command C++ Compile, which compiles only the selected node.

   Project | Compile translates the current node if the project window is selected. If an editor window is selected, Project | Compile translates the the text in the editor.

Editing the project tree

You can edit the project tree using keystrokes or menu commands. Some menu commands appear only on the SpeedMenu. To display a SpeedMenu in the Project window, right-click a node, or select a node and press *Alt+F10*. The options available on the SpeedMenu reflect the type of selected node and vary slightly among node types.

When editing projects, you can add, delete, and move targets and nodes, and you can copy nodes. You can also change node and target attributes.

Editing target attributes with TargetExpert

Target attributes describe a target type. For example, a target can be a 16-bit Windows DLL that you want to change to be 32 bits. You can change attributes for Standard and AppExpert target types, but not for Source Pools (see page 35 for information on Source Pools). Also, you can't change a target type to be another target type (for example, you can't change a Source Pool target to be an AppExpert target type).

To change a Standard or AppExpert target's attributes,

1. Select the target in the project window.
2. Press *Alt+F10* or right-click the target node.
3. Choose TargetExpert from the SpeedMenu. The TargetExpert dialog box appears.
4. Change the target attributes, then choose OK. Target attributes are explained on page 27.

Editing node attributes

Node attributes describe a node and define the options and translator used when translating a node. To edit a node's attributes,

1. Select the node in the project window.
2. Press *Alt+F10* or right-click the node.
3. Choose Edit node attributes from the SpeedMenu. The Node Attributes dialog box appears.
4. Change the node attributes, then choose OK. Node attributes, which usually display in the project window, are defined as follows:

   Name is the name of the node.

   Description is any text that describes the node.

   Style Sheet is the name of the Style Sheet the project manager uses when it translates that node. If `<<None>>` is specified, the project manager uses the parent's Style Sheet.

   Translator names the translator used on that node, which is usually the default translator for the node type (CppCompile for a .CPP node). If you change the translator, you'll override the default translator for this node, affecting builds and makes for this node. See page 40 for more information on translators.

   Node type defines the node and the available translators for that node.

Adding and deleting a node

You can also use the SpeedMenu to add nodes. Press *Alt+F10*, then choose Add node (you can add one or more nodes using this command).

To add one node to the project,

1. Select the node you want the new node to appear under. If you want the new node to appear under the target, select the target node.
2. Press *Ins* or right-click the selected node and choose Add node from the SpeedMenu.
3. Choose the file or files you want associated with the new node, or type the name of the node you want to add (if the file you type doesn't exist in the current directory, the IDE creates the file).
4. Choose OK. The new node appears under the selected node.

➡ If you want to add many nodes to a project,

1. Start the Windows File Manager and select the files you want to add as nodes to your project. Make sure you can view the project window in the IDE.
2. Drag the files from the File Manager. The project manager automatically adds them under the selected node.

To delete a node in the project, select the node and press *Del*, or choose Delete node on the SpeedMenu. You can delete many nodes by selecting the ones you want to delete (use *Ctrl* or *Shift* with the left mouse button to select multiple nodes), then pressing *Del*. The project manager asks if you want to delete the nodes before it proceeds.

Adding and deleting targets

To add a target to a project,

1. Choose Project | New target from the main menu.
2. Type the name for the new target and choose a target type:

   Standard (default) can be an executable, DLL, or other file.

   AppExpert is an ObjectWindows-based application. See Chapter 4 for more information on this type of target.

   Source Pool is a collection of files that can be referenced in other targets. See page 35 for more information on using Source Pools.
3. Choose OK. If the target type is Standard, the TargetExpert dialog box appears so you can further define your target (see page 27 for more information on these choices). If the target type is AppExpert, see Chapter 4. If the target type is Source Pool, the target is added to the project and you can add nodes to it immediately.

➡ To delete one or more targets,

1. Select the target and view the SpeedMenu (right-click the target or press *Alt+F10*).
2. Choose Delete node.
3. The project manager asks if you're sure you want to delete the target. Click OK. *Note that you cannot undo this deletion.*

Moving nodes and targets

You can move nodes and targets in several ways:

- Drag the node with the mouse. The node moves under the selected node when you release the mouse button.
- Select the node and press *Alt* and the arrow keys. This moves the selected node up or down through the *visible* nodes. You can also use *Alt* and the right and left arrow keys to move a node through levels of dependencies. For example, if you have a header file dependent on a .CPP file (so the .H file appears under the .CPP in the project window), you can move the header file to the same level as the .CPP file by selecting the header file and pressing *Alt*+left arrow.

Copying nodes

Nodes can be copied completely or by reference. A complete copy lets you take the node and its attributes and put an identical, but *separate*, copy somewhere in the project. A complete copy inherits the attributes from its parent node unless you override any options.

A reference copy lets you take a node and its dependents and reference them in another place in the project; a reference copy isn't distinct—if you add or delete dependents of the original, the reference copy is also updated. A reference copy is a copy of a node and its dependents.

To make a complete copy of a node,

1. Select the node or nodes you want to copy (use *Shift* or *Ctrl* and the mouse to select multiple nodes). If a node has dependents, the dependents are copied automatically—you don't need to select them.
2. Hold down the *Ctrl* key and drag the selected nodes to where you want to place the complete copies.
3. When you release the mouse button, the copied nodes appear. If you edit the original node, the complete copy is not changed.

To make a reference copy,

1. Select the node you want to reference copy. You don't need to select the node's dependents because they are copied automatically.
2. Hold down the *Alt* key and drag the selected node to where you want to place the reference copy.
3. When you release the mouse button, the copied node appears. The reference-copied node appears in a lighter (unbold) font. This helps you remember that the copy is referenced rather than complete. If you edit the original (such as adding or deleting dependents), *all* reference copies are updated.

Warning! If you delete an original node, *all* references to that node are also deleted. You cannot undo this deletion.

Using Source Pools

A Source Pool is a collection of nodes. The Source Pool target isn't built, but can be referenced during a build. Source Pools let different targets use a common set of source code. One use for a Source Pool might be to create two target applications—one 16-bit and the other 32-bit. To see a working example of Source Pools, open the sample project called SRCPOOL.IDE in the EXAMPLES\IDE\SRCPOOL directory. This project file includes a text file that describes how the Source Pool is used in that example.

Source Pools can contain several files that you want to copy by reference in your project. For example, you might have several header files that you want to place throughout your project. If you place these files in a Source Pool, then reference copy the Source Pool throughout the project, you only have to update the original Source Pool. If you need to add a new header file to the collection, you can add it in the original Source Pool and all the referenced copies are automatically updated.

Source Pools are useful when you want to assign a single Style Sheet to multiple targets. For example, if you have three targets in a project and you want all the targets to use the same Style Sheet, you can either attach the Style Sheet to each target individually, or you can move the targets under a Source Pool, then attach the single Style Sheet to the Source Pool node. If you want to reassign a Style Sheet (for example, you want to compile without debug information), you only have to reassign the Style Sheet to the Source Pool—not to each target.

Setting project options

Once you create a project, you might want to change the default build options. These options tell the project manager how to build your project (for example, they specify whether you want to include debugging information in your application).

To change project options,

1. Choose Options | Project. A dialog box appears.
2. Edit the options you want to change. See Chapters 1 and 3 for a description of the options.
3. Choose OK when you're done changing options.

When you build your project, the options you set are used for your entire project. If you create a new project, it receives the project options from the last open project.

There are times when you want to select *different* options for a specific node in the project (for example, you might have a specific file you don't want compiled with debugging information, but you want the rest of your files to include debugging information). To use different options for a node, you can use Local Override or a Style Sheet.

Local Override

Project options can be overridden locally. Local Override is useful when you use project options, but you want to override a particular option for a single node. If you want to override many options, use a separate Style Sheet instead (see the next section on Style Sheets).

To override an option,

1. Choose the node whose options you want to override.
2. Right-click the node (or press *Alt+F10*) and choose Edit local options from the SpeedMenu. The Style Sheet dialog box appears with the options used for that node.
3. Select the option you want to override. The Local Override box is checked automatically.
4. Click OK.

Caution! To undo an override, uncheck Local Override. The checkmark in Local Override shows only when the cursor is in the override option, which makes it difficult to know which options you overrode. The Local Override box is dark gray if no options in that section of options is overridden. The box turns light gray if any option in that section is overridden, but you still must select the individual options to find which one is overridden.

If you find yourself overriding more than one or two options, you might want to create and use a separate Style Sheet for that node instead of using Local Override.

Using Style Sheets

Style Sheets are a collection of build options for a project. Every project uses a default set of options. These "defaults" are saved in a Style Sheet, and by default the project uses a Style Sheet called "Default Project Options". The settings in the Style Sheet determine how the project is built. If all the components in your project can be built with the same options, you can set the options using the Options | Project dialog box (this is a way of editing the "Default Project Options" Style Sheet. If you want to change options for a single node, use Local Override, but if you find you're using Local Override a lot, you might want to use Style Sheets.

When a project is built, the project's Style Sheet is used unless the node being built references a different Style Sheet or uses Local Override (see page 36 for information on using Local Override). You can use Style Sheets *and* Local Override. You might want to do this if you attach Style Sheets to your targets but want to slightly modify (override) the Style Sheet for a node under the target.

When the project manager builds a node, it uses the node's Style Sheet and any Local Override options. If the node doesn't have its own Style Sheet, the project manager uses the Style Sheet of the node's parent. If the parent node doesn't use a Style Sheet, the project manager looks at the next parent, continuing until it uses the project's Style Sheet.

Different nodes in a project usually need to be built with different options. For example, you might want to compile .C files with one set of options but .CPP files with another. Or, you might want to build one target with 16-bit options and another with 32-bit options. To see how Style Sheets can be used in a project, open the project file called STYLESHT.IDE in the directory \EXAMPLES\IDE\STYLESHT. This file uses Style Sheets for each of the targets (two versions of WHELLO). The project also contains a text file that explains the use of Style Sheets.

Attaching a Style Sheet to a node

The project manager contains several Style Sheets that you can use, but you can also create your own. To attach an existing Style Sheet to a node,

1. Select the node and right-click it.
2. Choose Edit node attributes.
3. Select a Style Sheet from the list box.

4. Click OK.

You can also click the Styles button to create a new Style Sheet. See the next section for more information on creating Style Sheets.

Creating a Style Sheet

To create a Style Sheet for a project,

1. Choose Options | Style Sheets from the main menu.
2. Click Create. Type a name for the Style Sheet and press *Enter.*
3. Click Edit. The Style Sheet dialog box appears.
4. Edit the options for your Style Sheet. Most options are described in Chapter 3.
5. Click OK when you've completed setting the options for your new Style Sheet.

Editing Style Sheets

You can edit, rename, and copy existing Style Sheets. Choose Options | Style Sheets to view the Style Sheets dialog box. You can do any of the following tasks from that dialog box:

➡ Compose lets you create a Style Sheet that contains the combined options from one or more Style Sheets:

1. Create a new Style Sheet (click New and type a name), then click Compose.
2. Select a Style Sheet you want included in your new Style Sheet, then click Add.
3. Continue adding Style Sheets, then click OK when you're finished. You can't edit a composed Style Sheet, but you can click Compose again to add or delete Style Sheets from the Composed one.

➡ To copy a Style Sheet,

1. Select the Style Sheet you want to copy and then click Copy.
2. Type a name for the copied Style Sheet, then click OK. You can now click Edit to change any of the copied options. Copying is a fast way to create a Style Sheet that closely resembles another—you only have to change the options you want.

➡ To edit any Style Sheet,

1. Select the Style Sheet and click Edit.
2. Change the options you want, then click OK.

➡ To rename a Style Sheet,

1. Select the Style Sheet and click Rename.
2. Type the new Style Sheet name, then click OK.

➡ To remove a Style Sheet, select it and click Remove.

Sharing Style Sheets

If you create Style Sheets for a project, then choose Project | New project, the new project inherits the Style Sheets (and tools and options) from the old project. However, if you close a project or restart the IDE, you'll have to reopen the project with Style Sheets, then create a new project to inherit the Style Sheets.

Caution!

You can also share Style Sheets between projects another way. Every time you create a project file (.IDE), you also create a Project Description Language file (.PDL), which contains information about the Style Sheets and Tools used in the project. Be careful when editing the text in this file because you run the risk of corrupting the file to the point where the project manager can't read it.

When you open a project file, the project manager opens the .PDL file with the same name as the .IDE file.

To share a Style Sheet between projects,

1. Open the .PDL file containing the Style Sheet you want to share. You can open the .PDL file using any text editor.
2. Search for the Style Sheet's name. For example, if you created a Style Sheet called MYSTYLE, you'll see a section in the .PDL file that starts `{ StyleSheet = "MYSTYLE"`.
3. Copy all the text from the beginning brace to the ending brace. You can copy more than one Style Sheet.
4. Open the .PDL file to receive the copied Style Sheet.
5. Find the section for Style Sheets, then paste the copied text to the end of the existing Style Sheet list.
6. Save the .PDL file that received the copied Style Sheet. When you open the project associated with the updated .PDL file, you'll see the pasted Style Sheets when you choose Options | Style Sheets.

Viewing options in a project

Because each node can have its own Style Sheet *and* you can override the options in the Style Sheet, you need a quick way to view the options for each node.

To view the hierarchy of options,

1. Right-click any node in the project and choose View options hierarchy.

   The Options Hierarchy dialog box appears, listing the nodes in the project and the options each uses. Autodependency nodes don't appear. You can expand and collapse the list of nodes just like you can in the project window.
2. Click a node you want to view. Its options appear to the right.
3. If you want to edit an option, double-click the option or select it and then click Edit. If the option belongs to a Style Sheet, you'll be editing the entire Style Sheet. If the option is a Local Override option, you'll be editing the Local Override options for the selected node.
4. When you finish viewing node options, click Close.

The Options list shows the name of the node in square brackets followed by the name of the Style Sheet for that node. It also lists any options that are overridden for that node. This hierarchy lets you see what options are sent to dependent nodes.

Translators

A translator is any program that changes (*translates*) one file type to another. For example, the compiler is a translator that uses .C and .CPP files to create .OBJs, and the linker is a translator that uses .OBJ, .LIB, .DEF, and .RES files to produce an .EXE file.

The project manager lets you define your own translators. Translators you add to a project remain with that project file—they aren't added as permanent parts of the IDE. However, translators, viewers, other tools, and Style Sheets can be passed to the next project: if you have a project file open that contains added-on tools or Style Sheets, the next project you create (choose Project | New project) inherits the translators, viewers, other tools, and Style Sheets from the previous project. For more information on sharing information between projects, see page 39.

You can view default translators by choosing Options | Tools—this list also shows tools and viewers.

Installing a translator

To install a translator,

1. Choose Options | Tools. A dialog box appears that lists Tools, Viewers, and Translators. You can also install translators by choosing Build Attributes from the project manager SpeedMenu.
2. Click New.
3. Type the name of the translator, the path to the translator, and any command-line options for the translator. You can use transfer macros on this command line. For more information on transfer macros, see the online Help.
4. Type Menu text. This text can appear on SpeedMenus and on the Tools main menu. If you want to assign a shortcut key to your menu text, precede a letter with an ampersand—this letter will appear underlined in the menu. For example, the shortcut key for File is F. In the menu text, File would appear as `&File`. If you want an ampersand in your menu text, use two ampersands (`&&Test` appears as `&Test` in the menu).
5. Type any help text you want. Help hint text appears in the status line when you select the menu item.
6. Click Advanced. The Tool Advanced Options dialog box appears.
7. Check Translator.
8. Check Place on Tools menu if you want this translator to appear on the Tools main menu. Check Place on SpeedMenu if you want it to appear when you right-click a node associated with your translator.
9. Check Target translator if you want the translator to work on targets. When you use this translator, the node becomes a target and the translated file is saved to the Final directory. If you don't check Target translator, the translated file is saved in the Intermediate directory.
10. Type an extension for the files you want to associate with this translator. For example, BCC is a translator for .C and .CPP files, so Translate From would show `.c;.cpp:`. Use a semicolon to separate file extensions and a colon to designate the end of the list.
11. Type an extension for the resulting translated file. For example, BCC converts .CPP files to .OBJ, so `.obj` appears in the Translate to box.
12. If you want your new translator to be the default for a node type, type the file extension and a colon in the Default for box.
13. Choose OK.

Using Special on the SpeedMenu

When you display the SpeedMenu, some node types have a Special command that lists other translators for the type of node you've selected. For example, you see the commands Assembler Output and Preprocess if a .CPP node is selected, but you see the command Implib if you selected a .DLL target node.

Installing viewers and tools

Viewers let you see the contents of the selected node. For example, an editor is a viewer that lets you examine the code in a .CPP file. On the SpeedMenu for a .CPP node, you'll see the Text Edit command. The default editor for the Text Edit view is the IDE editor.

To view a node, either

- Double-click it in the project window, or
- Right-click it and choose View to display a list of the available viewers.

Other node types have other viewers available. For example, Resource Workshop can view .RC files. You can't view an .EXE node in a text editor, but you can choose to view it using the integrated debugger, Turbo Debugger for Windows, the ObjectBrowser, or even as an executing program.

Tools are applications you want to run from the IDE, such as Turbo Debugger and GREP. You can install viewers and tools just like you can translators. For more information, see the steps for installing a translator on page 41.

C H A P T E R 3

Compiling

You can compile applications using either the IDE or the command-line programs BCC.EXE (for 16-bit applications) or BCC32.EXE (for 32-bit applications). You can control how the compiler generates code by using compiler options that specify the type of application you want to build (a debugging version for example), where to find header files and link libraries, how C++ code is handled, and much more.

This chapter is organized into three parts:

- How to compile in the IDE
- How to compile with BCC or BCC32
- Compiler options reference

Table 3.1 cross-references the command-line compiler options with the IDE options. Compiler options are discussed in detail according to their topic groups in the Project Options dialog box.

Compiling in the IDE

This section describes how to compile simple programs using compiler options in the IDE. To learn how to build large projects, read Chapter 2.

The IDE SpeedBar has three compiling buttons that correspond to menu commands:

Project | Compile (*Alt+F9*) compiles the code in the selected editor window using the compiler options set in the Project Options dialog box. If a project window is selected, all the selected nodes in the project are translated; child nodes aren't translated unless they're selected (see Chapter 2 for information on translators).

Project | Make all (*F9*) translates all the out-of-date nodes in a project. If no project is open, all the files in edit windows are built using the default project translators.

Project | Build all translates all nodes in a project—even if they are up-to-date. Project | Build all always starts at the first project node and builds down the project. Click Cancel to stop a build.

Using IDE compiler options

There are two ways to set compiler options in the IDE:

- Choose Options | Project and set the options in the dialog box. These options are used when you compile with no project file loaded. If a project is loaded, these options affect the entire project when it is built.
- Set project options locally for each file; you must use the project manager to do this. See Chapter 2 for information on local options.

For example, to compile code in an editor window (you don't have a project loaded) that generates a 32-bit application for a 80386,

1. Select the editor window displaying the code to compile.
2. Choose Options | Project from the menu.
3. In the Project Options dialog box, click the 32-bit Compiler topic to display the subtopics, then click Processor.
4. The Processor options for 32-bit appear on the right. Check 80386. This option stays on until you change it or exit the IDE. To save the option as a default (so that every time you compile, you get a 32-bit application for a 80386), choose Options | Save from the main menu.
5. Click the Compile button on the SpeedBar, or choose Project | Compile (this command compiles the code in the current editor window if no project is loaded).

Using the command-line compilers

This section explains how to use the command-line compilers (BCC.EXE and BCC32.EXE). BCC.EXE is a 16-bit application that generates 16-bit code. BCC32.EXE is a 32-bit application that generates 32-bit code. BCC and BCC32 work the same, but have different defaults (specified in Table 3.1) and generate different code. Unless specified, instructions and options for BCC also apply to BCC32.

You can use BCC to send files to TLINK or TASM (.ASM files if you have TASM installed on your machine). The general syntax for BCC.EXE is

```
BCC [option [option...]] filename [filename...]
```

To see a list of common compiler options, type `BCC` (without any options or file names), then press *Enter*. The BCC command and each option and file name must be separated by at least one space. Precede each option by either a hyphen (–) or a forward slash (/); for example, to specify an include path type:

```
BCC -Ic:\code\hfiles
```

Options and file names entered on the command line override settings in configuration files.

By default, BCC compiles files without extensions and files with the .CPP extension as C++ files. Files with a .C extension or with extensions other than .CPP, .OBJ, .LIB, or .ASM compile as C files.

BCC.EXE and BCC32.EXE have options that are on by default (these options are marked with bullets in Table 3.1). To turn off a default option, type `BCC -`*`option`*`-`.

By default, BCC tries to link with a module-definition file with the same name as the executable. Use TLINK to link with a module-definition file with a different name. You can't link with more than one module-definition file.

Configuration files

If you repeatedly use a certain set of options at the command-line, you might want to list them in a *configuration file* (a standard ASCII text file). You must separate options by spaces; options can appear on one or more lines.

TURBOC.CFG configures BCC.EXE, and BCC32.CFG configures BCC32.EXE; project files (.IDE) configure the IDE.

By default, BCC.EXE uses a configuration file called TURBOC.CFG, and BCC32.EXE uses BCC32.CFG (these defaults are marked with bullets in Table 3.1). The compilers look for the .CFG files first in the directory where you typed `BCC`, then in the directory where BCC.EXE or BCC32.EXE is stored.

You can create multiple configuration files or modify TURBOC.CFG. To use a configuration file, type `+[`*`path`*`]`*`filename`* at the BCC command line. For example, to use a configuration file called MYCONFIG.CFG, you could use the following command line:

```
BCC +C:\MYCONFIG.CFG mycode.cpp
```

Options typed on the BCC command line override configuration files.

Response files

To specify multiple options or files on the command line, place them in a *response file* (a standard ASCII text file). Response files let you have a longer command line than most operating systems allow.

To use response files,

1. Type the command-line options you want to use in a file and save the file. Options can appear on one or more lines in the file, separated by spaces. Response files shipped with Borland C++ have the .RSP extension.
2. Type `BCC @[path]respfile.rsp`.

You can specify more than one response file by typing `BCC @[path]respfile.rsp @[path]otheresp.rsp`. Options typed at the command line override any option or file name in a response file.

Option precedence rules

BCC.EXE and BCC32.EXE evaluate options from left to right, and follow these rules:

- If you duplicate any option except **–I** or **–L**, the last option typed overrides any earlier one.
- If you list multiple **–L** or **–I** options on the command line, the result is cumulative: the compiler searches all the directories listed, in order from left to right.
- Options typed at the command line override configuration and response file options.

Compiler options reference

Table 3.1 lists compiler options for the IDE and the command line. Most IDE options appear in the Project Options dialog box; if an option doesn't appear in the Project Options dialog box, the IDE equivalent option or command appears in angle brackets <>. Some topic names are abbreviated or repeated in this table. You can find a more detailed explanation for each option on the pages referenced in the table.

Default options for both 16- and 32-bit command-line compilers are marked by a bullet (■); otherwise, the bullet is marked for 16-bit only default (■16) or 32-bit only default (■32). Note that defaults in the IDE are different. The main default difference is that the IDE compiles with debug and browser information, making your compiled files larger than if you compiled with the command-line compilers (the applications will be the same size if you use the same set of options for both IDE and command-line).

Table 3.1: Options summary

	Option	Page	IDE setting	Description
	@filename	45	<use project file name>	Read compiler options from the response file *filename*.
	+filename	45	<none>	Use the alternate configuration file *filename*.
	-1	79	<none>	Generate 80186 instructions.
■16	-2	61	16-bit Compiler|Processor|80286	Generate 80286 protected-mode compatible instructions (16-bit only).
	-3	61	16-bit Compiler|Processor|80386	Generate 16-bit 80386 protected-mode compatible instructions (BCC option).
■32	-3	68	32-bit Compiler|Processor|80386	Generate 32-bit 80386 protected-mode compatible instructions (BCC32 option).
	-4	61	16-bit Compiler|Processor|i486	Generate 16-bit 80486 protected-mode compatible instructions (BCC option).
	-4	68	32-bit Compiler|Processor|i486	Generate 32-bit 80486 protected-mode compatible instructions (BCC32 option).
	-5	68	32-bit Compiler|Processor|Pentium	Generate 32-bit Pentium protected-mode compatible instructions.
	-A	59	Compiler|Source|ANSI	ANSI language compliance.
	-A-, -AT	59	Compiler|Source|Borland extensions	Borland C++ language compliance.
	-AK	60	Compiler|Source|Kernighan and Ritchie	Kernighan and Ritchie language compliance.
	-AU	60	Compiler|Source|UNIX V	UNIX V language compliance.
	-a*n*	68	16- or 32-bit Compiler|Processor|Byte, Word, Double Word	Align to *n*: 1 = Byte, 2 = Word, 4 = Double Word (32-bit only).
■	-a-	62	16-bit Compiler|Processor|Byte	Align to one byte.
	-B	79	<none>	Compile and call the assembler to process assembly code.
■	-b	57	Compiler|Code Generation|Allocate enums and ints	Make enums always integer-sized.
	-b-	57	Compiler|Code Generation|Allocate enums and ints (uncheck)	Make enums byte-sized when possible.
	-C	59	Compiler|Source|Nested Comments	Turn nested comments on.
■	-C-	59	Compiler|Source|Nested Comments (uncheck)	Turn nested comments off.
	-c	79	<Project|Compile>	Compile to .OBJ but don't link.
	-D*name*	57	Compiler|Defines	Define *name* to the null string.
	-D*name=string*	57	Compiler|Defines	Define *name* to *string*.
	-d	57	Compiler|Code Generation|Duplicate strings merged	Merge duplicate strings.
■	-d-	57	Compiler|Code Generation|Duplicate strings merged (uncheck)	Don't merge duplicate strings.

Table 3.1: Options summary (continued)

	Option	Page	IDE setting	Description
	-dc	64	16-bit Compiler|Memory Model|Put strings in code segments	Move string literals from data segment to code segment (16-bit only).
	-e*filename*	79	<Edit node attributes in project manager>	Link to produce *filename.*
	-E*filename*	80	<none>	Use *filename* as the assembler.
	-Fc	59	Compiler|Compiler Output|Generate COMDEFs	Generate COMDEFs (16-bit C only).
	-Ff	64	16-bit Compiler|Memory Model|Automatic far data	Create far variables automatically (16-bit only).
	-Ff=*size*	64	16-bit Compiler|Memory Model|Far Data Threshold	Create far variables automatically; set the threshold to *size* (16-bit only).
	-Fm	80	<none>	Enable the **-Fc**, **-Ff**, and **-Fs** options (16-bit only).
	-Fs	80	<TargetExpert|Alternate startup>	Assume DS = SS in all memory models (16-bit DOS only).
■	-f	58	Compiler|Floating point|No floating point (uncheck)	Allow floating point.
	-f-	58	Compiler|Floating point|No floating point	Don't do floating point.
■	-ff	58	Compiler|Floating point|Fast floating point	Fast floating point.
	-ff-	58	Compiler|Floating point|Fast floating point (uncheck)	Strict ANSI floating point.
	-f87	80	<none>	Use 8087 hardware instructions.
	-f287	80	<TargetExpert and click Fast floating point>	Use 80287 hardware instructions (for DOS applications).
	-G	72	Optimizations|Specific|Executable Speed	Select code for speed.
■	-G-	72	Optimizations|Specific|Executable Size	Select code for size.
■	-g*n*	77	Messages|Stop after *n* warnings	Warnings: stop after *n* messages (100 by default).
	-H	61	Compiler|Precompiled headers|Generate and use	Generate and use precompiled headers.
■	-H-	61	Compiler|Precompiled headers|Do not generate or use	Do not generate or use precompiled headers.
	-Hc	80	<none>	Cache precompiled headers. Must be used with **-H** or **-H*xxx*.**
	-Hu	61	Compiler|Precompiled headers|Use but don't generate	Use but don't generate precompiled headers.
	-H"*xxx*"	61	Compiler|Precompiled headers|Stop precompiling after header	Stop compiling precompiled headers at file "*xxx*" (32-bit only). This must be used with **-H**, **-Hu**, or **-H=*filename*.**
	-H=*filename*	61	Compiler|Precompiled headers|Precompiled header file name	Set the name of the file for precompiled headers.

Table 3.1: Options summary (continued)

	Option	Page	IDE setting	Description
	-h	64	16-bit Compiler|Memory Model|Fast huge pointers	Use fast huge pointer arithmetic (16-bit only).
	-I*path*	55	Directories|Include	Set search path for directories for include files.
■	-i*n*	59	Compiler|Source|Identifier length	Make significant identifier length to be *n* (the default is 32).
■	-Jg	71	C++ Options|Templates|Smart	Generate definitions for all template instances and merge duplicates.
	-Jgd	71	C++ Options|Templates|Global	Generate public definitions for all template instances; duplicate result in redefinition errors.
	-Jgx	71	C++ Options|Templates|External	Generate external references for all template instances.
■	-j*n*	77	Messages|Stop after *n* errors	Errors: stop after *n* messages (25 messages by default).
	-K	57	Compiler|Code Generation|Unsigned characters	Default character type **unsigned**.
■	-K-	57	Compiler|Code Generation|Unsigned characters (uncheck)	Default character type **signed.**
	-K2	69	C++ Options|C++ Compatibility|Don't treat **char** as distinct	Allow only two character types (**unsigned** and **signed**); **char** is treated as signed (16-bit only). Compatibility with Borland C++ 3.1 and earlier.
■	-k	60	Compiler|Debugging|Standard stack frame	Turn on standard stack frame.
	-L*path*	55	Directories|Library	Set search path for library files.
	-l*x*	80	<set Linker options>	Pass option *x* to the linker (can use more than one *x*).
	-l-*x*	80	<set Linker options>	Disable option *x* for the linker.
	-M	80	<check Linker|Map File|Segment, Public, or Detailed>	Instruct the linker to create a map file.
	-mc	63	16-bit Compiler|Memory Model|Compact	Compile using compact memory model (16-bit only).
	-mh	63	16-bit Compiler|Memory Model|Huge	Compile using huge memory model (16-bit only).
	-ml	63	16-bit Compiler|Memory Model|Large	Compile using large memory model (16-bit only).
	-mm	63	16-bit Compiler|Memory Model|Medium	Compile using medium memory model (16-bit only).
	-mm!	63	16-bit Compiler|Memory Model|Medium and Never	Compile using medium memory model; assume DS != SS (16-bit only).
■	-ms	63	16-bit Compiler|Memory Model|Small	Compile using small memory model (16-bit only).

Table 3.1: Options summary (continued)

Option	Page	IDE setting	Description
-ms!	63	16-bit Compiler|Memory Model|Small and Never	Compile using small memory model; assume DS != SS (16-bit only).
-mt	63	16-bit Compiler|Memory Model|Tiny	Compile using tiny memory model (16-bit only).
-mt!	63	16-bit Compiler|Memory Model|Tiny and Never	Compile using tiny memory model; assume DS != SS (16-bit only).
-N	60	Compiler|Debugging|Test stack overflow	Check for stack overflow.
-n*path*	55	Directories|Final	Set the output directory.
-O	73	Optimizations|Size|Jump optimizations	Optimize jumps.
-O1	72	Optimizations|Specific|Executable size	Generate smallest possible code.
-O2	72	Optimizations|Specific|Executable speed	Generate fastest possible code (same as **-Ox**).
-Oa	72	Optimizations|Specific|Assume no pointer aliasing	Optimize assuming pointer expressions aren't aliased on common subexpression evaluation.
-Ob	74	Optimizations|Size|Dead code elimination	Eliminate dead code.
-Oc	72	Optimizations|Specific|Optimize locally	Eliminate duplicate expressions within basic blocks.
-Od	72	Optimizations|Disable all optimizations	Disable all optimizations.
-Oe	74	Optimizations|Size|Global register allocation	Allocate global registers and analyze variable live ranges.
-Og	72	Optimizations|Specific|Optimize globally	Eliminate duplicate expressions within functions.
-Oi	74	Optimizations|Speed|Inline intrinsic functions	Expand common intrinsic functions inline.
-Ol	73	Optimizations|Size|Loop optimization	Compact loops.
-Om	75	Optimizations|Speed|Invariant code motion	Move invariant code out of loops.
-Op	76	Optimizations|Speed|Copy propagation	Propagate copies.
-Os	72	Optimizations|Specific|Executable size	Generate smallest possible code.
-Ot	72	Optimizations|Specific|Executable speed	Generate fastest possible code (same as **-Ox**).
-Ov	76	Optimizations|Speed|Induction variables	Enable loop induction variable and strength reduction.
-OW	74	Optimizations|Size|Windows prolog/epilog	Suppress the inc bp/dec bp on Windows far functions (16-bit only).
-Ox	72	Optimizations|Specific|Executable speed	Generate fastest code; Microsoft compatible.

Table 3.1: Options summary (continued)

	Option	Page	IDE setting	Description
	-o*filename*	81	<none>	Compile source file to *filename*.OBJ.
	-P	81	<use Tools>	Perform a C++ compile regardless of source file extension.
	-P*ext*	81	<use Tools>	Perform a C++ compile and set the default extension to *ext*.
■	-P-	81	<use Tools>	Perform a C++ or C compile depending on source file extension.
	-P-*ext*	81	<use Tools>	Perform a C++ or C compile depending on extension; set default extension to *ext*.
	-p	62	16-bit Compiler|Calling Convention|Pascal	Use Pascal calling convention with 16-bit applications (BCC option).
	-p	68	32-bit Compiler|Calling Convention|Pascal	Use Pascal calling convention with 32-bit applications (BCC32 option).
■16	-p- -pc	62	16-bit Compiler|Calling Convention|C	Use C calling convention (BCC option).
■32	-p- -pc	68	32-bit Compiler|Calling Convention|C	Use C calling convention (BCC32 option).
	-po	57	Compiler|Code Generation|FastThis	Use **fastthis** calling convention for passing **this** parameter in registers (16-bit only).
	-pr	62	16-bit Compiler|Calling Convention|Register	Use fastcall calling convention for passing parameters in registers (BCC option).
	-pr	68	32-bit Compiler|Calling Convention|Register	Use fastcall calling convention for passing parameters in registers (BCC32 option).
	-ps	69	32-bit Compiler|Calling Convention|Standard call	Use stdcall calling convention (32-bit only).
■	-r	58	Compiler|Code Generation|Automatic	Use register variables.
	-r-	58	Compiler|Code Generation|None	Disable the use of register variables.
	-rd	58	Compiler|Code Generation|Register keyword	Allow only declared register variables to be kept in registers.
	-R	61	Compiler|Debugging|Browser reference information in OBJs	Include browser information in generated .OBJ files.
■	-RT	71	C++ options|Exception handling/RTTI|Enable run-time type info	Enable run-time type information.

Table 3.1: Options summary (continued)

	Option	Page	IDE setting	Description
	-S	81	<project manager SpeedMenu>	Produce .ASM output file.
	-T*string*	81	<use Tools>	Pass *string* as an option to TASM, TASM32, or assembler specified with **-E**.
	-T-	81	<use Tools>	Remove all previous assembler options.
	-tD	81	<TargetExpert>	Make a DOS .EXE file.
	-tDc	81	<TargetExpert>	Make a DOS .COM file.
■16	-tDe	81	<TargetExpert>	Make a DOS .EXE file.
	-tW	66	16-bit Compiler|Entry/Exit|Windows all functions exportable	Make the target a GUI .EXE with all functions exportable.
	-tWC	81	<TargetExpert>	Make the target a console .EXE with all functions exportable.
	-tWCD	81	<TargetExpert>	Make the target a console .DLL with all functions exportable.
	-tWCDE	81	<TargetExpert>	Make the target a console .DLL with explicit functions exportable.
	-tWD	67	16-bit Compiler|Entry/Exit|Windows DLL, all functions exported	Make the target a GUI .DLL with all functions exportable.
	-tWDE	67	16-bit Compiler|Entry/Exit|Windows DLL, explicit funcs exported	Make the target a GUI .DLL with explicit functions exportable.
	-tWE	67	16-bit Compiler|Entry/Exit|Windows explicit functions exported	Make the target a GUI .EXE with explicit functions exportable.
	-tWM	81	<TargetExpert and check Multithread>	Make the target multithread (32-bit only).
	-tWS	67	16-bit Compiler|Entry/Exit|Windows smart callbacks, all funcs	Make the target a Windows .EXE that uses smart callbacks (16-bit only).
	-tWSE	67	16-bit Compiler|Entry/Exit|Windows smart callbacks, explicit	Make the target a Windows .EXE that uses smart callbacks, with explicit functions exportable (16-bit only).
	-U*name*	81	<use Local Override in project>	Undefine any previous definitions of *name*.
■	-u	59	Compiler|Compiler Output|Generate underscores	Generate underscores.
	-v, -v-	60	Compiler|Debugging|Debug information in OBJs	Turn on source debugging.
	-vi,-vi-	60	Compiler|Debugging|Out-of-line inline functions	Control expansion of inline functions.
	-V0	70	C++ Options|Virtual Tables|External	External C++ virtual tables.
	-V1	70	C++ Options|Virtual Tables|Public	Public C++ virtual tables.

Table 3.1: Options summary (continued)

	Option	Page	IDE setting	Description
■	-V	70	C++ Options|Virtual Tables|Smart	Use smart C++ virtual tables.
	-Va	69	C++ Options|C++ Comp|Pass class values via reference to temp	Pass class arguments by reference to a temporary variable.
	-Vb	69	C++ Options|C++ Compatibility|Same size as 'this' pointer	Make virtual base class pointer same size as 'this' pointer of the class (16-bit only).
■16	-Vb-	69	C++ Options|C++ Compatibility|Always near	Make virtual base class pointer always near (16-bit only).
	-Vc	69	C++ Options|C++ Compatibility|Disable constructor displacement	Don't change the layout of classes to relax restrictions on member pointers (16-bit only).
	-Vf	64	16-bit Compiler|Memory Model|Far virtual tables	Far C++ virtual tables (16-bit only).
	-Vmd	69	C++ Options|Member Pointer|Smallest for class	Use the smallest representation for member pointers.
	-Vmm	69	C++ Options|Member Pointer|Support multiple inheritance	Member pointers support multiple inheritance.
	-Vmp	69	C++ Options|Member Pointer|Honor precision of member pointers	Honor the declared precision for all member pointer types.
	-Vms	69	C++ Options|Member Pointer|Support single inheritance	Member pointers support single inheritance.
■	-Vmv	69	C++ Options|Member Pointer|Support all cases	Member pointers have no restrictions (most general representation).
	-Vo	81	<none>	Enable all backward compatibility options (-Va, -Vb, -Vc, -Vp, -Vt, -Vv).
	-Vp	70	C++ Options|C++ Compatibility|Push "this" first for Pascal	Pass the 'this' parameter to 'pascal' member functions as the first parameter on the stack.
	-Vs	70	C++ Options|Virtual Tables|Local	Local C++ virtual tables.
	-Vt	70	C++ Options|C++ Comp|Virtual table Pointer follows data members	Place the virtual table pointer after nonstatic data members.
	-Vv	70	C++ Options|C++ Compatibility|'deep' virtual bases	Don't add the hidden members and code to classes with pointers to virtual base class members.
	-W	66	16-bit Compiler|Entry/Exit|Windows all functions exportable	Make the target a GUI .EXE with all functions exportable.
	-WD	67	16-bit Compiler|Entry/Exit|Windows DLL, all functions exportable	Make the target a Windows .DLL with all functions exportable.

Table 3.1: Options summary (continued)

	Option	Page	IDE setting	Description
	-WDE	67	16-bit Compiler|Entry/Exit|Windows DLL, explicit funcs exported	Make the target a Windows .DLL with explicit functions exportable.
	-WE	67	16-bit Compiler|Entry/Exit|Windows explicit functions exported	Make the target a Windows .EXE with explicit functions exportable.
	-WM	81	<TargetExpert and check multithread>	Make the target multithread (32-bit only).
	-WS	67	16-bit|Entry/Exit|Windows smart callbacks, all functions exported	Make the target a Windows .EXE that uses smart callbacks with all functions exportable (16-bit only).
	-WSE	67	16-bit|Entry/Exit|Win smart callbacks, explicit functions exported	Make the target a Windows .EXE that uses smart callbacks, with explicit functions exportable (16-bit only).
	-w!	79	Make|Break Make on warnings	Returns a non-zero return code from the command-line compiler when there are warnings and doesn't compile to .OBJ.
■	-w	77	Messages|All	Display warnings on.
	-w-	77	Messages|None	Don't display warnings.
	-w*xxx*	77	Messages|Selected (see specific warning)	Enable *xxx* warning message.
	-w-*xxx*	77	Messages|Selected (see specific warning)	Disable *xxx* warning message.
	-X	59	Compiler|Compiler Output|Autodependency information (uncheck)	Don't use compiler autodependency output.
■	-X-	59	Compiler|Compiler Output|Autodependency information (check)	Use compiler autodependency output.
■	-x	71	C++ Options|Exception handling|Enable exceptions	Enable exception handling.
■	-xd	71	C++ Options|Exception handling|Enable destructor cleanup	Enable destructor cleanup.
	-xp	71	C++ Options|Exception handling|Enable exception location info	Enable exception location information.
	-Y	81	<TargetExpert DOS Overlay>	Enable overlay code generation.
	-Yo	81	<edit node attributes and check Overlay this module>	Overlay the compiled files.
	-y	60	Compiler|Debugging|Line numbers	Line numbers on.
	-Z	74	Optimizations|Size|Suppress redundant loads	Enable register load suppression optimization.
	-zA*name*	66	16-bit Compiler|Segment Names Code|Code Class	Code class.
	-zB*name*	65	16-bit Compiler|Segment Names Data|UnInitialized Data Class	BSS class.
	-zC*name*	66	16-bit Compiler|Segment Names Code|Code Segment	Code segment.
	-zD*name*	65	16-bit Compiler|Segment Names Data|UnInitialized Data Segment	BSS segment.
	-zE*name*	65	16-bit Compiler|Segment Names Far Data|Far Data Segment	Far segment (16-bit only).
	-zF*name*	65	16-bit Compiler|Segment Names Far Data|Far Data Class	Far class (16-bit only).

Table 3.1: Options summary (continued)

	Option	Page	IDE setting	Description
	-zG*name*	65	16-bit Compiler|Segment Names Data|UnInitialized Data Group	BSS group.
	-zH*name*	65	16-bit Compiler|Segment Names Far Data|Far Data Group	Far group (16-bit only).
	-zP*name*	66	16-bit Compiler|Segment Names Code|Code Group	Code group.
	-zR*name*	65	16-bit Compiler|Segment Names Data|Initialized Data Segment	Data segment.
	-zS*name*	65	16-bit Compiler|Segment Names Data|Initialized Data Group	Data group.
	-zT*name*	65	16-bit Compiler|Segment Names Data|Initialized Data Class	Data class.
	-zV*name*	65	16-bit Compiler|Segment Names Far Data|Virtual Table Segment	Far virtual segment (16-bit only).
	-zW*name*	66	16-bit Compiler|Segment Names Far Data|Virtual Table Class	Far virtual class (16-bit only).
■	-z*X**	82	<none>	Use default name for *X*; *X* is A-H, P, R, S, T, V, or W.

■ Default for both 16- and 32-bit ■16 16-bit default ■32 32-bit default

Directories

–I*path* **Include** searches *path* (the drive specifier or path name of a subdirectory) for include files (in addition to searching the standard places). A drive specifier is a single letter, either uppercase or lowercase, followed by a colon (:). A directory is any valid directory or directory path. You can use more than one **–I** (which is an uppercase I) directory option.

–L*path* **Library** forces the linker to get the C0x.OBJ start-up object file and the Borland C++ library files from the named directory. By default, the linker looks for them in the current directory.

Source is the directory where the compiler looks for source code.

Intermediate is where the compiler places any temporary files it creates.

–n*path* **Final** places any final output files (.OBJ, .I, or .ASM) created by the compiler in the directory or drive named by *path*.

You can enter multiple directories on the command line in the following ways:

- You can stack multiple entries with a single **–L** or **–I** option by using a semicolon:

```
BCC.EXE -Ldirname1;dirname2;dirname3 -Iinc1;inc2;inc3 myfile.c
```

- You can place more than one of each option on the command line, like this:

```
BCC.EXE -Ldirname1 -Ldirname2 -Ldirname3 -Iinc1 -Iinc2 -Iinc3 myfile.c
```

- You can mix listings:

```
BCC.EXE -Ldirname1;dirname2 -Ldirname3 -Iinc1;inc2 -Iinc3 myfile.c
```

If you list multiple **–L** or **–I** options on the command line, the result is cumulative: The compiler searches all the directories listed, in order from left to right. The IDE also supports multiple library directories.

File-search algorithms

The Borland C++ include-file search algorithms search for the header files listed in your source code in the following way:

- If you put an `#include <somefile.h>` statement in your source code, Borland C++ searches for somefile.h only in the specified include directories.
- If, on the other hand, you put an `#include "somefile.h"` statement in your code, Borland C++ searches for somefile.h first in the current directory; if it doesn't find the header file there, it then searches in the include directories specified in the command line.

The library file search algorithms are similar to those for include files:

- Implicit libraries: Borland C++ searches for implicit libraries only in the specified library directories; this is similar to the search algorithm for **#include** <*somefile*.h>. Implicit library files are the ones Borland C++ automatically links in and the start-up object file (C0*x*.OBJ).
- Explicit libraries: Where Borland C++ searches for explicit (user-specified) libraries depends in part on how you list the library file name. Explicit library files are ones you list on the command line or in a project file; these are file names with a .LIB extension.
  - If you list an explicit library file name with no drive or directory (like this: `mylib.lib`), Borland C++ searches for that library in the current directory first. Then (if the first search was unsuccessful), it looks in the specified library directories. This is similar to the search algorithm for **#include** "*somefile*.h".
  - If you list a user-specified library with drive and/or directory information (like this: `c:mystuff\mylib1.lib`), Borland C++ searches *only* in the location you explicitly listed as part of the library path name and not in the specified library directories.

Compiler|Defines

Macro definitions let you define and undefine macros (also called *manifest* or *symbolic* constants) on the command line or in the IDE. Macros defined on the command line or in the Options Setting dialog box override those in your source file. Type IDE macro definitions in the Defines box under the Code Generation | Settings topic.

–D*name* Defines the named identifier *name* to the null string. separate macros with a semicolon.

–D*name=string* Defines the named identifier *name* to the string *string* after the equal sign. *string* can't contain any spaces or tabs. Separate macros with a semicolon.

Borland C++ lets you make multiple **#define** entries on the command line in any of the following ways:

- You can include multiple entries after a single **–D** option by separating entries with a semicolon:

```
BCC.EXE -Dxxx;yyy=1;zzz=NO MYFILE.C
```

- Multiple **–D** options can be included if they are separated by spaces:

```
BCC.EXE -Dxxx -Dyyy=1 -Dzzz=NO MYFILE.C
```

- You can mix multiple **–D** listings with semicolon entries:

```
BCC.EXE -Dxxx -Dyyy=1;zzz=NO MYFILE.C
```

Compiler|Code-generation

–b –b- **Allocate enums as ints** allocates a two-byte **int** (16-bit) or four-byte **int** (32-bit) for enumeration types. This option is on by default. Unchecked (**-b-**), this option allocates the smallest variable size that can hold the enumeration values: the compiler allocates an **unsigned** or **signed char** if the values of the enumeration are within the range of 0 to 255 (minimum) or –128 to 127 (maximum), or an **unsigned** or **signed short** if the values of the enumeration are within the range of 0 to 65,535 (minimum) or –32,768 to 32,767 (maximum). The compiler allocates a two-byte **int** (16-bit) or a four-byte **int** (32-bit) to represent the enumeration values if any value is out of range.

–K –K- **Unsigned characters** treats all **char** declarations as if they were **unsigned char**, which provides compatibility with other compilers. BCC defaults **char** declarations to **signed** (unchecked or **–K-**).

–d **Duplicate strings merged** merges literal strings when one string matches another, producing smaller programs but slightly longer compilation times. This option can cause errors if one string is modified. This option is unchecked by default (**-d-**).

–po **FastThis** uses the **_ _fastthis** calling convention for passing **this** in a register to member functions.

With **fastthis** enabled (16-bit applications only because **fastthis** is always used for 32-bit applications), the compiler compiles member functions to expect their **this** pointer to be passed in a register (or a register pair in 16-bit large data models). Likewise, calls to member functions load the register (or register pair) with **this**.

You can enable **fastthis** using the **–po** command-line option or with the Compiler | Code Generation | FastThis calling option. You can also use the language-specifier keyword **__fastthis**.

In small or flat data models, **this** is supplied in the SI register; 16-bit-large data models use DS:SI. If necessary, the compiler saves and restores DS. All references in the member function to member data are done via the SI register.

The names of member functions compiled with **fastthis** are mangled differently from non-**fastthis** member functions, to prevent mixing the two. It's easiest to compile all classes with **fastthis**, but you can compile some classes with **fastthis** and some without.

–r- **None** doesn't use register variables.

–rd **Register keyword** specifies that register variables are used only if you use the **register** keyword and a register is available. You can use **–rd** in **#pragma options**. Use this option or the **–r** option to optimize the use of registers.

–r **Automatic** uses register variables. The compiler automatically assigns variables to registers if possible, even when you don't specify a register variable by using the **register** type specifier. The **–r** option is on by default.

Compiler|Floating Point

–f –f- **No floating point** (**-f-**) specifies that the program contains no floating-point calculations, so no floating-point libraries are linked. Unchecked (**-f**), this option emulates 80x87 calls at run time.

–ff –ff- **Fast floating point** (**-ff**) optimizes floating-point operations without regard to explicit or implicit type conversions. This option can provide answers faster than under ANSI operating mode. Unchecked (**-ff-**) this option turns off the fast floating-point option. The compiler follows strict ANSI rules regarding floating-point conversions.

Compiler|Compiler Output

-X -X- **Autodependency information** (**-X-**) generates autodependency information. Modules compiled with this option on can use MAKE's autodependency feature. By default, autodependency is turned on (**-X-**).

-u **Generate underscores** automatically puts an underscore in front of identifiers before saving them in the object module. Underscores for C and C++ identifiers are optional, but are on by default. You can set them off with **–u-**. But note that setting the underscores off causes link errors when linking with the standard Borland C++ libraries. See Chapter 10 in the *Programmer's Guide* for details about underscores.

-Fc **Generate COMDEFs** (16-bit only) generates communal variables (COMDEFs) for global C variables that are not initialized and not declared as **static** or **extern**. The advantage of this option is that header files that are included in several source files can contain declarations of global variables. As long as a given variable doesn't need to be initialized to a nonzero value, you don't need to include a definition for it in any of the source files. You can use this option when porting code that uses a similar feature with another implementation.

Compiler|Source

-C **Nested comments** lets you nest comments. Comments normally can't be nested.

-i*n* **Identifier length** causes the compiler to recognize only the first *n* characters of identifier names. All identifiers, whether variables, preprocessor macros, or structure members, are treated as distinct only if their first *n* characters are unique. Specifying *n* to be 0 or greater than 249, or not specifying the **–i*n*** option at all, allows identifiers of unlimited length.

By default, Borland C++ uses 32 characters per identifier. Other systems, including some UNIX compilers, ignore characters beyond the first eight. If you are porting to other environments, you might want to compile your code with a smaller number of significant characters; this helps you locate name conflicts in long identifiers that have been truncated.

-A- -AT **Borland extensions** uses Borland C++ keywords. See Chapter 1 in the *Programmer's Guide* for a complete list of the Borland C++ keywords.

-A **ANSI** compiles ANSI-compatible code. Any Borland C++ keyword is ignored and can be used as a normal identifier.

–AU **UNIX V** uses only UNIX language-extension compliance.

–AK **Kernighan and Ritchie** uses only Kernighan and Ritchie language compliance.

Compiler|Debugging

–k **Standard stack frame** generates a standard stack frame, which is useful when using a debugger to trace back through the stack of called subroutines. This option is on by default. When it's off, any function that doesn't use local variables and has no parameters is compiled with abbreviated entry and return code, which makes your code smaller and faster.

–N **Test stack overflow** generates stack overflow logic at the entry of each function. It causes a stack overflow message to appear when a stack overflow is detected at run time. This is costly in terms of both program size and speed but is provided as an option because stack overflows can be very difficult to detect. If an overflow is detected, the message `Stack overflow!` is printed and the program exits with an exit code of 1.

–vi **Out-of-line inline functions** expands C++ **inline** functions inline. To control the expansion of inline functions, the **–v** option acts slightly different for C++: when inline function expansion isn't enabled, the function is generated and called like any other function. Debugging with inline expansion can be difficult, so Borland C++ provides the following options:

- **–v** turns debugging on and inline expansion off. With this option off, you can link larger .OBJ files. This option doesn't affect execution speed, but it does affect compile time.
- **–v-** turns debugging off and inline expansion on.
- **–vi** turns inline expansion on.
- **–vi-** turns inline expansion off.

For example, if you want to turn both debugging and inline expansion on, you must use **–v –vi**.

–y **Line numbers** includes line numbers in the .OBJ for the IDE's integrated debugger. This increases the size of the .OBJ but doesn't affect size or speed of the executable program. This option is useful with symbolic debuggers. In general, **–v** is more useful than **–y** with the integrated debugger.

The debugging options include debugging information in your generated code. For information on debugging your applications, see Chapter 6; for browsing information, see Chapter 1.

–v	**Debug information in OBJs** includes debugging information in .OBJ files so that they can be debugged with either the integrated debugger or a standalone debugger. The compiler passes this option to the linker so it can include the debugging information in the .EXE file. For debugging, this option treats C++ inline functions as normal functions.
–R	**Browser reference information in OBJs** includes browser information when the compiler generates .OBJ files; this lets you inspect the application using the IDE's integrated Browser. The Browser is described in Chapter 1. When this option is off, you can link larger .OBJ files. This option doesn't affect execution speed, but it does affect compile time.

Compiler|Precompiled headers

–H	**Generate and use** generates and uses precompiled headers using the default file name BCDEF.CSM (16-bit) or BC32DEF.CSM (32-bit) for the command-line compilers and <*projectname*>.CSM for projects in the IDE. Precompiled headers can dramatically increase compile speed, although they require considerable disk space. See page 403 for more information on precompiled headers.
–Hu	**Use but do not generate** uses but doesn't generate precompiled headers.
–H-	**Do not generate or use** doesn't generate or use precompiled headers.
–H=*filename*	**Precompiled header name** generates and uses precompiled headers and sets the name of the file for precompiled headers (other than BCDEF.CSM or BC32DEF.CSM).
–H"*xxx*"	**Stop precompiling after header file** stops compiling precompiled headers when it compiles the file specified as *xxx*.

16-bit Compiler|Processor

–2	**80286** generates 16-bit 80286 protected-mode compatible instructions.
–3	**80386** generates 16-bit 80386 protected-mode compatible instructions.
–4	**i486** generates 16-bit 80486 protected-mode compatible instructions.

-a*n* **Data alignment Byte/Word** align to *n*: 1 = **Byte**, 2 = **Word** (2-bytes). See also **-a*n*** for 32-bit applications on page 68. **Word** (**-a**) forces integer-size and larger items to be aligned on a machine-word boundary. Extra bytes are inserted in a structure to ensure member alignment. Automatic and global variables are aligned properly. **char** and **unsigned char** variables and fields can be placed at any address; all others are placed at an even-numbered address. **Byte** (**-a-**) allows byte-wise alignment. Word alignment increases the speed with which 80x86 processors fetch and store data.

16-bit Compiler|Calling Convention

Calling conventions are discussed in more detail in Chapter 2 in the *Programmer's Guide.*

-pc -p- **C** generates all subroutine calls and all functions using the C calling convention, which is equivalent to declaring all subroutine and functions with the **__cdecl** keyword. The C convention permits a function call to pass a variable number of arguments. You can use the **__cdecl, __pascal,** or **__fastcall** keyword to declare a specific function or subroutine using another calling convention.

-p **Pascal** generates all subroutine calls and functions using the Pascal calling convention, which is equivalent to declaring all subroutine and functions with the **__pascal** keyword. The resulting function calls are usually smaller and faster than they would be if compiled with the C calling convention (-pc). Functions must pass the correct number and type of arguments. You can use the **__cdecl, __stdcall,** or **__fastcall** keyword to specifically declare a function or subroutine using another calling convention.

-pr **Register** generates all subroutine calls and all functions using the Register calling convention, which is equivalent to declaring all subroutine and functions with the **__fastcall** keyword. You can use the **__stdcall, __pascal,** or **__cdecl** keyword to specifically declare a function or subroutine using another calling convention.

16-bit Compiler|Memory Model

Memory model options let you tell the compiler what memory model to use when compiling 16-bit applications (32-bit applications are always flat model). The available memory models are small, medium, compact, and large. See Chapter 8 in the *Programmer's Guide* for in-depth information on memory models.

-ms -ms! **Small** compiles using small memory model (the default). The command-line option **–ms!** compiles using small model and assumes DS != SS. To do this in the IDE, you need to check two options (Small and Never).

-mm -mm! **Medium** compiles using medium memory model. The command-line option **–mm!** compiles using medium model and assumes DS != SS. To do this in the IDE, you need to check two options (Medium and Never).

The net effect of the **–ms!** and **–mm!** options is actually very small. If you take the address of a stack variable (parameter or auto), the default (when DS == SS) is to make the resulting pointer a near (DS relative) pointer. This way, you can assign the address to a default sized pointer in those models without problems. When DS != SS, the pointer type created when you take the address of a stack variable is an **_ss** pointer. This means that the pointer can be freely assigned or passed to a far pointer or to a **_ss** pointer. But for the memory models affected, assigning the address to a near or default-sized pointer produces a "Suspicious pointer conversion" warning. Such warnings are usually errors, and the warning defaults to on.

The net effect of the **–mt!**, **–ms!**, and **–mm!** options is actually very small. If you take the address of a stack variable (auto or parameter), the default (when DS == SS) is to make the resulting pointer a near (DS relative) pointer. This way, you can simply assign the address to a default-sized pointer in those models without problems. When DS != SS, the pointer type created when you take the address of a stack variable is an **_ss** pointer. This means that the pointer can be freely assigned or passed to a far pointer or to a **_ss** pointer. But for the memory models affected, assigning the address to a near or default-sized pointer produces a "Suspicious pointer conversion" warning. Such warnings are usually errors, and the warning defaults to on.

-mc **Compact** compiles using compact memory model.

-ml **Large** compiles using large memory model.

-mh **Huge** compiles using huge memory model.

-mt -mt! **Tiny** compiles using tiny memory model. The command-line option **–mt!** compiles using small model and assumes DS != SS. To do this in the IDE, you need to check two options (Tiny and Never).

Default for model uses the model to determine if the stack segment is equal to the data segment.

-Fs- **Never** assumes that the data segment is never equal to the stack segment, regardless of the memory model.

Always assumes that DS is equal to SS in all memory models; you can use it when porting code originally written for an implementation that makes the stack part of the data segment.

–dc **Put strings in code segments** moves all string literals from the data segment to the code segment of the generated object file, making the data type **const** (16-bit only). Using this options saves data segment space. In large programs, especially those with a large number of literal strings, this option shifts the burden from the data segment to the code segment

–Ff **Automatic far data** changes global variables greater than or equal to the threshold size to **far**. The threshold size defaults to 32,767. This option is useful for code that doesn't use the huge memory model but declares enough large global variables that their total size exceeds (or is close to) 64K. For tiny, small, and medium models this option has no effect. If you use this option in conjunction with **–Fc**, the generated COMDEFs become **far** in the compact, large, and huge models.

–Vf **Far virtual tables** causes virtual tables to be created in the code segment instead of the data segment (unless changed using the **–zV** and **–zW** options), and makes virtual table pointers into full 32-bit pointers (the latter is done automatically if you are using the huge memory model).

There are two primary reasons for using this option: to remove the virtual tables from the data segment, which might be getting full, and to be able to share objects (of classes with virtual functions) between modules that use different data segments (for example, a DLL and an executable using that DLL). For all modules that can share objects, you must compile either entirely with or entirely without this option. You can get the same effect by using the **huge** or **_export** modifiers on a class-by-class basis.

–h **Fast huge pointers** offers an alternative method of calculating huge pointer expressions; this method is much faster than the standard method, but must be used with caution. When you use this option, huge pointers are normalized only when a segment wraparound occurs in the offset part. This causes problems with huge arrays if any array elements cross a segment boundary. This option is off by default.

Normally, Borland C++ normalizes a huge pointer whenever adding to or subtracting from it. This ensures that, for example, if you have a huge array of **struct**s that's larger than 64K, indexing into the array and selecting a **struct** field always works with **struct**s of any size. Borland C++ accomplishes this by always normalizing the results of huge pointer operations, so that the offset part contains a number that is no higher than 15, and a segment wraparound never occurs with huge pointers. The disadvantage of this approach is that it tends to be quite expensive in terms of execution speed. This option is automatically selected when compiling for Windows.

-Ff=*size*	**Far Data Threshold** changes the point where data is forced to be **far** (used by the **-Ff** option).

16-bit Compiler|Segment Names Data

Use these options only if you have a good understanding of segmentation on the 80*x*86 processor. Under normal circumstances, you don't need to specify segment names.

-zR*name*	**Initialized Data Segment** sets the name of the initialized data segment to *name*. By default, the initialized data segment is named _DATA.
-zS*name*	**Initialized Data Group** changes the name of the initialized data segment group to *name*. By default, the data group is named DGROUP.
-zT*name*	**Initialized Data Class** sets the name of the initialized data segment class to *name*. By default, the initialized data segment class is named DATA.
-zD*name*	**Uninitialized Data Segment** changes the name of the uninitialized data segment to *name*. By default, the uninitialized data segment is named _BSS.
-zG*name*	**Uninitialized Data Group** changes the name of the uninitialized data segment group to *name*. By default, the data group is named DGROUP.
-zB*name*	**Uninitialized Data Class** changes the name of the uninitialized data segment class to *name*. By default, the uninitialized data segments are assigned to class BSS.

16-bit Compiler|Segment Names Far Data

-zE*name*	**Far Data Segment** changes the name of the segment where **__far** objects are put to *name*. By default, the segment name is the name of the source file followed by _DATA. A name beginning with an asterisk (*) indicates that the default string should be used (16-bit only).
-zH*name*	**Far Data Group** causes **__far** objects to be put into group *name*. By default, **__far** objects aren't put into a group. A name beginning with an asterisk (*) indicates that the default string should be used (16-bit only).
-zF*name*	**Far Data Class** changes the name of the class for **__far** objects to *name*. By default, the name is FAR_DATA. A name beginning with an asterisk (*) indicates that the default string should be used (16-bit only).

-zV*name* **Far Virtual Tables Segment** sets the name of the far virtual table segment to *name.* By default, far virtual tables are generated in the code segment (16-bit only).

-zW*name* **Far Virtual Tables Class** sets the name of the far virtual table class segment to *name.* By default, far virtual table classes are generated in the CODE segment (16-bit only).

16-bit Compiler|Segment Names Code

-zC*name* **Code Segment** changes the name of the code segment to *name.* By default, the code segment is named _TEXT.

-zP*name* **Code Group** causes any output files to be generated with a code group for the code segment named *name.*

-zA*name* **Code Class** changes the name of the code segment class to *name.* By default, the code segment is assigned to class CODE.

16-bit Compiler|Entry/Exit Code

Entry/Exit code options specify what type of application the compiler creates. Although these options are listed in the 16-bit compiler section, they also work for 32-bit applications. Use TargetExpert to specify if your application is 16-bit or 32-bit (see Chapter 2 for more information).

-tW -W -WC- **Windows all functions exportable** creates a Windows object module with all far functions exportable. This option creates the most general kind of Windows executable, although not necessarily the most efficient. This is the default option (**-W-** is the default). This option generates the necessary overhead information for every **far** function, whether the function needs it or not. It assumes that all functions are capable of being called by the Windows kernel or by other modules.

This option, when used with a 16-bit application, creates a Windows .EXE function prolog/epilog for all **far** functions, then sets up those functions to be called from another module. To actually export the function address from the .EXE to a .DLL, the code includes a call to *MakeProcInstance(),* passing the resulting pointer to the .DLL requesting the address of the function. To actually export the function address from the .DLL, function names need to be included in the .DEF file of the executable.

-tWE -WE **Windows explicit functions exported** creates a Windows object module in which only those functions declared as **_export** functions are exportable. Use this option if you have functions that aren't called by the Windows kernel. Windows Explicit Functions Exported operates the same as Windows All Functions Exportable except that only those functions marked with the _export keyword (and methods of classes marked as _export) are given the extra prolog/epilog.

This option is far more efficient for 16-bit applications than Windows All Functions Exportable, because only those functions called from outside the module get the overhead of the prolog. This option does, however, require determining in advance which functions/classes need to be exported. MakeProcInstance() is still used, but no .DEF file manipulation is needed.

-tWS -WS **Windows smart callbacks, all functions exportable** (16-bit only) creates an object module with functions using smart callbacks and all functions exported. Use this option only if the compiler can assume that DS == SS for all functions in the module (which is the vast majority of Windows programs and the default for Borland tools).

This option creates a Windows EXE function prolog/epilog for all 'far' functions, then sets up those functions to be called from another module. *MakeProcInstance()* need not be called and no .DEF file editing is needed.

-tWSE -WSE **Windows smart callbacks, explicit functions exportable** creates a 16-bit Windows application with smart callbacks and explicit exported functions. This option is the same as Windows Smart Callbacks except that only those functions marked with the _export keyword (and methods of classes marked as _export) are given the extra prolog/epilog. This is far more efficient because only those functions called from outside the module get the overhead of the prolog. This option does, however, require determining in advance which functions/classes need to be exported.

-tWD -WD **Windows DLL, all functions exportable** creates a DLL object module with all functions exportable. This option creates a Windows DLL function prolog/epilog for all 'far' functions, then sets up those functions to be called from another module. To actually export the function address from the .DLL, function names need to be included in the .DEF file of the executable.

-tWDE -WDE **Windows DLL, explicit functions exported** creates a DLL object module in which only those functions marked as **_export** are exportable. The Windows DLL, Explicit Functions Exported is the same as Windows DLL, All Functions Exportable, except that only those functions marked with the _export keyword (and methods of classes marked as _export) are given the extra prolog/epilog. This is far more efficient than Windows DLL, All

Functions Exportable because only those functions called from outside the module get the overhead of the prolog. This option does, however, require determining in advance which functions/classes need to be exported. No .DEF file manipulation is needed.

32-bit Compiler|Processor

–3 **80386** generates 32-bit 80386 protected-mode compatible instructions.

–4 **i486** generates 32-bit 80486 protected-mode compatible instructions.

–5 **Pentium** generates Pentium protected-mode compatible instructions.

–a*n* **Data alignment Byte/Word/Double word** aligns to *n* at the command-line where *n* matches the IDE options as follows: 1 = **Byte,** 2 = **Word** (2-bytes), 4 = **Double word** (4-bytes). Word alignment increases the speed with which 80x86 processors fetch and store data. See page 62 for information on using this option with 16-bit applications.

32-bit Compiler|Calling Convention

Calling conventions are discussed in more detail in Chapter 2 in the *Programmer's Guide.*

-pc -p- **C** generates all subroutine calls and all functions using the C calling convention, which is equivalent to declaring all subroutine and functions with the _ _**cdecl** keyword. The C convention permits a function call to pass a variable number of arguments. You can use the _ _**cdecl,** _ _**pascal,** or _ _**fastcall** keyword to declare a specific function or subroutine using another calling convention.

–p **Pascal** generates all subroutine calls and functions using the Pascal calling convention, which is equivalent to declaring all subroutine and functions with the _ _**pascal** keyword. The resulting function calls are usually smaller and faster than they would be if compiled with the C calling convention (-pc). Functions must pass the correct number and type of arguments. You can use the _ _**cdecl,** _ _**stdcall,** or _ _**fastcall** keyword to specifically declare a function or subroutine using another calling convention.

–pr **Register** generates all subroutine calls and all functions using the Register calling convention, which is equivalent to declaring all subroutine and functions with the _ _**fastcall** keyword. You can use the _ _**stdcall,**

__pascal, __fastcall, or **__cdecl** keyword to specifically declare a function or subroutine using another calling convention.

–ps **Standard call** uses stdcall calling conventions. This option tells the compiler to use Pascal ordering for pushing parameters. Parameters are pushed starting from left to right.

C++ Options|Member Pointer

–Vmp **Honor precision of member pointers** uses the declared precision for member pointer types. Use this option when a pointer to a derived class is explicitly cast as a pointer-to-member of a simpler base class (when the pointer is actually pointing to a derived class member).

–Vmv **Support all cases** lets member pointers point to any members. Member pointers use the most general (but not always the most efficient) representation.

–Vmm **Support multiple inheritance** lets member pointers point to members of multiple inheritance classes except members of virtual base classes.

–Vms **Support single inheritance** lets member pointers point to members of single inheritance classes only.

–Vmd **Smallest for class** uses the smallest representation that lets member pointers point to all members of their class. If the class isn't fully defined at the point where the member pointer type is declared, the most general representation is chosen by the compiler (a warning is issued).

C++ Options|C++ Compatibility

–K2 **Do not treat 'char' as distince type** treats **char** as signed. Compatibility with Borland C++ 3.1 and earlier (16-bit only).

–Vb- **Always near** stores a hidden pointer as near. When a class inherits virtually from a base class, the compiler stores a hidden pointer in the class object to access the virtual base class subobject. This option generates smaller and more efficient code.

–Vb **Same size as 'this' pointer** matches the hidden pointer to the size of the 'this' pointer used by the class itself.

–Va **Pass class values via reference to temporary**. When an argument of type class with constructors is passed by value to a function, this option instructs the compiler to create a temporary variable at the calling site, initialize this temporary variable with the argument value, and pass a reference to this temporary to the function.

-Vc **Disable constructor displacements**. When the Disable Constructor Displacements option is on, the compiler doesn't add hidden members and code to a derived class (the default). This option ensures compatibility with previous versions of the compiler.

-Vp **Push 'this' first for Pascal member functions** directs the compiler to pass the 'this' parameter to 'pascal' member functions as the first parameter on the stack. By default, the compiler passes the 'this' parameter as the last parameter on the stack.

-Vv **'deep' virtual bases**. When a derived class overrides a virtual function that it inherits from a virtual base class, and a constructor or destructor for the derived class calls that virtual function using a pointer to the virtual base class, the compiler can sometimes add hidden members to the derived class, and add more code to its constructors and destructors. This option directs the compiler *not* to add the hidden members and code, so that class instance layout is same as with previous versions of Borland C++.

-Vt **Virtual table pointer follows data members** places the virtual table pointer after any nonstatic data members of the particular class, ensuring compatibility when class instances are shared with non-C++ code and when sharing classes with code compiled with previous versions of Borland C++.

C++ Options|Virtual Tables

The **–V*n*** option controls the C++ virtual tables. It has five variations:

-V **Smart** generates common C++ virtual tables and out-of-line **inline** functions across modules within your application. As a result, only one instance of a given virtual table or out-of-line **inline** function is included in the program. This produces the smallest and most efficient executables, but uses .OBJ and .ASM extensions only available with TLINK or TASM.

-Vs **Local** generates local virtual tables and out-of-line **inline** functions. As a result, each module gets its own private copy of each virtual table or out-of-line **inline** function it uses; this setting produces larger executables than the Smart setting.

-V0 **External** creates external references to virtual tables. If you don't want to use the Smart or Local options, you can use External and Public to produce and reference global virtual tables.

-V1 **Public** produces public definitions for virtual tables. When using External or Public options, at least one of the modules in the program must be compiled with the Public option to supply the definitions for the virtual tables. All other modules should be compiled with the **-V0** option to refer to that Public copy of the virtual tables.

C++ Options|Templates

For more information about templates, see Chapter 3 in the *Programmer's Guide.*

-Jg **Smart** generates public definitions of all template instances. If more than one module generates the same template instance, the linker merges them to produce a single copy of the instance. To generate the instances, however, the compiler must have available the function body (in the case of a template function) or the bodies of member functions and definitions for static data members (in the case of a template class).

-Jgd **Global** generates public definitions for all template instances encountered. Duplicate instances are *not* merged, causing the linker to report public symbol redefinition errors if more than one module defines the same template instance.

-Jgx **External** generates external references to template instances. Make sure instances are publicly defined in some other module (using the **-Jgd** option), so that external references are properly resolved.

C++ Options|Exception handling/RTTI

-x **Enable exceptions** enables C++ exception handling. If you use C++ exception handling constructs in your code and compile with this option disabled (by unchecking the option in the IDE or using the **-x-** command-line option), you'll get an error.

-xp **Enable exception location information** makes available run-time identification of exceptions by providing the line numbers in the source code where the exception occurred. This lets the program query the file and line number from where a C++ exception occurred.

-xd **Enable destructor cleanup** destructors are called for all automatically declared objects between the scope of the catch and throw statements when an exception is thrown. Note that destructors aren't automatically called for dynamic objects and dynamic objects aren't automatically freed.

-RT **Enable runtime type information** generates code that allows run-time type identification.

Optimizations

The Borland compiler contains an optimizer for improving your application's speed or reducing its size. Compiling takes only 50% longer for full speed optimizations and 20% longer for size optimizations. You can compile with optimizations any time during your project cycle. When debugging, compiling with optimizations on can sometimes help reveal bugs in your code (the integrated debugger works with optimized code).

-Od **Disable all optimizations** turns off all optimizations. You can override this using named options sets in the project manager.

Optimizations|Specific

-O2 -Ot -Ox -G **Executable speed** creates the fastest code. The compiler determines whether it can safely generate code to perform a **rep movsw** instruction instead of calling a helper function to do the copy. This produces faster structure copies for structures and unions over eight bytes long than does the helper function call. The command-line option **–Ox** is provided for compatibility with the Microsoft compiler.

-O1 -Os -G- **Executable size** creates the smallest code by scanning the generated code for duplicate sequences. When such sequences warrant, the optimizer replaces one sequence of code with a jump to the other and eliminates the first piece of code. This occurs most often with **switch** statements.

No optimization doesn't optimize common subexpressions. This option is on by default. The command-line compilers don't optimize common subexpressions by default, so there is no command-line equivalent option (you don't need to specify an option).

-Oc **Optimize locally** eliminates common subexpressions within groups of statements unbroken by jumps (basic blocks).

-Og **Optimize globally** eliminates duplicate expressions within the target scope and stores the calculated value of those expressions once (instead of recalculating the expression). Although in theory this optimization could reduce code size, it optimizes for speed and rarely results in size reductions. Use this option if you prefer to reuse expressions rather than recalculate them for each instance.

-Oa **Assume no pointer aliasing** affects the way the optimizer performs common subexpression elimination and copy propagation by letting the optimizer maintain copy propagation information across function calls and maintain common subexpression information across some stores. Without this option the optimizer must discard information about copies and subexpressions. Pointer aliasing might create bugs that are hard to spot, so it is applied only when you use **–Oa**.

–Oa controls how the optimizer treats expressions with pointers in them. When compiling with common subexpressions and **–Oa** enabled, the optimizer recognizes `*p * x` as a common subexpression in function **func1**.

```
int g, y;

int func1(int *p)
{
  int x=5;
  y = *p * x;
  g = 3;
  return (*p * x);
}

void func2(void)
{
  g=2;
  func1(&g);  // This is incorrect--the assignment g = 3
              // invalidates the expression *p * x
}
```

Optimizations|Size

-O **Jump optimizations** optimize jumps. When the Jump Optimization option is on, the compiler reduces the code size by eliminating redundant jumps and reorganizing loops and switch statements. When this option is on, the sequences of tracing and stepping in the debugger can be confusing, because there might be multiple lines of source code associated with a particular generated code sequence. When this option is off, you'll get the best stepping results when debugging.

-Ol **Loop optimization** takes advantage of the string move instructions on the 80*x*86 processors by replacing the code for a loop with a string move instruction, making the code faster.

```
int v[100];
void t(void)
{
    int i;
    for (i = 0; i < 100; i++)
      v[i] = 0;
}
```

Depending on the complexity of the operands, the compacted loop code can also be smaller than the corresponding noncompacted loop.

-Z **Suppress redundant loads**, which you should always use when compiling with optimizations, optimizes for both speed and size by keeping track of the values loaded into registers. Values already in a register aren't loaded again.

-Ob **Dead-code elimination** reveals variables that might not be needed. Because the optimizer must determine where variables are no longer used and where their values are needed (live range analysis), you must use Global Register Allocation (**-Oe**) when using **–Ob**.

-OW **Windows prolog/epilog** suppresses the inc bp/dec bp of an exported Windows far function Prolog and Epilog code. If the Debug information in OBJs (**-v**) option is on, this option is disabled because some debugging tools (such as WinSpector or Turbo Debugger for Windows) need the inc bp/dec bp to display stack frame information.

-Oe **Global register allocation**, which you should always use when optimizing code, increases the speed and decreases the size of your application. When the Global Register Allocation option is on, global register allocation and variable live range analysis are enabled.

OptimizationsISpeed

-Oi **Inline intrinsic functions** generates the code for memory functions (such as **strcpy** or **memcmp**) within your function's scope, thus eliminating the need for a function call. The resulting code executes faster, but it is larger. The following functions are inlined with this option:

■ alloca	■ memset	■ strchr	■ strncmp
■ fabs	■ rotl	■ strcmp	■ strncpy
■ memchr	■ rotr	■ strcpy	■ strnset
■ memcmp	■ stpcpy	■ strlen	■ strrchr
■ memcpy	■ strcat	■ strncat	

You can control the inlining of these functions with the pragma **intrinsic**. For example, `#pragma intrinsic strcpy` generates inline code for all subsequent calls to **strcpy** in your function, and `#pragma intrinsic -strcpy` prevents the compiler from inlining **strcpy**. Using these pragmas in a file overrides compiler options.

When inlining any intrinsic function, you must include a prototype for that function before you use it, because the compiler creates a macro that renames the inlined function to a function that the compiler recognizes internally. In the previous example, the compiler would create a macro `#define strcpy _ _strcpy_ _`.

The compiler recognizes calls to functions with two leading and two trailing underscores and tries to match the prototype of that function against its own internally stored prototype. If you don't supply a prototype or if the prototype you supplied doesn't match the compiler's prototype, the compiler rejects the attempt to inline that function and generates an error.

-Om **Invariant code motion** moves invariant code out of loops and optimizes for speed. The optimizer uses information gathered about all the expressions in the function during common subexpression elimination to find expressions whose values don't change inside a loop. To prevent the calculation from being done many times inside the loop, the optimizer moves the code outside the loop so that it is calculated only once. The optimizer then reuses the calculated value inside the loop. You should use loop-invariant code motion whenever you are compiling for speed and you have used global common subexpressions, because moving code out of loops can result in enormous speed gains. For example, in the following code, `x * y * z` is evaluated in every iteration of the loop:

```
int v[10];
void f(void)
{
  int i,x,y,z;
  for (i = 0; i < 10; i++)
    v[i] = x * y * z;
}
```

The optimizer rewrites the code:

```
int v[10];
void f(void)
{
  int i,x,y,z,t1;
  t1 = x * y * z;
  for (i = 0; i < 10; i++)
    v[i] = t1;
}
```

–Op **Copy propagation** is primarily speed optimization, but it never increases the size of your code. Like loop-invariant code motion, copy propagation relies on the analysis performed during common subexpression elimination. Copy propagation means that the optimizer remembers the values assigned to expressions and uses those values instead of loading the value of the assigned expressions. Copies of constants, expressions, and variables can be propagated.

–Ov **Induction variables** creates induction variables and performs strength reduction, which optimizes loops for speed. Use this option when you're compiling for speed and your code contains loops. The optimizer uses induction to create new variables (*induction variables*) from expressions used in loops. The optimizer assures that the operations performed on these new variables are computationally less expensive (reduced in strength) than those used by the original variables.

Optimizations are common if you use array indexing inside loops, because a multiplication operation is required to calculate the position in the array that is indicated by the index. For example, the optimizer creates an induction variable out of the operation `v[i]` in the following code because the `v[i]` operation requires multiplication, which also eliminates the need to preserve the value of *i*:

```
int v[10];
void f(void)
{
  int i,x,y,z;
  for (i = 0; i < 10; i++)
    v[i] = x * y * z;
}
```

With **Induction variables** enabled, the code changes:

```
int v[10];
void f(void)
{
  int i,x,y,z, *p;
  for (p = v; p < &v[10]; p++)
    *p = x * y * z;
}
```

Messages

Messages on by default contain an asterisk next to the command-line option; these options are checked in the IDE.

-w	**All** displays all warning messages.
-wxxx -w-xxx	**Selected** enables the specific warning message typed at the command line or checked in the IDE. Using the pragma **warn** in your source code overrides messages options set either at the command line or in the IDE. See Chapter 5 in the *Programmer's Guide* for more information on pragmas.
-w-	**None** doesn't display warning messages. Errors are still displayed.
-g*n*	**Stop after n warnings** stops compiling after *n* warnings occur in your project.
-j*n*	**Stop after n errors** stops compiling after *n* errors occur in your project.

Messages|Portability

-wrpt*	Nonportable pointer conversion.
-wcpt*	Nonportable pointer comparison.
-wrng*	Constant out of range in comparison.
-wcln	Constant is long.
-wsig	Conversion may lose significant digits.
-wucp	Mixing pointers to different 'char' types.

Messages|ANSI Violations

-wvoi*	Void functions may not return a value.
-wret*	Both return and return with a value used.
-wsus*	Suspicious pointer conversion.
-wstu*	Undefined structure *structure.*
-wdup*	Redefinition of *macro* is not identical.
-wbig*	Hexadecimal value contains more than 3 digits.
-wbbf	Bit fields must be **signed** or **unsigned int.**
-wext*	*Identifier* is declared as both external and static.
-wdpu*	Declare *type* prior to use in prototype.
-wzdi*	Division by zero.
-wbei*	Initializing *identifier* with *identifier.*
-wpin	Initialization is only partially bracketed.
-wnak	Non-ANSI keyword used: *word.*

Messages|Obsolete C++

-wobi*	Base initialization without a class name is now obsolete.
-wofp*	Style of function definition is now obsolete.
-wpre*	Overloaded prefix operator used as a postfix operator.

Messages|Potential C++ Errors

-wnci*	Constant member *identifier* is not initialized.
-weas*	Assigning *type* to enumeration.
-whid*	*Function1* hides virtual function *function2.*
-wncf*	Non-const function *function* called for const object.
-wibc*	Base class *base1* is also a base class of *base2.*
-wdsz*	Array size for 'delete' ignored.
-wnst*	Use qualified name to access nested type *type.*
-whch*	Handler for *xxx* is hidden by previous handler for *yyy.*
-wmpc*	Conversion to *type* will fail for members of virtual base *base.*
-wmpd*	Maximum precision used for member pointer type *type.*
-wntd*	Use '> >' for nested templates instead of '>>'.
-wncf*	Non-volatile function *function* called for volatile object.

Messages|Inefficient C++ Coding

-winl*	Functions containing *identifier* are not expanded inline.
-wlin*	Temporary used to initialize *identifier.*
-wlvc*	Temporary used for parameter in call to *identifier.*

Messages|Potential errors

-wpia*	Possibly incorrect assignment.
-wdef	Possible use of *identifier* before definition.
-wnod	No declaration for function *function.*
-wpro*	Call to function with no prototype.
-wrvl*	Function should return a value.
-wamb	Ambiguous operators need parentheses.
-wccc*	Condition is always true/false.

Messages|Inefficient Coding

-waus*	*Identifier* is assigned a value that is never used.
-wpar*	Parameter *identifier* is never used.
-wuse	*Identifier* is declared but never used.
-wstv	Structure passed by value.
-wrch*	Unreachable code.
-weff*	Code has no effect.

Messages|General

-wasm	Unknown assembler instruction.
-will*	Ill-formed pragma.
-wias*	Array variable *variable* is near.
-wamp	Superfluous & with function.
-wobs*	*Identifier* is obsolete.
-wpch*	Cannot create precompiled header: *header*.

Make

-w!	**Break Make on warnings** returns a non-zero return code from the command-line compiler when there are warnings and doesn't compile to .OBJ.

Command-line options

The options listed here can be used only with the command-line compilers (BCC.EXE and BCC32.EXE). There are no direct equivalent options in the IDE; however, because you can do most of these tasks in the IDE, each option contains directions for the IDE.

-1	Generates extended 80186 instructions. It also generates 80286 programs running in real mode.
-B	Compiles towards assembly and calls TASM to assemble code. If you don't have TASM, using this option generates an error. Also, old versions of TASM might have problems with 32-bit code.
-c	Compiles and assembles the named .C, .CPP, and .ASM files, but does not execute a link command. Choose Project | Compile in the IDE.

–e*filename*	Derives the executable program's name from *filename* by adding the file extension .EXE (the program name is then *filename*.EXE). *filename* must immediately follow the **–e**, with no intervening whitespace. Without this option, the linker derives the .EXE file's name from the name of the first source or object file in the file name list.
–E*filename*	Uses *name* as the name of the assembler to use. By default, TASM is used. In the IDE, you can add a tool for the assembler program you want to use. See Chapter 2 for information on adding tools to the IDE.
–f87	Uses 8087 hardware instructions (16-bit DOS only).
–f287	Uses 80287 hardware instructions (16-bit DOS only). In the IDE, check Fast floating point in TargetExpert when you create a DOS target.
–Fm	Enables all the other **–F** options (**-Fc**, **–Ff** and **–Fs**). You can use it as a handy shortcut when porting code from other compilers. To do this in the IDE, check the IDE options for **–Fc**, **–Ff** and **–Fs**.
–Fs	Assumes that DS is equal to SS in all memory models. You can use this option when porting code originally written for an implementation that makes the stack part of the data segment. When you specify this optin, the compiler links in an alternate startup module (C0F*x*.OBJ) that places the stack in the data segment. In the IDE, check Alternate Startup in the TargetExpert dialog box. This option works with DOS applications only.
–Hc	Cache precompiled headers. Must be used with **–H** or **–H*xxx***. This option is useful when compiling more than one precompiled header.
–l*x*	Passes option *x* to the linker (TLINK for BCC and TLINK32 for BCC32). More than one option can appear after the **–l** (a lowercase l). You can select linker options in the IDE by choosing Options | Project | Linker. See Chapter 9 for a list of linker options.
–l-*x*	Suppresses linker option *x*. More than one option can appear after the **–l-** (a lowercase l). You can check and uncheck linker options in the IDE by choosing Options | Project | Linker.
–M	Forces the linker to produce a full link map. The default is to produce no link map. In the IDE, check Segment, Public, or Detailed in the Linker | MapFile section of the Project Options dialog box.
–o*filename*	Compiles the named file to the specified *filename*.obj.
–P-	Compiles files with the .CPP extension as C++ files; other files compile as C files. In the IDE, use different tools for compiling a project node. See Chapter 2 for more information.

–P	Compiles all files as C++, regardless of extension. In the IDE, use different tools for compiling a project node. See Chapter 2 for more information.
–P*ext*	Compiles all files as C++; it changes the default extension to whatever you specify with *ext*. This option is available because some programmers use .C or another extension as their default extension for C++ code. In the IDE, use different tools for compiling a project node. See Chapter 2 for more information.
–P-*ext*	Compiles based on the extension (.CPP for C++ code, all other file-name extensions for C code) and defines the default extension (other than .CPP). In the IDE, use different tools for compiling a project node. See Chapter 2 for more information.
–S	Generate assembler source compiles the named source files and produces assembly language output files (.ASM), but does not assemble. When you use this option, Borland C++ includes the C or C++ source lines as comments in the produced .ASM file. In the IDE, use different tools for compiling a project node. Select Special | C++ to Assembler from the project window SpeedMenu. See Chapter 2 for more information.
–T*string*	Passes *string* as an option to TASM (or as an option to the assembler defined with **–E**).
–T-	Removes all previously defined assembler options.
–tD –tDe	Creates a 16-bit DOS .EXE file. In the IDE, choose this target type when you create a target for your project.
–tDc	Creates a 16-bit DOS .COM file. You can't create .COM files from the IDE.
–tWC –WC	Creates a 32-bit console mode application. In the IDE, choose this target type when you create a target for your project.
–tWCD –WCD	Creates a 32-bit console mode DLL with all functions exported. In the IDE, choose this target type when you create a target for your project.
–tWCDE –WCDE	Creates a 32-bit console mode DLL with explicit functions exported. In the IDE, choose this target type when you create a target for your project.
–tWM –WM	Creates a multithread application or DLL. Use this option with **–Wm** and **–WCD**. In the IDE, choose this target type when you create a target for your project.
–U*name*	Undefines any previous definitions of the named identifier *name*.
–Vo	This option is a "master switch" that sets on all of the backward-compatibility options listed in this section. It can be used as a handy

shortcut when linking with libraries built with older versions of Borland C++.

-Y	Enable overlay code generation. In the IDE, choose DOS Overlay for the target type when you create a target for your project.
-Yo	Overlay the compiled files. In the IDE, check Overlay this module in the node attributes dialog box for any nodes under a DOS Overlay (**-Y**) target.
-z*X**	Uses the default name for *X*. For example, **-zA*** assigns the default class name CODE to the code segment class.

C H A P T E R 4

Building applications with AppExpert

See Chapter 5 or the online Help for reference material on AppExpert.

This chapter teaches you how to create ObjectWindows 2.0-based Windows applications using AppExpert. AppExpert works with Resource Workshop, ObjectWindows 2.0 classes, and the IDE's project manager to form a visual approach to application generation. You should be familiar with these components to effectively use AppExpert.

AppExpert lets you create a Windows executable with features such as a SpeedBar, a status bar, a menu structure, online Help, and MDI windows. You can also select options to support printing, print preview, and document/view.

AppExpert basics

The process of creating applications with AppExpert consists of four steps:

1. Use AppExpert to define the user interface and application features and to generate the code.
2. Use ClassExpert to add classes and event handlers, to implement virtual functions, and to navigate to existing class source code. ClassExpert can also associate Resource Workshop objects with classes or handlers. You should always use ClassExpert to help you with event handling, virtual function implementation, and instance variables.
3. Use Resource Workshop to edit or add resources.
4. Use the project manager to build the executable.

AppExpert always creates the following files for each application:

AppExpert creates an .APX file that contains important information that ClassExpert uses.

- A project file (.IDE)
- A main source file (.CPP)
- A main header file (.H)
- A resource script file (.RC)
- A resource header file (.RH)
- A database file for the AppExpert source (.APX)

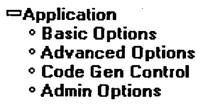
Depending on what options you choose, AppExpert can create the following files:

- Help source files (.RTF)
- A Help project file (.HPJ)
- Icon and bitmap files (.ICO and .BMP)

Creating an application with AppExpert

This section tells you how to create an AppExpert application.

1. Start the IDE and choose Project | AppExpert. A dialog box appears.
2. Type a name for your project file. By default, most generated files (including the .EXE) are derived from the project name (for example, `<prjname>`.CPP).
3. Select a path where you want the AppExpert project file to be stored (AppExpert creates the directory if it doesn't already exist). This directory becomes the default location for all created source files (you can change this directory in the Application Generation Options dialog box before generating the application). (You might want to place each AppExpert project in its own directory for ease of use when making changes to files.) Click OK. The AppExpert Application Generation Options dialog box appears.
4. You can click the Generate button at the bottom of the Options dialog box to generate the default Windows application, or you can change options in the dialog box and then generate the application. The Application Generation Options dialog box contains a list of topics on the left and a brief description of the topic on the right (you can press the Help button for information on the options in that topic). To change application options,



```
Application
  ◦ Basic Options
  ◦ Advanced Options
  ◦ Code Gen Control
  ◦ Admin Options
```

   - View options by clicking any + to display a subtopic (the + means the topic contains subtopics, a – means all subtopics are displayed) then selecting a subtopic. For example, click the + next to the Application topic (or double-click the word "Application"), then select the subtopic Basic Options. The panel on the right displays the basic options for an AppExpert application.
   - Check the options you want in your application. For example, you can edit the Base directory where AppExpert files are saved (which you specified in step 3).
5. Click the Generate button at the bottom of the Options dialog box.

6. A dialog box confirming code generation appears. Click Yes to generate the code (click No to return to setting options). When AppExpert is generating your application, a message box appears.

   AppExpert creates all the files for your application and places them in the Base directory (you can edit the directories before generating the application; see the Base Directory option on page 86).

   With AppExpert, you choose your application options once, then generate the code. After you generate the code and resources, you can edit them and add to them, but you can't go back to AppExpert and change options. For example, if you generate an application that doesn't contain a status line, you can't use AppExpert to add that functionality—you need to add it manually.
7. The project window appears, listing some of the files required for your application (files for bitmaps, icons, and help text don't display). You can use ClassExpert to modify your application or you can build it first (see Chapter 5 for information on ClassExpert). To build your application, choose Project | Make all (you can choose Build all, but Make all is faster). By default, the executable (.EXE) is saved in the Base directory.

Default AppExpert applications

If you don't change any AppExpert options when you generate your application, you get a default application. You can browse through the options to view what the default options are.

If you uncheck options you don't need, the application generates and builds faster.

Application options

Application options control how your application looks.

Multiple Document Interface sets the style of your application to follow the Multiple Document Interface (MDI) model.

Single Document Interface sets the style of your application to follow the Single Document Interface (SDI) model.

Document/View determines whether your application supports the Document/View model for handling application objects. The "document" is the data and the "view" is the user interface to the data. In a Document/View model, these two are separate (see the *ObjectWindows Programmer's Guide* for more information on Document/View). You can use this option with either SDI or MDI applications.

SpeedBar places a SpeedBar at the top of the main window of your application.

Status Line places a status line at the bottom of the main window of your application and generates code to display help hints in the status line when menu items are highlighted.

Drag/Drop supports standard Windows drag-and-drop actions.

Printing supports printing-related activities and creates the menus File | Print Setup, Print Preview, and Print.

Application|Basic Options

Basic Options define where generated code is stored and controls Help file support.

Target Name defines the name of the project you want to create as a basis for the default names of other elements in your project (for example, header files, class database, application class, and source files).

Base Directory sets the base directory path from which all of the project directories are located. All paths in the project are relative to this directory. You can choose a directory by typing it yourself or by selecting it from the Browse dialog box (click the Browse button). The name of this directory is passed to the project manager for the new AppExpert target. The default value for the base directory is the directory of the parent node of the project defined in the project manager. If you specify a new directory, AppExpert creates the directory.

Help File Support generates Help source files (.RTF) and a Help project file (.HPJ). The Help project file is added to the Project Manager project and automatically built with the target application. The Help source file contains placeholder text for the menu items in the application.

Help File Name names the help files (.HLP and .HPJ) associated with your application.

Application| Advanced Options

Advanced Options control the behavior of your application when it starts running, and its appearance.

Start Up sets the initial state of the application's main window.

- Normal (default) starts in a default size (defined by WS_NORMAL).
- Minimized starts as an icon on the Windows desktop.
- Maximized fills the entire Windows desktop when it starts running.

Control Style determines which type of controls the application uses.

- Windows (default) uses standard Windows controls.

- BWCC uses the Borland custom control style.
- 3D uses the new three-dimensional Windows controls.

Application|Code Gen Control

Code Gen Control options name various aspects of the code-generation process and determine where the generated code is stored.

Target Name displays the name of the project as defined in Basic Options | Target.

Base Directory displays the base directory for the project as defined in Basic Options | Base Directory.

Source Directory specifies the directory where the source files for the application are stored. This path is relative to the directory specified as the Base Directory. If an absolute path is specified, it is converted to a path relative to the Base Directory (you can't specify another drive). You can choose a directory by typing it yourself or by selecting one (click the Browse button). The default value for the Source Path is ".\".

Header Directory specifies the directory where the header files for the application are stored. This path is relative to the directory specified as the Base Directory. If an absolute path is specified, it is converted to a path relative to the Base Directory (you can't specify another drive). You can choose a directory by typing it yourself or by selecting one (click the Browse button). The default value for the Header Path is ".\".

Main Source File names the main application source file.

Main Header File names the main application header file.

Application Class names the class that AppExpert derives from *TApplication*. The default class name is based on the project name.

About Dialog Class names the class that AppExpert derives from *TDialog*. The default class name is based on the project name.

Comments documents the generated code partially (terse) or fully (verbose).

Application| Admin Options

Admin Options identify information placed in a comment block at the beginning of all the files generated for the project. Some of the information is displayed in the application's Help | About dialog box.

Version Number sets the project version number that displays in the Help | About dialog box (the default version number is "1.0"). This information is stored in the .RC file for your project.

Copyright defines the copyright information that displays in the Help | About dialog box.

Description describes the application and displays the text in the application's Help | About dialog box. The default value is the name of the project.

Author names the programmers who generate the source code and is used to comment the generated code.

Company names the programmers' company and is used to comment the generated code.

Main Window options

Main Window options control the features of your application's main window—its appearance and type.

Window title names the text for the title bar of the application's main window.

Background color sets the background color of the application's main window; click the Background color button to select a color.

Main Window| Basic Options

Basic Options control the general appearance of the application's main window.

Window Styles controls the appearance of the application's main window, specifying its non-client area styles.

- **Caption** creates a single, thin border and a title bar where a caption can be displayed.
- **Border** puts a single, thin border without a title bar around the main window.
- **Max box** adds a maximize button to the right side of the main window title bar. This option is available only if the Caption option is on.
- **Min box** adds a minimize button to the right side of the main window title bar (available only if the Caption option is on).
- **Vertical scroll** adds a vertical scroll bar to the right side of the main window. This option is available only if you check either Caption or Border.
- **Horizontal scroll** adds a horizontal scroll bar to the bottom of the main window. This option is available only if you check either Caption or Border.

- **System menu** adds a control-menu button on the left side of the main window title bar (available only if the Caption option is on).
- **Visible** makes the main window visible. When Visible is off, the WS_VISIBLE style is changed to NOT WS_VISIBLE.
- **Disabled** disables the main window by default (for example, if you want to display a bitmap when the application is started).
- **Thick frame** puts a double border on the main window and makes the main window resizable.
- **Clip siblings** protects the siblings of the child windows. Painting is restricted to that window (see WS_CLIPSIBLINGS in the API online Help).
- **Clip children** protects child windows from being painted over by the application's main window (see WS_CLIPCHILDREN in the API online Help).

Main Window|SDI Client

SDI Client defines the class that represents the client area of the Single Document Interface main window.

Client/view class names the class of the SDI client area window or view. The interpretation of this value depends on whether you selected the Document/view option in the Application Model settings. If Document/view is selected, Client/view class selects the class of the view of the default document/view. If Document/view is not selected, Client/view class selects the class of the client window.

Table 4.1
Client/view class with Document/view

Document/view on	Document/view off
TEditView (default)	*TEditFile* (default)
TListView	*TListBox*
TWindowView	*TWindow*

This value is automatically mapped to the Document/view setting. For example, if you turn off the Document/view option, *TListView* is switched to *TListBox*. Conversely, if you turn on the Document/view option, *TListBox* switches to *TListView*.

Document class (*TFileDocument* by default) names the class of the default document (available if Document/view is on).

Description describes the class of files associated with the document/view. The default value is "All Files (*.*)".

Filters (*.* by default) lists wildcard file specifications, separated by semicolons or commas, that specify the file names you want the application to recognize. This value is passed to Windows common-file dialog boxes to filter files displayed in them.

Default extension specifies the default file-name extension. This value is passed to Windows common-file dialog boxes to be added to file names when no extension is given. The default extension is used in the File | Open and File | New dialog boxes.

Main Window|MDI Client

MDI Client describes the class that defines the client window of the Multiple Document Interface main window (available if MDI is selected in Application Model settings).

Client class specifies the name AppExpert uses for the class derived from TMDIClient that represents the client area of the MDI frame window.

Source file names the source file that stores the implementation of the class named in Client Class.

Header file names the header file that stores the definition of the class named in Client Class.

MDI Child/View options

MDI Child/View options define the class for child window or document/view (available if MDI and Document/view from Application Model settings are selected).

MDI child names the class derived from TMDIChild that represents the frame of the default MDI child windows.

Source file names the source file that stores the implementation of the class named in MDI child.

Header file names the header file that stores the definition of the class named in MDI child.

MDI Child/View| Basic Options

Basic Options defines the default MDI child window.

MDI client/view class names the class of the default MDI view. The interpretation of this value depends on whether you selected the Document/View option in the Application settings:

Table 4.2
MDI client/view class with Document/view

Document/view on	Document/view off
TEditView (default)	*TEditFile* (default)
TListView	*TListBox*
TWindowView	*TWindow*

This value is automatically mapped to the Document/view settings. For example, if you turn off the Document/view option, *TListView* is switched to *TListBox*. Conversely, if you turn on the Document/view option, *TListBox* switches to *TListView*.

Document class names the class of the document in the default document/view (*TFileDocument* by default).

Description describes the class of files associated with the document/view. The default value is "All Files (*.*)".

Filters (*.* by default) lists wildcard file specifications, separated by semicolons or commas, that specify the file names you want the application to recognize. This value is passed to Windows common file dialog boxes to filter files displayed in them.

Default extension specifies the default file-name extension passed to Windows common file dialog boxes to be added to file names when no extension is given.

C H A P T E R 5

Using ClassExpert

ClassExpert displays virtual functions and events for existing classes and checks the ones implemented in your application.

ClassExpert lets you create new classes, edit and refine the implementation of classes, and navigate through the source code for existing classes in your AppExpert applications. You can use ClassExpert with Resource Workshop to associate classes to resources (for example, associating a *TDialog* class to a dialog resource).

Starting ClassExpert

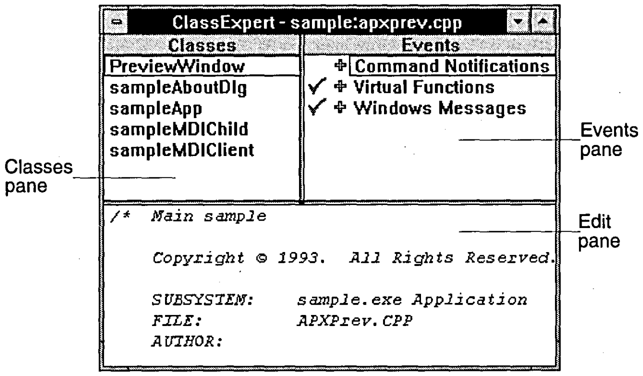
To start ClassExpert,

1. Open an AppExpert project file by choosing Project | Open project.
2. Double-click the AppExpert target node (ClassExpert is the default viewer for AppExpert targets), or choose View | ClassExpert or click the SpeedBar button shown at left. ClassExpert appears, listing the classes and their implementation for your application.

ClassExpert basics

This section describes the three ClassExpert panes and their functionality. You can size the panes by dragging their borders. If you resize ClassExpert, the panes keep their relative proportions.

Figure 5.1
The ClassExpert window



```
/*  Main sample

    Copyright © 1993.  All Rights Reserved.

    SUBSYSTEM:     sample.exe Application
    FILE:          APXPrev.CPP
    AUTHOR:
```

Classes pane

The Classes pane lists the classes ClassExpert manages for the current target. The information in the Events and Edit panes depends on which class is selected here. You can double-click a class to jump to the class constructor source code, which displays in the Edit pane. Using the Class SpeedMenu (right-click in the Classes pane), you can add classes, associate document classes with view classes, get information about a class, jump to the class source code or header file, edit the class, and start Resource Workshop (by choosing Edit dialog or Edit menu).

Events pane

The Events pane lists events and virtual functions from the base class of the class selected in the Classes pane. The information in the Events pane depends on the base class type.

Using the Event SpeedMenu (right-click in the Event pane), you can add or delete message handlers and instance variables.

Edit pane

The Edit pane is an editor that displays the source code for the items selected in the Classes pane and the Events pane. The Edit pane has the same functionality as an IDE editor window; if you make changes or update the IDE editor options, those options are immediately available in the ClassExpert Edit pane.

The Edit pane uses the IDE main menu, and it has a SpeedMenu that you access by right-clicking in the Edit pane. The Edit pane works exactly like an editor window in the IDE except you can't split panes or open other files into the Edit pane.

Adding a class

ClassExpert lets you add ObjectWindows-based classes and supports one level of inheritance (you can manually add more derivations).

To add a class,

1. Right-click in the Classes pane. The SpeedMenu appears.
2. Choose Create new class or click the SpeedBar button shown at left. The Add New Class dialog box appears.
3. Select the ObjectWindows base class you want your class derived from. Press *Tab.*
4. Type the name you want to give to the new class. Press *Tab.*
5. Type the name of the source file you want the source code to appear in. The file is saved in the project's Source path. Press *Tab.*

6. Type the name of the header file that defines the class. This file defaults to the source file name but uses the extension .H. Press *Tab*.
7. Your next selections depend on the base class:

   - If the base class is *TDialog*, you must specify or select a dialog template ID. The Dialog ID list box contains IDs for all dialog resources in your AppExpert application. If you specify an ID that doesn't already exist, AppExpert creates an empty dialog box with your specified ID (for consistency, you might want to use the prefix IDD_), then Resource Workshop loads so you can define the dialog box.
   - If the base class is *TFrameWindow* or a *TFrameWindow*-derived class, you can choose an existing class in the Client class list box to represent the client area of the new frame window.
   - If the base class is *TWindow* or a *TWindow*-derived class, you can click the Set Window Properties button. A dialog box appears where you set properties for the window such as color, border, and caption. See the online Help (click the Help button) for information on each property.
8. Click OK to add the new class.

Creating document types

When you create an AppExpert application that supports document/view, you can use ClassExpert to create view classes and document types.

To create a document type,

1. Create a class for the view unless you want to use one of the three predefined view classes (*TEditView*, *TListView*, or *TWindowView*).
2. Start ClassExpert from your project. Right-click in the Classes pane, then choose Create doc types from the SpeedMenu.
3. Select a View class (if you created your own class, it appears in this list). The default view classes are:

   See the ObjectWindows documentation for more information on these classes.

   - *TEditView* provides a view wrapper for ObjectWindows text edit class.
   - *TListView* provides views for list boxes.
   - *TWindowView* provides window-based views.
4. Type a description for the types of files your document type will support. This text appears in the File | Open dialog box.
5. Type any Filters you want and separate them with commas; these filters appear in the File | Open dialog box and are used to filter for any files a

user can open and use in your application. For example, if you're creating a document type for bitmaps, you might have a filter `*.BMP`.

6. Type a Default extension for your application to use when saving files.
7. Click the Style's button to set styles for the document/view. The styles you can choose are as follows (see the ObjectWindows documentation for more information):

   - *dtAutoDelete* deletes the document object when the last view is closed.
   - *dtNoAutoView* doesn't automatically create a default view type.
   - *dtSingleView* provides only a single view for each document.
   - *dtAutoOpen* opens a document when it's created.
   - *dtUpdateDir* updates the directory with the dialog directory.
   - *dtHidden* hides the template from the list of user selections.
   - *dtSelected* indicates the last selected template.
   - *dtReadOnly* checks the read-only check box when the dialog box is created.
   - *dtOverWritePrompt* asks users if it's OK to overwrite an existing file when they use the Save As dialog box.
   - *dtHideReadOnly* hides the read-only checkbox.
   - *dtPathMustExist* lets the user type only existing paths.
   - *dtFileMustExist* lets the user type only existing file names.
   - *dtCreatePrompt* prompts the user before creating a new document.
   - *dtNoReadOnly* returns the specified file as writeable.

8. Click Add to add the document type to your application (this updates a data structure in the main source file that describes all available document types). The document/view appears in the list of existing types.
9. Repeat steps 1-8 for any document types you want to add. When you're finished, click Close to return to ClassExpert.

Adding and deleting event handlers

To add a handler for an event,

1. Select the class for the message handler. The events appear in the Events pane.
2. Select the event to handle (you might have to expand the list of events), then right-click the event to view the SpeedMenu.
3. Choose Add handler from the SpeedMenu. If you choose to add a handler for a Windows message, ClassExpert adds an entry to the response table whose name is defined by default, then the function

associated with the handler appears in the edit window. Other handlers, such as commands, prompt you for the name of the function before adding the entry to the response table.

4. ClassExpert places a check mark next to the event in the Events pane to show you that the event is handled. A lighter gray checkmark means some events under the event category are handled (expand the list to view the events).

To delete a handler for an event,

1. Select the class for the message handler. The events appear in the Events pane.
2. Select the checkmarked event with the handle you want to remove (you might have to expand the list of events), then right-click the event to view the SpeedMenu.
3. Choose Delete handler. ClassExpert deletes only the entry in the response table, not the code in the source file. The code for the handler appears in the Edit pane, so you can delete it. If you delete the function, remove the function definition from the header file (you can choose Edit header from the Classes pane SpeedMenu to view the file).

Adding and deleting instance variables

Instance variables let you handle lots of controls easily. When you create instance variables, ClassExpert adds a transfer buffer in your code. This transfer buffer collects information at run time, so you can use the information in the transfer buffer instead of creating code that checks whether each check box is checked. For example, if you have a dialog box that has six check-box controls and you want your application to do something based on which boxes are checked, you can use instance variables for each of the controls and then use the transfer buffer data in your code. Consult the *ObjectWindows Programmer's Guide* for more information on transfer buffers.

To add (associate) an instance variable to a control,

1. Select the control in the Events pane (you might need to expand the list of events to view the controls).
2. Right-click the control and choose Add Instance variable.
3. In the Add Instance variable dialog box, type a name for the variable. Click OK. ClassExpert adds the following to your application code:

   - In the header file, it adds a structure declaration with an entry for the instance variable.

- In the class constructor in the .CPP source file, the instance variable is allocated (this associates the ObjectWindows class with the resource object).
- In the .CPP file, a static instance of the transfer structure is declared.

4. The control label in the Events pane shows the class and name of the instance variable you just created.

To delete an instance variable,

1. Select the control with the instance variable you want to delete.
2. Right-click the control and choose Delete Instance variable.
3. ClassExpert deletes the following from your code:
   - The entry from the structure
   - The pointer variable in the class declaration
   - The allocation of the class variable associated with the resource control in the constructor

   If you delete all instance variables from your code, you'll be left with an empty structure and the set transfer buffer call. This information doesn't affect the rest of your code, so you don't need to delete it manually.

Jumping to class source code

To view the source code for a class, select the class in the Classes pane (click the class name once). The code appears in the Edit pane. If you move the cursor in the Edit pane, ClassExpert remembers the position the next time you select the class.

To jump the cursor to the class constructor code, double-click the class name in the Classes pane. To jump to a handled event, double-click the event in the Events pane. You can also view the source file or its header file in an IDE editor:

1. Select the class in the Classes pane.
2. Right-click the class. A SpeedMenu appears.
3. Choose Edit source to view the source file for the class constructor (the .CPP file), or choose Edit header to view the header file where the class is defined.

Using Resource Workshop with ClassExpert

Resource Workshop is the default viewer for ClassExpert resource scripts (.RC files). When you start Resource Workshop from ClassExpert (by right-clicking a class and choosing Edit dialog or Edit menu), Resource Workshop automatically loads the RC script for that application.

When using Resource Workshop with AppExpert-generated code, you should always run it from ClassExpert because Resource Workshop and ClassExpert update each other as you make changes to the project. When you start Resource Workshop, it checks the resource code for changes and sends any updates immediately to ClassExpert. For example, if you add a button to a dialog box, Resource Workshop tells ClassExpert, which then adds the control to the Events pane. To view the control in ClassExpert, select the control in Resource Workshop, right-click it, then choose ClassExpert from the SpeedMenu. Resource Workshop returns you to ClassExpert with the control highlighted in the Events pane.

Running Resource Workshop from the IDE

When you start Resource Workshop as a viewer for an AppExpert application (either through the project manager in the IDE or through ClassExpert), its behavior differs from running Resource Workshop separately:

- When you make changes in Resource Workshop that affect the class structure or functionality (such as editing menus or dialog boxes), those changes are updated instantly in the ClassExpert window.
- You can't open another script (no File | Open or File | New).
- If you close the IDE, Resource Workshop is also closed and any changes you made are automatically saved.
- If you close the AppExpert project file that started Resource Workshop, Resource Workshop is also closed.
- If you build a project while Resource Workshop is open, it creates a .RES file based on the resources loaded. For example, if you edit a dialog box, but haven't saved it, the .RES file will reflect the *unsaved* edits.
- You can access the IDE from Resource Workshop using the SpeedMenu (right-click) and choosing ClassExpert.

Using Rescan

Rescan is a special project tool that examines all the source code listed in your AppExpert project (.IDE file) and updates or rebuilds the project's database (the .APX file) according to what it finds in the source code. Rescan looks for special markers in the source code to reconstruct the AppExpert database file and then starts Resource Workshop to reconstruct information about project resources. If the rescan is successful, the original project database file is renamed to *.~AP and a new database file is created; otherwise, the original database is left as *.APX.

You can use Rescan to:

- Delete a class
- Move a class from one source file to another
- Rename a class, handler, instance variable, or dialog ID
- Import a class from another AppExpert project
- Rebuild a lost or damaged project database (*.APX) file

Deleting a class

To delete a class,

1. Remove the class source file from the IDE project by selecting the source node, right-clicking the node, and choosing Delete node. If the class shares a source file with other classes, delete the code related to the class from the source file and delete references to the class from other source files.
2. Select the AppExpert target in the project, right-click it, then choose Special | Rescan. Rescan scans the source files listed as dependencies for the AppExpert target. Resource Workshop scans and updates the resource files. When Rescan is complete, you'll return to the updated project file where you can either build your application or use ClassExpert. You can add the deleted class to the project by adding the class source file as a dependent of the AppExpert target, then rescanning.

Moving a class

To move a class from one source file to another,

1. Move (cut and paste) the source code of the class to the new file. If the new file isn't in the project as a node under the AppExpert target, add it (see Chapter 2). If the moved class was its own source file, you might want to delete that empty source file from the project.
2. Select the AppExpert target in the project, right-click it to view the SpeedMenu, then select Special | Rescan. When complete, Rescan returns you to the project window in the IDE.

Renaming an AppExpert element

To rename a class, event handler function, instance variable, or dialog ID,

1. Use the IDE editor to search and replace all occurrences of the original name with the new name. Be sure to check all source files associated with the project (.CPP, .H, .RC, and .RH files).

2. In the project window, select the AppExpert target, right-click it, then choose Special | Rescan. When complete, Rescan returns you to the project window in the IDE.

Importing a class

To import a class from one AppExpert project to another,

1. Move or copy the source and header file that defines the class to the source and header directory of the other project. All source files for a project must be in the project's source directory (.CPP files) or header directory (.H files). These directories were created when you first generated the AppExpert project.
2. Add the class source file as a dependent node under the AppExpert target in the IDE project (use Add node from the SpeedMenu).
3. In the project window, select the AppExpert target, right-click, then choose Special | Rescan.

Rebuilding the .APX database file

To rebuild a lost or damaged database file (the .APX file),

1. Open the project file that contains the AppExpert target and dependent nodes (this is the .IDE file).
2. Select the AppExpert target, right-click it, then choose Special | Rescan from the SpeedMenu. Rescan automatically creates a new database file using markers from the source code for the AppExpert application.

C H A P T E R 6

Using the integrated debugger

No matter how careful you are when you code, your program is likely to have *bugs*, or errors, that prevent it from running the way you want it to. *Debugging* is the process of locating and fixing program errors that prevent your programs from operating correctly. This chapter explains how to locate errors in your Windows programs and how to correct them using the IDE.

Types of bugs

There are three basic types of program bugs: compile-time errors, run-time errors, and logic errors.

Compile-time errors

Compile-time errors, or *syntax* errors, occur when your code violates a rule of C or C++ syntax. The IDE can't compile your program unless the program contains valid C or C++ statements.

If your code has syntax errors, the compiler opens the Message window and displays all the errors and warnings. Errors must be fixed one at a time. To correct an error, double-click it and the IDE positions your cursor on the source code line that caused the problem so you can make your correction. If your code had more than one syntax error, you can repeat this process with each error until all the errors are corrected.

Warnings that appear in the Message window don't stop your code from compiling, but they do indicate areas in your code you might want to examine for problems. For example, warnings can alert you to code that isn't portable, code that violates the ANSI standard, or inefficient code. You can choose which type of warnings you want to see:

1. Choose Options | Project and double-click Messages.

   The IDE displays the various message categories and (on the right) the settings that affect those messages.

2. Select the option you want. These options determine which warnings the IDE displays.
   - Select All and the IDE displays all warning messages.
   - Select Selected and the IDE displays only the selected warning messages. (You'll be selecting the messages in Step 4.)
   - Select None and the IDE displays no warnings.
3. Limit the number of error and warning messages displayed in the Message window. In the Stop After boxes, indicate the maximum number of error messages and warning messages you want displayed each time you compile. The number can be any whole number from 0 to 255.

   If you enter 0 in either box, the IDE won't limit the number of messages it displays in the Message window.
4. Under Topics, choose the category of warning messages you want.
5. On the right, select the specific messages you want the IDE to display.

   For example, if you want the compiler to warn you about suspicious pointer conversions that might violate the ANSI standard, choose ANSI Violations and check the Suspicious Pointer Conversion option.

Common causes of compile-time errors are typographical errors, omitted semicolons, references to variables that haven't been declared, passing the wrong number (or type) of arguments to a function, and assigning values of the wrong type to a variable.

After you correct all the errors, you can restart the compilation. Once you've eliminated all the syntax errors and your program compiles successfully, you're ready to run the program and look for run-time errors and logic errors.

Run-time errors

If your program compiles but it fails when you try to run it, you've encountered a run-time error. Your program contains legal statements, but the statements can't be executed properly. For example, your program might be trying to open a nonexistent file for input or to divide by zero. The operating system detects this situation and stops your program from executing.

Logic errors

Logic errors are errors in design and implementation. Your statements are valid and they do *something*, but what they do is not what you intended. These errors are often hard to track down, because the IDE can't find them automatically. Fortunately, the IDE includes debugging features that can help you locate logic errors.

Logic errors occur when variables have incorrect or unexpected values, when graphic images don't look right, or when code isn't executed when you expect it. The rest of this chapter discusses techniques for tracking down these logic errors.

Generating debugging information

You must compile and link your program so that debugging information is generated in your program's .OBJ and .EXE files.

- To add debugging information to your .OBJ files, choose Options | Project to open the Option Settings dialog box, and select Compiler | Debugging | Debug Information in OBJs. This is the default setting.
- To include debugging information in your .EXE files, in the Options Settings dialog box, select Linker | General | Include Debug Information. This is the default setting.

Now when you compile the program, the compiler generates a special table of all the identifiers used, and stores it in the executable file. This list, called the *symbol table,* is used by the debugger to track all variables, constants, types, function names, and statements used in your program.

Specifying program arguments

If the program you want to debug requires that arguments be passed to it, you must specify those arguments:

1. Choose Options | Environment and select the Debugger topic.
2. In the Run Arguments box, type in the arguments you want passed to the program.

Controlling program execution

The most important element of debugging is controlling the execution of your program. Because you can control when each statement is executed, it's easier to determine which part of your program is causing a problem.

Stepping and *tracing* let you run your program one statement at a time; the next statement won't execute until you tell the debugger to continue. You can step or trace until you reach the spot in your code where things go awry. You can then examine the state of the program and its data, view the program's output and the value of its variables, or modify or evaluate expressions in your program before you tell the debugger to execute the next line.

Watching program output

As you step or trace through your program, you can watch your application's output in its window. Set up your windows so you can see both your source code and your application's window as you step and trace. If the IDE desktop window and your application window overlap as you debug, you'll see some flickering in your application window. If you arrange these windows so they don't overlap, your program's execution will be quicker and smoother.

Stepping through code

All execution in the debugger, including stepping, tracing, and halting at breakpoints, is based on lines of source code. If a statement is more than one line on the screen, the statement is still considered to be one line.

You can control the rate of debugging to the level of a single line of source code. If you string several statements together on one line, you can't debug those statements individually. On the other hand, you can spread a single statement out over multiple lines for debugging purposes, and the statement still executes as a single step.

Each time you tell the debugger to step or trace, the *execution point* (the highlighted line that marks your place in the program you're debugging) moves to the next line. The execution point always shows you the next line to be executed.

Stepping is the simplest way to move through your code a little bit at a time. To step through your code, choose Debug | Step Over (or *F8* or the Step Over button on the SpeedBar) to execute the code indicated by the execution point, including any functions it might call before returning control to you. The execution point then indicates the next complete line.

The following example helps explain how stepping works. Let's say that these are the first lines of a program loaded into an edit window:

```
  ...
BOOL InitApplication ( HINSTANCE hInstance )
{
  WNDCLASS  wc;

  wc.style       = CS_HREDRAW | CS_VREDRAW;
  wc.lpfnWndProc = (long (FAR PASCAL*)())MainWndProc;
  wc.cbClsExtra  = 0;
  wc.cbWndExtra  = 0;
      ...
  return ( RegisterClass ( &wc ) );
}

  ...

int PASCAL WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow )
{
  MSG msg;

  if ( !hPrevInstance )
  if ( !InitApplication ( hInstance ) )
    return ( FALSE );

  if ( !InitInstance ( hInstance, nCmdShow ) )
    return ( FALSE );
  ...
}
```

In this example, **InitApplication** is a function you defined in a module that has been compiled with debugging information. If you were actually debugging this program, each time you chose Debug | Step Over (or pressed *F8*), the debugger would execute the line highlighted by the execution point, and then the execution point would advance to the next line. If you chose Step Over when the execution point got to the statement

```
if ( !InitApplication ( hInstance ) )
```

the debugger would execute the **InitApplication** function and return a Boolean value, but you wouldn't see the execution point move through the actual **InitApplication** function. Instead, the debugger would *step over* the function. Stepping returns control to you *after* the function finishes.

Tracing into code

Tracing into code is very much like stepping through code, except that when you come to a line that calls a function, tracing into the code moves the execution point *into* the code in the function. In Listing 6.0, if you chose Debug | Trace Into (*F7* or the Trace Into SpeedBar button) to execute each statement, you'd see the execution point jump to the code that implements the **InitApplication** function when the debugger reached the statement that evaluates the return value of **InitApplication**. As you debug, you can choose to trace into some functions and step over others, depending upon your needs.

Stepping and tracing class member functions

If you use classes in your programs, you can still use the integrated debugger to step and trace. The debugger handles member functions the same way it would step over or trace through functions in a program that is not object-oriented.

Stepping and tracing external code

If you link external code into your program, you can step over or trace into that code if the .OBJ file you link in contains debugging information.

You can debug external code written in any language, including C, C++, Pascal, and assembly language. As long as the code meets all the requirements for external linking and contains full Borland symbolic debugging information, the IDE's debugger can step or trace through it.

Stepping over large sections of code

Sometimes you don't want to step through each line of your code just to get to the part that is causing problems. With the integrated debugger, you can step over large amounts of code and regain control at the point where you want to begin executing line-by-line again.

Running to a specific location

You can tell the debugger you want to execute your program normally (not step-by-step) until a certain location in your code is reached:

1. Position the cursor at the line where you want to resume debugging control.
2. Choose Run to Cursor on the SpeedMenu (or press *F4).*

You can use Run to Cursor as a way to start your debugging session or after you've already been stepping and tracing.

Locating a function

You can locate a particular function quickly with the Locate Function command on the Search menu. Locate Function asks you for the name of a function, then positions the cursor on the proper line in the file where that function is defined. You must be debugging (stepping or tracing through code) before you can use Locate Function.

Returning to the execution point

While you are debugging, you are free to browse through any file in any edit window, go to any place in your file, and even open and close files. You can then return to the execution-point location very quickly.

To go to the execution-point location, choose Debug | Find Execution Point.

The debugger positions the cursor at the execution point. If you closed the edit window containing the execution point, Find Execution Point opens an edit window and displays the source code containing the execution point.

Navigating backward

While you're debugging, it can be useful to know how you got to where you are. The Call Stack window shows you the sequence of function calls that brought you to your current state. Use View | Call Stack to display the Call Stack window.

The Call Stack window is particularly useful if you accidentally trace into code you want to step over. You can step back, and then resume stepping and tracing where you originally intended:

1. In the Call Stack window, double-click the call that calls the function you traced into by mistake. (It will be the second call from the top of the Call Stack window.)

   The edit window becomes the active window with your cursor positioned at the place the call was made.
2. In the edit window, move the cursor completely past the call.
3. Choose Run to Cursor on the edit window SpeedMenu.

The Call Stack window is also useful when you want to view the arguments passed to each function.

You can view or edit the source code that contains a particular call. Select the call in the Call Stack window and right-click to display the SpeedMenu to display view and edit commands.

Stopping the program

Instead of stepping over or tracing through code, you can use a simpler technique to pause your program. Choose Debug | Pause Program, and your program will stop executing. Then you can examine the value of variables and inspect data at this state of the program. When you're done, choose Debug | Run to continue the execution of your program.

If for some reason your program assumes control and won't allow you to return to the debugger (for example, it might be in an infinite loop), you can press *Ctrl+Alt+Sys Req* to stop your program.

Starting over

While debugging, you might occasionally want to start over from the beginning. Choose the Debug | Terminate Program command or press *Ctrl+F2*. This ends your program so that all subsequent running, stepping, or tracing begins at the start of the main program.

Examining values

Stepping and tracing through your code can help you find problems in program flow, but you'll usually want to watch what happens to the values of variables while you step. For example, when you step through a **for** loop, it's helpful to know the value of the index variable. The IDE has several tools to help you examine the contents of your program's variables:

- The Watch window lets you track the value of a variable or expression.
- The Evaluate Expression dialog box lets you evaluate any expression meaningful to the program you're debugging, and it lets you change the value of a variable while you're debugging your program.
- The Data Inspector window lets you examine and modify the value in a data element.

What's an expression?

Watching, evaluating, and inspecting operate at the level of *expressions*. An expression consists of constants, variables, and data structures combined with operators. Almost anything you can use as the right side of an assignment statement can be used as a debugging expression.

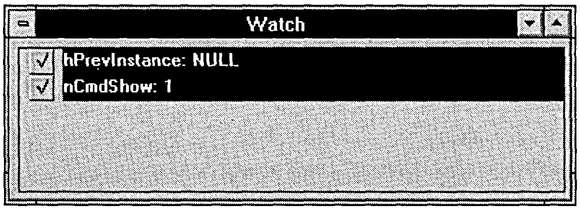
Watching expressions

If you want to keep track of the value of a variable or expression while you step through your code, use a watch. A watch is an expression you enter in the Watch window, which then displays the current value of the expression. As you step through your program, the value of the watch expression changes when the program does something to change it.

If the execution point steps out of the scope of a watch expression, the watch expression is undefined. Once the execution point enters the scope of the expression once again, the Watch displays the current value of the expression once more.

To open the Watch window, choose View | Watch. If you haven't added any watches yet, the window is empty.

Figure 6.1
The Watch window



```
hPrevInstance: NULL
nCmdShow: 1
```

Adding a watch

To add a variable to the Watch window, choose Debug | Add Watch. The IDE opens a Watch Properties dialog box, prompting you to type in a watch expression.

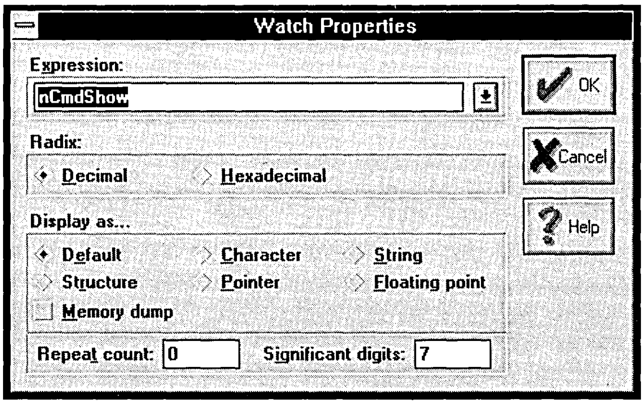
Just like other IDE windows, you can move, resize, and close the Watch window. If the Watch window is closed, you can display it again with View | Watch. If the Watch window is the active window, you can add a watch by choosing Add Watch from the SpeedMenu.

If the edit window is the active window, you can quickly add the expression at your cursor into the Watch window. To do so, choose Set Watch on the SpeedMenu.

Formatting watch expressions

If you double-click a watch in the Watch window, a Watch Properties dialog box appears. You can also display a Watch Properties dialog box by selecting a watch in the Watch window and then choosing Set Properties on the Watch window SpeedMenu.

Figure 6.2
The Watch Properties dialog box



The default expression in a Watch Properties dialog box is the word at the cursor in the current edit window. A history list keeps track of expressions you've entered as watches previously.

You can format watch expression results by selecting options in the Watch Properties dialog box. For example, although integer values normally display in decimal form, you can specify that an expression be displayed as hexadecimal by selecting the Hexadecimal button. After selecting either Decimal or Hexadecimal, you can modify the format of the expression further with the Display As options.

If you're watching a data element such as an array, you can display the values of the consecutive data elements. For example, for an array of five integers named *xarray*, you would type the number 5 in the Repeat Count box of a Watch Properties dialog box to see all five values of the array. An expression used with a repeat count must represent a single data element. The debugger views the data element as the first element of an array if the element isn't a pointer, or as a pointer to an array if it is.

If you select the Floating Point display option, you can also indicate the number of significant digits you want displayed in the watch expression result. Specify the number of digits in the Significant box.

Disabling a watch

If you prefer not to watch an expression you've entered in the Watch window, but you don't want to delete it because you might want to use it later, you can disable the watch.

If the IDE must evaluate many watch expressions in the Watch window, stepping might slow down. You can choose to disable a watch, and then enable it later when you need it. To disable a watch, click the check box next to the watch. To enable it again, click the check box again.

You can also disable and enable watches with SpeedMenu commands. To disable or enable multiple watches at a time, click and drag over a block of watches to select them in the Watch window, or press *Ctrl* while you click all the watches you want to disable or enable. Then choose the appropriate command on the SpeedMenu.

Deleting a watch

To delete a watch expression, select the expression in the Watch window and choose Delete Watch on the SpeedMenu. To delete all watch expressions, choose Delete All Watches on the SpeedMenu.

Editing a watch

To change the properties of a watch, display the Watch Properties dialog box for that watch and edit the properties you want changed.

Evaluating and modifying expressions

In addition to watching variables as your program executes, you can evaluate expressions at any time and you can change the values of variables at run time.

Evaluating expressions

To evaluate an expression, choose Debug | Evaluate/Modify. The debugger displays an Expression Evaluator dialog box. By default, the word at the cursor position in the current edit window is highlighted in the Expressions box. You can edit the expression, type in another, or choose one from the history list of expressions you evaluated previously.

The current value of the expression in the Expression box shows in the Result box when you choose Evaluate. You can evaluate any valid C or C++ expressions except those that contain these things:

- Symbols or macros defined with **#define**
- Local or static variables not in the scope of the function being executed
- Function calls

You can format expressions by adding a comma and one or more format specifiers. For example, to display the result in hexadecimal, type **,H** after the expression. Table 6.1 shows all the legal format specifiers and their effects.

Table 6.1: Format specifiers for debugger expressions

Character	Types affected	Function
H or X	Integers	**Hexadecimal**. Shows integer values in hexadecimal with the 0x prefix, including those in data structures.
C	Characters, strings	**Character.** Shows special display characters for ASCII 0..31. By default, such characters are shown using the appropriate C escape sequences (**\n**, **\t**, and so on).
D	Integers	**Decimal**. Shows integer values in decimal form, including those in data structures.
F*n*	Floating point	**Floating point.** Shows *n* significant digits (where *n* is in the range 2..18, and 7 is the default).
*n*M	All	**Memory dump**. Shows *n* bytes starting at the address of the indicated expression. If *n* is not specified, it defaults to the size in bytes of the type of the variable. By default, each byte shows as two hex digits. The C, D, H, and S specifiers can be used with M to change the byte formatting.
P	Pointers	**Pointer**. Shows pointers as *seg:ofs* with additional information about the address pointed to. It tells you the region of memory in which the segment is located, and the name of the variable at the offset address, if appropriate.
R	Structures, unions	**Structure/Union**. Shows both field names and values such as (*X*:1;*Y*:10;*Z*:5).
S	*Char*, strings	**String.** Shows ASCII 0..31 as C escape sequences. Use only to modify memory dumps (see nM above).

Modifying variables

While debugging, you can change the value of a variable by using the Expression Evaluator dialog box. Enter the variable in the Expression box, then type the new value in the New Value box. If you want the modified value to take effect in your program, choose Modify. Otherwise, when you close the dialog box, the debugger ignores the modified value.

Keep these points in mind when you change the values of variables:

- You can change individual variables or elements of arrays or structures, but not arrays or structures themselves.
- The expression in the New Value box must evaluate to a result that is assignment-compatible with the variable you want to assign it to. A good rule of thumb is that if the assignment would cause a compile-time or run-time error, it's not a legal modification value.

- You can't directly modify untyped arguments passed into a function, but you can typecast them and then assign new values.
- You can use the Expression Evaluator dialog box to examine and modify values in registers, including the flags register. For example, you can enter expressions such as these: _CS, _BX, _FLAGS. These values are bitmasks.

Warning!

- Modifying values, and especially pointer values and array indexes, can have undesirable effects because you might overwrite other variables and data structures. Be careful.

Inspecting data elements

You can examine and modify values in a data element in an Inspector window. To inspect a data element,

1. Choose Debug | Inspect to display the Data Inspector window.
2. Type in the expression you want to inspect.
3. Choose Inspect to display an Inspector window.

If the execution point is in the scope of the expression you are inspecting, the value appears in the Data Inspector window. If the execution point is outside the scope of the expression, the value is undefined.

You can also display an Inspector directly from the edit window:

1. Position your cursor on the data element you want to inspect.
2. Choose Inspect on the SpeedMenu or press *Enter.*

If you choose this method, the data element is always evaluated within the scope of the line on which the data element appears.

With either method, the appearance of the data in an Inspector window depends on the type of data being inspected. For example, if you inspect an array, you'll see a line for each member of the array with the array index of the member. The value of the member follows in its display format, followed by the value in hexadecimal.

Once you're in an Inspector window, you can inspect certain elements to isolate the view. To inspect an item,

1. Select the item you want to inspect further.
2. Choose Inspect on the SpeedMenu or press *Enter.*

You can change the value of inspector items. To change the value of a single inspector item:

1. Select the item.
2. Choose Change on the SpeedMenu.
3. Type in a new value and choose OK.

If you're inspecting a data structure, it's possible the number of items displayed might be so great that you'll have to scroll in the Inspector window to see data you're interested in. For easier viewing, you can narrow the display to a range of data items.

To display a range of items,

1. Click in the Inspector window.
2. Choose Range on the SpeedMenu.
3. In the Start Index box, enter the index of the item you want to see first in the window.
4. In the Count box, enter the number of items you want to see in the Inspector window.

Examining register values

While debugging, you can display the values in the data, pointer, index, segment, and instruction pointer registers, as well as the settings of the status word or flags. Choose View | Register to display the Registers window.

You can also view register values with the Expression Evaluator dialog box. See page 114.

Table 6.2 CPU flags in the Register window

Letter in pane	Flag name
c	Carry
z	Zero
s	Sign
o	Overflow
p	Parity
a	Auxiliary carry
i	Interrupt enable
d	Direction

The Registers SpeedMenu lets you select the format to display the values in the registers; choose from Hexadecimal or Decimal. You can also choose to view the 16-bit (word) or 32-bit (double word) registers.

Using breakpoints

A *breakpoint* is a designated position in the code where you want the program to stop executing and return control to the debugger. A breakpoint is similar to the Run to Cursor command, because the program runs at full speed until it reaches a certain point. But unlike Run to Cursor, you can have multiple breakpoints and you can choose to stop at a breakpoint only under certain conditions.

Setting breakpoints

To set a breakpoint in your code, move the cursor to the line where you want to break. The line needs to contain executable code—if it's a comment, a blank line, or a declaration, the debugger will declare it invalid. To select the line as a breakpoint, choose Toggle Breakpoint (*F5*) on the Debug menu or the edit window SpeedMenu; the line becomes highlighted.

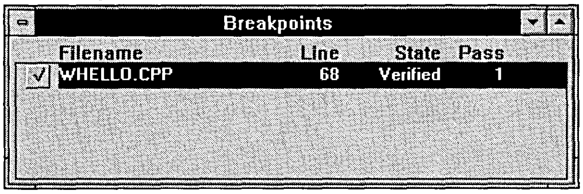
Now when you run your program from the IDE it will stop whenever it reaches that line, but *before* it executes the line. The line containing the breakpoint shows in the edit window with the execution point on it. At that point, you can do any other debugging actions such as stepping, tracing, watching, inspecting, and evaluating.

To delete a breakpoint, move the cursor to the line containing the breakpoint and choose Toggle Breakpoint (*F5*) from the Debug menu or the edit window SpeedMenu.

Working with breakpoints

The IDE keeps track of all your breakpoints during a debugging session and associates them with your current project. You can maintain all your breakpoints from a single window, so you don't have to search through your source code files looking for them. Choose View | Breakpoint to display the Breakpoint window.

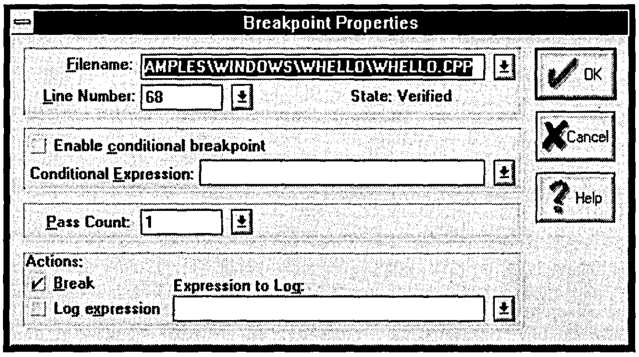
Figure 6.3
The Breakpoint window



Filename	Line	State	Pass
✓ WHELLO.CPP	68	Verified	1

From the Breakpoint window, you can set and view breakpoint properties. Double-click a breakpoint in the Breakpoint window, or select it and select Set Properties from the SpeedMenu. A Breakpoint Properties dialog box appears.

Figure 6.4
The Breakpoint
Properties dialog box



Filename: AMPLES\WINDOWS\WHELLO\WHELLO.CPP
Line Number: 68
State: Verified
Enable conditional breakpoint
Conditional Expression:
Pass Count: 1
Actions:
✓ Break
Expression to Log:
Log expression

Deleting breakpoints

To delete a breakpoint, select it in the Breakpoint window and choose Delete Breakpoint on the SpeedMenu. To delete all breakpoints, choose Delete All on the SpeedMenu.

You can also delete breakpoints with the Delete Breakpoint and Delete All Breakpoints commands on the Breakpoint Properties SpeedMenu.

Disabling and enabling breakpoints

You can disable breakpoints temporarily. The simplest way is to uncheck the check box for the breakpoint you want to disable in the Breakpoint. When you want to enable the breakpoint again, check the breakpoint's check box again.

You can also disable and enable breakpoints with SpeedMenu commands. To disable or enable multiple breakpoints at a time, click and drag over a block of breakpoints to select them in the Breakpoint window, or press *Ctrl* while you click all breakpoints you want to disable or enable. Then choose the appropriate command on the SpeedMenu.

Viewing and editing code at a breakpoint

Even if a breakpoint isn't in your current edit window, you can quickly find a breakpoint in your source code. To do so, select the breakpoint in the Breakpoint window, then choose View Source on the SpeedMenu. The breakpoint appears in the current edit window with your cursor positioned on the breakpoint. The Breakpoint window remains open and is the active window.

If you prefer to *edit* the source code marked as a breakpoint rather than just view it, choose Edit Source on either the Breakpoint window SpeedMenu. The breakpoint appears in the current edit window with your cursor positioned on the breakpoint, ready for you to edit.

Resetting invalid breakpoints

Breakpoints must be set on executable code, or they are invalid. For example, a breakpoint set on a comment, a blank line, or declaration is an invalid breakpoint. If you unintentionally set an invalid breakpoint and run your application, the debugger checks all breakpoints and reports an invalid breakpoint by displaying an Invalid Breakpoint dialog box. Close the dialog box and open the Breakpoints window. Find the invalid breakpoint and delete it; if you want to set the breakpoint in a proper location, do so now. Then continue to run your application.

You can choose to ignore the Invalid Breakpoint dialog box by putting the dialog box away and choosing Run to continue to execute your program. Your application will resume and the IDE will disable the invalid breakpoint.

Changing breakpoint properties

To see and change the properties of a breakpoint, double-click it, or select it in the Breakpoint window and choose Edit Breakpoint on the SpeedMenu. A Breakpoint Properties dialog box appears.

Besides examining a breakpoint's properties, you can use a Breakpoint Properties dialog box to do such things as set a new breakpoint, modify an existing breakpoint, and make a breakpoint conditional.

The next sections explain how to do these things.

Setting a breakpoint

Although using Toggle Breakpoint on the edit window SpeedMenu is the simplest way to set a breakpoint, there are times you might want to set a breakpoint using the Breakpoint Properties dialog box, especially if you want to create a conditional breakpoint:

1. Position the cursor where you want the breakpoint to occur.
2. Choose Add Breakpoint on either the Debug menu or the Breakpoints SpeedMenu.

   A Breakpoint Properties dialog box appears.

You can also set a breakpoint in a file that is not open in an edit window:

1. In the Breakpoint Properties dialog box, type the name of the file you want the breakpoint set in.
2. In the Line Number box, enter the line number of your code where you want the breakpoint to appear.

Modifying an existing breakpoint

You can modify an existing breakpoint by changing its properties in the Breakpoint Properties dialog box. Enter new information for the properties you want to alter.

Setting breakpoints after starting a program

While your program is running, you can switch to the debugger (just like you would switch to any Windows application) and set a breakpoint. When you return to your application, the new breakpoint will be set, and your application will halt when it reaches the breakpoint.

Creating conditional breakpoints

The breakpoints added by the Toggle Breakpoint command on the edit window SpeedMenu are unconditional: any time you get to that line, the debugger stops. When you're editing a new or existing breakpoint, however, you have extra options in the Breakpoint Properties dialog box that let you create *conditional* breakpoints. You can put two kinds of conditions on breakpoints: *Boolean conditions* and *number of passes*.

You can enter a Boolean expression as a condition for a breakpoint. For example, you might test if a variable falls in a certain range, or if a particular flag has been set. When the specified condition is met, the debugger breaks, turning control over to you.

Specifying a number of passes on a breakpoint tells the debugger not to break every time it reaches the specified condition, but instead to break only the *n*th time the specified condition is met. That is, if the pass count is 3, the debugger breaks only the third time the specified condition is met. If you don't specify a condition, the debugger breaks every time the line of code executes.

Logging expressions

You can choose to have the value of a specified expression written in the Event Log window each time a breakpoint is reached:

1. Select Log Expression in the Breakpoint Properties dialog box.
2. In the Expression to Log box, type the expression you want evaluated.

For example, if you type the name of a variable in the Expression to Log box in the Breakpoint Properties dialog box, the debugger writes the value of that expression in the Event Log window when it reaches your set breakpoint.

For more information about the Event Log window, see the "Using the Event Log window" section.

Customizing breakpoints and the execution point

You can use color to indicate enabled, disabled, and invalid breakpoint lines:

1. Choose Options | Environment and choose Syntax Highlighting | Customize Syntax Highlighting.
2. From the Element list, select the Break element you want to change and then change the background and foreground to the colors you want.

   To read more about using syntax highlighting, see page 14.

You can also change the color of the execution point. To do so, select the Execution Point element and then select your desired colors.

Catching general protection faults

If a general protection exception (GP exception) occurs while running your program in the IDE, your program halts and a General Protection Exception dialog box appears. If you choose OK to close the dialog box, the debugger displays the code that is responsible for the GP exception in the edit window. The execution point marks the offending code.

At this point, you must choose Debug | Terminate Program to prevent your program from crashing. Then you can correct the error that caused the problem before you run your program again.

Although the debugger can catch most GP exceptions, it might fail to catch all of them, depending on the cause of the exception.

Using the Event Log window

To display the Event Log window, choose View | Event Log.

In the previous section, you saw how to log expressions you've specified when the debugger reaches a breakpoint to the Event Log window.

You can also log window messages and output messages to the Event Log window. To select the events you want to log, choose Set Options on the Event Log window SpeedMenu. Select the Debugger topic and then select the Event Capture options you want.

The following table describes all the commands on the Event Log window SpeedMenu:

Table 6.3
Event Log window SpeedMenus

Command	Description
Save Events to File	Displays a dialog box so you can specify a file name. All events that are currently in the Event Log window are saved in that file, so you can examine the events later.
Add Comment	Prompts you for a line of text that is inserted in the Log window as a comment.
Clear Events	Clears the Event Log window.
Set Options	Displays Environment Options dialog box, so you can select Debugger options and set the Event Capture options you want.

Debugging dynamic-link libraries

When you step or trace into a DLL your application uses, the debugger automatically loads the DLL's symbol table, which is the list used to track all the variables, constants, types, and function names used in the DLL.

Because only one symbol table can be loaded into memory at a time, your .EXE's symbol table is unloaded when a DLL's symbol table is loaded. Therefore, you won't be able to watch variables, inspect data, and so on in the source code of the .EXE. You will, of course, be able to perform these operations on the DLL code.

To see which symbol table the debugger is using, choose Debug | Load Symbol Table. You'll see a listing of the current symbol table and any others that are available. For example, if your .EXE is named MYAPP.EXE and it uses a MYDLL.DLL, then when you are stepping in the DLL, MYDLL.DLL is the current symbol table and the MYAPP.EXE symbol table is listed as available.

You can switch between symbol tables. For example, if you're stepping in MYDLL.DLL and want to examine the value of a variable that's in MYAPP.EXE, you can do so by loading the MYAPP.EXE symbol table:

1. Choose Debug | Load Symbol Table.
2. Choose the symbol table you want to load from the Available list of symbol tables.

Once you've examined the variable and want to resume stepping, you must switch back to the original symbol table.

Debugging in soft and hard mode

Windows is continually processing and generating messages. Any executing Windows program, such as Borland C++ for Windows, must do the same. When you run your own Windows programs, they too send messages and process messages they receive.

When you set a breakpoint in your code, however, you want your program to stop executing when it reaches the breakpoint. But in the Windows environment, just doing such things as resizing a window, opening a dialog box, moving the mouse cursor, or opening a menu generates Windows messages that are sent to your executing program. Unless the debugger handles these messages for you, the messages could force your program to execute or possibly to lock up your system.

Fortunately, the integrated debugger handles the majority of these situations for you, thus freeing you to debug your program while using the full functionality of the Windows environment. When the debugger works simultaneously with other Windows programs, it is said to be in *soft mode.*

Because of limitations in Windows, you might occasionally need to stop all other Windows programs except the debugger from executing. This state is called *hard mode.* When the debugger is in hard mode, you won't be able to use the functionality of the Windows environment. For example, you won't be able to switch to another task. The debugger behaves as if it's the only program in memory.

You'll want to debug your programs in hard mode if they send intertask messages such as DDE messages and for other very low-level debugging. To change to hard mode,

1. Choose Options | Environment to open the Environment Options dialog box.
2. Choose Debugger.
3. Select the Hard Mode on All Stops option.

When the Smart option is selected, the debugger chooses hard or soft mode, depending on what is happening in the Windows environment. This is the option you'll want to use most frequently. To use the Smart option,

1. Choose Options | Environment.
2. Choose Debugger.
3. Select the Smart option.

C H A P T E R 7

WinSight

WinSight is a debugging tool that gives you information about windows, window classes, and messages. You can use it to study a Windows application, to see how windows and window classes are created and used, and to see what messages the windows receive.

You can configure WinSight to trace and display messages by

- Window
- Window class
- Message type
- A combination of these

WinSight is a passive observer: it intercepts and displays information about messages, but it doesn't prevent messages from getting to applications.

Getting started

To start WinSight, double-click the WinSight icon. The WinSight window appears in its default configuration; it displays the Window Tree view that lists all the windows currently active on the desktop. WinSight saves your configuration, so if you open all three views and exit WinSight, the next time you start it, all three views will display.

WinSight has three views: Class List, Window Tree, and Message Trace. You can display these views from left to right or top to bottom by choosing View | Split Horizontal or View | Split Vertical.

You can control the messages traced by WinSight (see page 127) and you can control when messages start and stop tracing as described in the following sections.

Starting and stopping screen updates

To turn on tracing, choose Start! from the menu (Start! then becomes Stop! on the menu). Normally, all three views are kept current as classes are registered, windows are created and destroyed, and messages are received. However, you can use Messages | Trace Off to suspend tracing of messages only (Class List and Window Tree will continue to update).

Use Stop! and Start! to

- Study a particular situation
- Control (minimize) the machine time WinSight uses when it updates itself constantly.

Turning off message tracing

To turn off tracing of message types, choose Messages | Trace Off. The Message Trace view remains visible, and tracing resumes when you choose Messages | Selected Classes, Selected Windows, or All Windows (provided tracing is on).

The following sections describe how to use the three views to get the information you need to debug an application. Choose Spy | Exit to leave WinSight.

Choosing a view

WinSight has three views that can appear within its main window: Class List, Window Tree, and Message Trace. You can choose to look at any or all of the views. WinSight automatically tiles the views within the main window.

You can hide or display views at any time, using the View menu. Information and selections are not lost when a view is hidden.

- The Class List view shows all the currently registered window classes.
- The Window Tree view displays the hierarchy of all the windows on the desktop. Window Tree displays by default when you start WinSight.
- The Message Trace view displays information about messages received by selected windows or window classes.

To get more detail about an item in Window Tree or Class List,

- Select a window or a class, then choose Spy | Open Detail.
- Double-click the window or class.

The Detail window displays the class name, executable module, and other information about the class or window.

Class List

A class is the name with which the window class was registered with Windows.

Sometimes, instead of choosing specific windows to trace, you might want to look at messages for entire classes of windows. WinSight lets you do this with the Class List view.

Using the Class List view

The Class List view shows all the currently registered window classes. To get details about a class, double-click it or select it and press *Enter.*

The diamonds next to the classes turn black momentarily whenever the window receives any messages. This gives you an overview of which windows are currently receiving messages. If a hidden child window receives a message, the diamond for the parent changes color.

Format `Class (Module) Function Styles`

Class is the name of the class. Some predefined Windows classes have numeric names. For example, the Popup menu class uses the number 32768 as its name. These predefined classes are shown with both the number and a name, such as #32768:PopupMenu. The actual class name is only the number (using the MAKEINTRESOURCE format, which is also used for resource IDs).

Module is the name of the executable module (.EXE or .DLL) that registered the class.

Function is the address of the class window function.

Styles is a list of the cs_ styles for the class. The names are the same as the cs_ definitions in WinTypes, except the cs_ is removed and the name is in mixed case (uppercase and lowercase).

Spying on classes

To trace messages for one or more classes, select the classes in Class List (*Shift-Click* or *Ctrl-Click*), then choose Messages | Selected Classes. If the Message Trace view is hidden, it becomes visible when you choose Messages | Selected Classes.

Note that tracing messages to a class lets you see all messages to windows of that class, including creation messages, which would otherwise not be accessible.

To change which classes are traced, change the selection in the Class List. Choose Messages | Trace Off to turn off all message tracing to the Message view.

Window Tree

The Window Tree view displays a hierarchical outline of all existing windows on the desktop. This display lets you:

- Determine what windows are present on the desktop.

- View the status of windows, including hidden windows.
- See which windows are receiving messages.
- Select windows for message tracing.

The lines on the left of the Window Tree view show the tree structure. Each window is connected to its parent, siblings, and children with these lines. The diamond next to each window shows whether the window has any children. When a window receives a message, the diamond next to it (or its parent window if the tree is collapsed) turns black.

◇ The window has no children.

⊕ The window has children but they aren't displayed. To show the next level of children, click the diamond next to the window. To show *all* the levels of child windows (children of children, and so on), right-click the diamond.

⊖ The window has children displayed (at least one level of child windows is visible; further levels might be collapsed). To hide *all* of a window's child windows, click (or double-click) the diamond next to the window.

Format

```
Handle {Class} Module Position "Title"
```

Handle is the window handle as returned by *CreateWindow*.

Class is the window class name, as described in the Class List view.

Module is the name of the executable module (.EXE or .DLL) that created the window. *Module* is the name of the module owning the data segment passed as the *hInstance* parameter to *CreateWindow*.

Position is "hidden" if the window is hidden. If the window is visible, *Position* is indicated by using screen coordinates (for parent windows) or coordinates in the parent's client area (for child windows). *Position* uses the following format:

```
xBegin,yBegin - xEnd,yEnd
```

Title is the window title or text, as returned by *GetWindowText* or a wm_GETTEXT message. If the title is the null string, the quotation marks are omitted.

Finding a window

WinSight has a special mode for locating windows. It can work in two ways: either identifying the line in the Window Tree that corresponds to a window you point at with the mouse, or highlighting a window you select in the Window Tree.

Important! All other applications are suspended while you're in Find Window mode.

Enter Find Window mode by choosing Spy | Find Window. In this mode, whenever the mouse passes into the boundaries of a window, a thick border appears around that window, and the window is selected in the Window Tree view.

Alternatively, once in Find Window mode, you can select windows in the Window Tree with the mouse or cursor keys, and WinSight will put a thick border around the selected window or windows. If you press *Enter*, you will see the Window Detail window for the selected window.

Leaving Find Window mode

Once you have located the window you want, you can leave Find Window mode by clicking the mouse button or by pressing the *Esc* key. This removes the border from the screen, leaving the current window's description selected in the Window Tree view.

Spying on windows

To spy on one or more windows, select the windows (using the mouse and the *Shift* or *Ctrl* key), then choose Messages | Selected Windows. To change which windows are traced, change the selected window in Window Tree.

To spy on all windows, regardless of what is selected in the Class List or the Window Tree, choose Messages | All Windows.

Message Trace becomes visible when you choose Messages | Selected Windows or Windows | All Windows.

Choose Messages | Trace Off to disable message tracing without hiding Message Trace.

Choosing messages to trace

Message Trace displays messages received by selected window classes or windows. Messages received via *SendMessage* are shown twice, once when they are sent and again when they return to show the return value. Dispatched messages are shown once only, since their return value is meaningless. The message display is indented to show how messages are nested within other messages.

Using the Message Trace view

By default, WinSight traces all messages and displays them in the Message Trace view. WinSight gives you several ways to narrow down the tracing of messages:

- Choose Messages | Selected Classes or Messages | Selected Windows, then select the classes (in the Class List view) or windows (in the Window Tree view) by using the mouse and *Shift* or *Ctrl.*
- Choose Message | All Windows.
- Choose Message | Options, then select any or all of fourteen groups of messages (the groups are described in Tables 7.1 through 7.14). Check All Messages in the Options dialog box to return to tracing all messages.

Other tracing options

The Message Trace Options dialog box lets you change the format of the messages in Message Trace. It also lets you trace messages to a file, printer, or an auxiliary monitor or window.

- Normally, the Message Trace view interprets each message's parameters and displays them in a readable format (Interpret Values is checked). Check Hex Values to view message parameters as hex values of *wParam* and *lParam*.
- Information on traced messages usually displays in the Message Trace view. However, you can send messages to a file, printer, or auxiliary monitor by checking Log File in the Message Trace Options dialog box and doing one of the following:
  - Type a file name to trace to a log file. If the file already exists, messages are appended to the file.
  - Type the name of the device (for example, type PRN) for the log file to send output to the printer port.
  - Type AUX to output trace messages to an auxiliary monitor or window. To do this, you must have WINOX.SYS or OX.SYS installed as a device in your CONFIG.SYS file.

  To stop logging message traces to a file, printer, or auxiliary monitor, uncheck Log File.

Format

```
Handle ["Title" or {Class}] Message Status
```

Handle is the window handle receiving the message.

Title is the window's title. If the title is the null string, the class name is displayed instead, in curly braces.

Message is the message name as defined by Windows. They are displayed in WinSight in all uppercase letters. Known undocumented Windows messages are shown in lowercase. Unknown message numbers (user-defined) are shown as *wm_User+0xXXXX* if they are greater-than or equal to *wm_User* or as *wm_0xXXXX* if they are less than *wm_User*. Registered message numbers (from *RegisterWindowsMessage*) are shown with their registered name in single quotes.

Status is one or more of the following:

- *Dispatched* indicates the message was received via *DispatchMessage*.
- *Sent [from XXXX]* indicates the message was received via *SendMessage*. If it was sent from another window, `from XXXX` gives that window's handle. If it was sent from the same window receiving it, this is shown with `from self`. If it was sent from Windows itself, the "from" phrase is omitted.
- *Returns* indicates the message was received via *SendMessage* and is now returning.
- Additional messages might include a numeric return value or text message such as *wm_GetText*. For sent and dispatched messages, WinSight interprets the parameters and gives a readable display. For messages that have associated data structures (*wm_Create*, for example) it takes those structures and includes them in the display.

Table 7.1: Mouse messages

WM_HSCROLL	WM_MBUTTONUP	WM_RBUTTONDOWN
WM_LBUTTONDBLCLK	WM_MOUSEACTIVATE	WM_RBUTTONUP
WM_LBUTTONDOWN	WM_MOUSEFIRST	WM_SETCURSOR
WM_LBUTTONUP	WM_MOUSELAST	WM_VSCROLL
WM_MBUTTONDBLCLK	WM_MOUSEMOVE	
WM_MBUTTONDOWN	WM_RBUTTONDBLCLK	

Table 7.2: Window messages

WM_ACTIVATE	WM_GETDLGCODE	WM_QUERYNEWPALETTE
WM_ACTIVATEAPP	WM_GETFONT	WM_QUERYOPEN
WM_CANCELMODE	WM_GETMINMAXINFO	WM_QUIT
WM_CHILDACTIVATE	WM_GETTEXT	WM_SETFOCUS
WM_CLOSE	WM_GETTEXTLENGTH	WM_SETFONT
WM_CREATE	WM_ICONERASEBKGND	WM_SETREDRAW
WM_CTLCOLOR	WM_KILLFOCUS	WM_SETTEXT
WM_DDE_FIRST	WM_MOVE	WM_SHOWWINDOW
WM_DESTROY	WM_PAINT	WM_SIZE
WM_ENABLE	WM_PAINTICON	WM_WINDOWPOSCHANGED
WM_ENDSESSION	WM_QUERYDRAGICON	WM_WINDOWPOSCHANGING
WM_ERASEBKGND	WM_QUERYENDSESSION	

Table 7.3: Input messages

WM_CHAR	WM_KEYUP	WM_SYSKEYDOWN
WM_CHARTOITEM	WM_MENUCHAR	WM_SYSKEYUP
WM_COMMAND	WM_MENUSELECT	WM_TIMER
WM_DEADCHAR	WM_PARENTNOTIFY	WM_VKEYTOITEM
WM_KEYDOWN	WM_SYSCHAR	
WM_KEYLAST	WM_SYSDEADCHAR	

Table 7.4: System messages

WM_COMPACTING	WM_PALETTECHANGED	WM_SYSCOLORCHANGE
WM_DEVMODECHANGE	WM_PALETTEISCHANGING	WM_SYSCOMMAND
WM_ENTERIDLE	WM_POWER	WM_TIMECHANGE
WM_FONTCHANGE	WM_QUEUESYNCH	WM_WININICHANGE
WM_NULL	WM_SPOOLERSTATUS	

Table 7.5: Initialization messages

WM_INITDIALOG	WM_INITMENU	WM_INITMENUPOPUP

Table 7.6: Clipboard messages

WM_ASKCBFORMATNAME	WM_DESTROYCLIPBOARD	WM_RENDERALLFORMATS
WM_CHANGECBCHAIN	WM_DRAWCLIPBOARD	WM_RENDERFORMAT
WM_CLEAR	WM_HSCROLLCLIPBOARD	WM_SIZECLIPBOARD
WM_COPY	WM_PAINTCLIPBOARD	WM_UNDO
WM_CUT	WM_PASTE	WM_VSCROLLCLIPBOARD

Table 7.7: DDE messages

WM_DDE_ACK	WM_DDE_EXECUTE	WM_DDE_REQUEST
WM_DDE_ADVISE	WM_DDE_INITIATE	WM_DDE_TERMINATE
WM_DDE_DATA	WM_DDE_POKE	WM_DDE_UNADVISE

Table 7.8: Nonclient messages

WM_NCACTIVATE	WM_NCLBUTTONDOWN	WM_NCPAINT
WM_NCCALCSIZE	WM_NCLBUTTONUP	WM_NCRBUTTONDBLCLK
WM_NCCREATE	WM_NCMBUTTONDBLCLK	WM_NCRBUTTONDOWN
WM_NCDESTROY	WM_NCMBUTTONDOWN	WM_NCRBUTTONUP
WM_NCHITTEST	WM_NCMBUTTONUP	
WM_NCLBUTTONDBLCLK	WM_NCMOUSEMOVE	

Table 7.9: Print messages

DM_COLOR	DM_MODIFY	DM_PRINTQUALITY
DM_COPIES	DM_ORIENTATION	DM_PROMPT
DM_COPY	DM_OUT_BUFFER	DM_SCALE
DM_DEFAULTSOURCE	DM_OUT_DEFAULT	DM_SPECVERSION
DM_DUPLEX	DM_PAPERLENGTH	DM_TTOPTION
DM_IN_BUFFER	DM_PAPERSIZE	DM_UPDATE
DM_IN_PROMPT	DM_PAPERWIDTH	DM_YRESOLUTION

Table 7.10: Control messages

BM_GETCHECK	CBN_SELCHANGE	EN_VSCROLL
BM_GETSTATE	CBN_SELENDCANCEL	LB_ADDSTRING
BM_SETCHECK	CBN_SETFOCUS	LB_DELETESTRING
BM_SETSTATE	DM_GETDEFID	LB_DIR
BM_SETSTYLE	DM_SETDEFID	LB_FINDSTRING
BN_CLICKED	EM_CANUNDO	LB_FINDSTRINGEXACT
BN_DISABLE	EM_EMPTYUNDOBUFFER	LB_GETCARETINDEX
BN_DOUBLECLICKED	EM_FMTLINES	LB_GETCOUNT
BN_HILITE	EM_GETFIRSTVISIBLELINE	LB_GETCURSEL
BN_PAINT	EM_GETHANDLE	LB_GETHORIZONTALEXTENT
BN_UNHILITE	EM_GETLINE	LB_GETITEMDATA
CB_ADDSTRING	EM_GETLINECOUNT	LB_GETITEMHEIGHT
CB_DELETESTRING	EM_GETMODIFY	LB_GETITEMRECT
CB_DIR	EM_GETPASSWORDCHAR	LB_GETSEL
CB_FINDSTRING	EM_GETRECT	LB_GETSELCOUNT
CB_FINDSTRINGEXACT	EM_GETSEL	LB_GETSELITEMS
CB_GETCOUNT	EM_GETTHUMB	LB_GETTEXT
CB_GETCURSEL	EM_GETWORDBREAKPROC	LB_GETTEXTLEN
CB_GETDROPPEDCONTROLRECT	EM_LIMITTEXT	LB_GETTOPINDEX
CB_GETDROPPEDSTATE	EM_LINEFROMCHAR	LB_INSERTSTRING
CB_GETEDITSEL	EM_LINEINDEX	LB_MSGMAX
CB_GETEXTENDEDUI	EM_LINELENGTH	LB_RESETCONTENT
CB_GETITEMDATA	EM_LINESCROLL	LB_SELECTSTRING
CB_GETITEMHEIGHT	EM_MSGMAX	LB_SELITEMRANGE
CB_GETLBTEXT	EM_REPLACESEL	LB_SETCARETINDEX
CB_GETLBTEXTLEN	EM_SCROLL	LB_SETCOLUMNWIDTH
CB_INSERTSTRING	EM_SETFONT	LB_SETCURSEL
CB_LIMITTEXT	EM_SETHANDLE	LB_SETHORIZONTALEXTENT
CB_MSGMAX	EM_SETMODIFY	LB_SETITEMDATA
CB_RESETCONTENT	EM_SETPASSWORDCHAR	LB_SETITEMHEIGHT
CB_SELECTSTRING	EM_SETRECT	LB_SETSEL
CB_SETCURSEL	EM_SETRECTNP	LB_SETTABSTOPS
CB_SETEDITSEL	EM_SETSEL	LB_SETTOPINDEX
CB_SETITEMDATA	EM_SETTABSTOPS	LBN_DBLCLK
CB_SETITEMHEIGHT	EM_SETWORDBREAK	LBN_ERRSPACE
CB_SHOWDROPDOWN	EM_UNDO	LBN_KILLFOCUS
CBN_CLOSEUP	EN_CHANGE	LBN_SELCANCEL
CBN_DBLCLK	EN_ERRSPACE	LBN_SELCHANGE
CBN_DROPDOWN	EN_HSCROLL	LBN_SETFOCUS
CBN_EDITCHANGE	EN_KILLFOCUS	STM_GETICON
CBN_EDITUPDATE	EN_MAXTEXT	STM_SETICON
CBN_ERRSPACE	EN_SETFOCUS	
CBN_KILLFOCUS	EN_UPDATE	

Table 7.11: Pen messages

WIN_USER	WM_HOOKRCRESULT	WM_RCRESULT
WM_GLOBALRCCHANGE	WM_PENWINFIRST	WM_SKB
WM_HEDITCTL	WM_PENWINLAST	

Table 7.12: Multimedia messages

MM_ADLIB	MM_MIM_CLOSE	MM_SNDBLST_MIDIIN
MM_JOY1BUTTONDOWN	MM_MIM_DATA	MM_SNDBLST_MIDIOUT
MM_JOY1BUTTONUP	MM_MIM_ERROR	MM_SNDBLST_SYNTH
MM_JOY1MOVE	MM_MIM_LONGDATA	MM_SNDBLST_WAVEIN
MM_JOY1ZMOVE	MM_MIM_LONGERROR	MM_SNDBLST_WAVEOUT
MM_JOY2BUTTONDOWN	MM_MIM_OPEN	MM_WAVE_MAPPER
MM_JOY2BUTTONUP	MM_MOM_CLOSE	MM_WIM_CLOSE
MM_JOY2MOVE	MM_MOM_DONE	MM_WIM_DATA
MM_JOY2ZMOVE	MM_MOM_OPEN	MM_WIM_OPEN
MM_MCINOTIFY	MM_MPU401_MIDIIN	MM_WOM_CLOSE
MM_MICROSOFT	MM_MPU401_MIDIOUT	MM_WOM_DONE
MM_MIDI_MAPPER	MM_PC_JOYSTICK	MM_WOM_OPEN

Table 7.13: Other messages

WM_COALESCE_FIRST	WM_MDIACTIVATE	WM_MDIRESTORE
WM_COALESCE_LAST	WM_MDICASCADE	WM_MDISETMENU
WM_COMMNOTIFY	WM_MDICREATE	WM_MDITILE
WM_COMPAREITEM	WM_MDIDESTROY	WM_MEASUREITEM
WM_DELETEITEM	WM_MDIGETACTIVE	WM_NEXTDLGCTL
WM_DRAWITEM	WM_MDIICONARRANGE	WM_SYSTEMERROR
WM_DROPFILES	WM_MDIMAXIMIZE	
WM_KEYFIRST	WM_MDINEXT	

Table 7.14: Messages not documented by Microsoft

WM_ALTTABACTIVE	WM_ENTERSIZEMOVE	WM_QUERYPARKICON
WM_BEGINDRAG	WM_EXITMENULOOP	WM_SETHOTKEY
WM_CONVERTREQUEST	WM_EXITSIZEMOVE	WM_SETVISIBLE
WM_CONVERTRESULT	WM_FILESYSCHANGE	WM_SIZEWAIT
WM_DRAGLOOP	WM_GETHOTKEY	WM_SYNCPAINT
WM_DRAGMOVE	WM_ISACTIVEICON	WM_SYNCTASK
WM_DRAGSELECT	WM_LBTRACKPOINT	WM_SYSTIMER
WM_DROPOBJECT	WM_NEXTMENU	WM_TESTING
WM_ENTERMENULOOP	WM_QUERYDROPOBJECT	WM_YOMICHAR

C H A P T E R 8

WinSpector

WinSpector and its utilities help you perform a postmortem examination of Unrecoverable Application Errors (UAEs) and General Protection Faults (GPFs). When a UAE or GPF occurs, WinSpector writes a log file to your disk that shows you helpful information about the cause of the exception, including

- The call stack that was active when an exception occurred
- Function and procedure names in the call stack
- CPU registers
- A disassembly of the machine instructions where the exception occurred
- Windows information about the program environment

Using WinSpector

Before using WinSpector, be sure that TOOLHELP.DLL (from Windows 3.1 or later) is in your search path (TOOLHELP.DLL ships with Borland C++). TOOLHELP.DLL is a Windows DLL that lets utilities access low-level system information. WinSpector uses TOOLHELP.DLL to log exceptions and to obtain the system information it writes to the log file. Don't use other exception debugging tools, except for Turbo Debugger, while running with WinSpector.

There are three ways to start WinSpector (it loads minimized):

WinSpector

- Include it in the "load=" section of your WIN.INI file.
- Include it in the Startup folder in Windows.
- Double-click the WinSpector icon to run WinSpector after you load Windows.

When an exception (UAE or GPF) occurs, WinSpector creates a *report* in a file called WINSPCTR.LOG (a text file) with information to help you determine what caused the error. WinSpector also creates a file called WINSPCTR.BIN, a binary file that the DFA utility translates into a text file called DFA.OUT (see page 139 for more information on DFA.EXE).

After the exception, WinSpector displays a dialog box with a brief exception report. Click OK to remove the box and read the log file to find the cause of the exception. You can control the output to WINSPCTR.LOG as described in the following section.

Configuring WINSPCTR.LOG

There are two ways you can set the WinSpector options that control the output to WINSPCTR.LOG:

- To use the WinSpector Preferences dialog box, start WinSpector, click the WinSpector icon and choose Preferences from the popup menu.
- To edit commands in WINSPCTR.INI, load the file into any text editor, edit or add commands, then save the file and restart WinSpector.

The following sections describe each option in the Preferences dialog box. WINSPCTR.INI options are listed to the left.

LogDir=[directory]

- Directory is the location of WINSPCTR.LOG. Type the path where you want the file (C:\WINDOWS is the default).

LogViewer=[viewername]

- Viewer is the program WinSpector uses to display the log file. Type the path and file name of the viewer you want to use (NOTEPAD.EXE is the default). For example, C:\WIN31\WRITE.EXE. If WinSpector can't find the editor, it displays the message

  ```
  Error: Unable to execute: [option]
  ```

  where *option* is the editor file name. Check to make sure the editor you indicated exists in the specified directory.

CreateNewLog= 0 (append) or 1 (overwrite)

- Append New Reports and Overwrite Previous Reports lets you control whether WinSpector appends reports to the existing log file or overwrites the old log file when a new report is generated.

ShowSystemInfo= 0 (omit) or 1 (show)

- Check System Information to add the Task List, the Module List, and information about the USER and GDI heaps to the log file.

LogToStdAux=0 (on) or 1 (off)

- Check AUX Summary to view an abbreviated form of the information sent to the log file on the AUX device. To use this option, you need a terminal connected to AUX or a device driver, such as OX.SYS, that redirects the AUX device to a second monitor.

PostMortemDump= 1 (show) or 0 (omit)

- Check PostMortem Dump to generate a WINSPCTR.BIN file. Use DFA.EXE to translate the BIN file into a text file you can read.

ShowStackInfo= 1 (show) or 0 (omit)

- Check Stack Frame Data to add a verbose stack trace display to the log file. For each stack frame that doesn't exceed 256 bytes, WinSpector performs a hex dump, starting at the SS:BP for that frame. If there are more than 256 bytes between two successive stack frames, the memory display is omitted for that frame. You can use this data to get the values of the parameters that were passed to the function.

It is usually easier to let DFA do the hard work of figuring out what your parameters are. However, for those cases where Turbo Debugger information is not available, you might find that a verbose trace supplies helpful information.

ShowUserInfo= 1 (show) or 0 (omit)

- Check User Comments if you want to add information to the log file about what was happening when the exception occurred. With User Comments checked, WinSpector displays a dialog box immediately after the exception. The comments you type are appended to the log file.

WINSPCTR.LOG reference

Each report in WINSPCTR.LOG has several sections that help you determine what caused the exception in your program. The first line of a report in WINSPCTR.LOG gives the date and time when the exception occurred; for example,

```
WinSpector failure report - 6/18/1992 11:04:25
```

The second line lists

- What type of exception occurred (Table 8.1 lists frequent exceptions)
- The module name
- The logical address
- The physical address
- The currently active task at the time of the exception

A second line might look like this:

```
Exception 13 at USER 002A:0429 (079F:0429)  (TASK=BAD)
```

Table 8.1: Exception types

Number	Name	Description
0	Division by zero	Occurs during a DIV or an IDIV interaction if the divisor is 0.
12	Stack fault	Usually occurs when there is not enough room on the stack to proceed.
13	General protection fault (GPF)	All protection errors that don't cause another exception cause an exception 13.

Exception 13 errors include, but are not limited to, the following errors:

- Invalid selector loaded into a segment register.
- Segment limit exceeded. Although the selector is valid, the offset value is greater than the segment limit (for example, an array index out of bounds error in DS, ES, or other segments).
- Execution is transferred to a nonexecutable segment, such as a bad function pointer.

- Accessing DS, ES, FS, or GS registers containing a null selector. (This error can cause a 0 to appear in the segment register of the log file.)

A log file lists both the physical and logical addresses where the exception occurred. These two types of addresses are important to Windows programs for the following reasons:

- When a program is loaded, Windows allocates space for each logical segment and assigns each segment a unique selector. The selector and its offset are combined to form a physical address.
- When a Windows .EXE file is linked, each segment is placed in a different section of the file, and a segment table is created.
- A logical address, which is actually a segment's position in the Windows segment table, consists of a module name, a logical segment, and an offset. You can run TDUMP on the file to find out segment size and other information, or you can generate a .MAP file that contains the same kind of information.

If the stack pointer is too small at the time of exception, TOOLHELP.DLL automatically switches the stack and appends the message `Stack Switched` to the end of the second line of the log.

Disassembly section

The Disassembly section in WINSPCTR.LOG begins with the assembly language instruction that caused the exception that is followed by the next few instructions in the program, which provide a point of reference for finding the task that caused the exception.

For example, given the following code, where ES is the segment register that contains a selector and BX is the offset into the segment, an exception 13 occurred because the value in BX was greater than the segment limit referenced by ES:

```
079F:0429   CMP    BYTE  PTR  ES:[BX],FF
079F:042D   JNE    043A
079F:042F   CMP    WORD  PTR  [BP+06],03
079F:0435   MOV    DI, 0001
```

Stack Trace section

The first line of the Stack Trace section in WINSPCTR.LOG identifies the function or procedure that was executing at the time of the exception. Stack Trace information includes the

- Frame number
- Module name

- Name of the closest function before the address of the one that caused the exception, plus a number indicating how far away you were from that function. (This information is present only if a .SYM file is present.)
- Logical and physical address for the stack frame
- Location where your program returns after the call

When WinSpector lists function names, it looks in the .SYM file for the closest symbol name that appears before the address in the call stack. Since some .SYM files do not contain information for all functions, the function name in the log file is the closest function in the .SYM file with an address preceding the frame address. If the offset field appears to be too high, function names might not be reliable.

The following stack trace information shows some of the functions that were executing at the time BAD, a sample task, caused an exception:

```
Stack Trace:
0  User  <no info>
   CS:IP 002A:0429  (079F:0429)     SS:BP  10FF:18CA
   C:\WIN31\SYSTEM\USER.EXE
   :
3  BAD  function5(unsigned long, unsigned long, unsigned long) + 0014
   CS:IP 0001:0184 (1107:0184)      SS:BP  10FF:1952
   C:\BIN\BAD.EXE
   :
```

Register section

The Register section in WINSPCTR.LOG lists the values stored in the standard registers when the exception occurred, as the following example shows:

```
Registers:
AX     0037
BX     0000
CX     0008
DX     10EE
SI     0037
DI     0028
```

Limits and access rights are given for the CS, DS, ES, and SS registers.

Message Queue section

The Message Queue section in WINSPCTR.LOG gives the last message received in the middle of processing. This section also lists any messages that were waiting in the queue at the time of exception. For each message, WinSpector lists the following information:

The Message Queue section might not list the last message the program received: Windows could bypass the message queue by using a *SendMessage* or similar function.

- The Window handle that identifies the destination window
- The Message ID number that identifies the message
- Two parameters that contain additional message information

The following Message Queue example shows one message received and one waiting in the queue:

```
Message Queue:
Last message received:
   hWnd: 0000   msg: 0001   wParam: 0002   lParam: 00000003
Waiting in queue:
   hWnd: 0000   msg: 0001   wParam: 0002   lParam: 00000003
```

Tasks section

The Tasks section in WINSPCTR.LOG lists the programs running when the exception occurred, including the

- Complete path to the executable file
- Module name
- Windows module handle
- Task handle
- Data segment value for the task (the instance handle)

Some of the tasks running when the BAD application caused an exception include

```
C:\WIN31\SYSTEM\NWPOPUP.EXE
   Module: NWPOPUP    hModule: 142F   hTask: 141F   hInstance: 13F6
   :
C:\BIN\WINSPCTR.EXE
   Module: WINSPCTR   hModule: 1397   hTask: 1387   hInstance: 135E
   :
C:\BIN\BAD.EXE
   Module: BAD       hModule: 1467   hTask: 1127   hInstance: 10FE
```

Modules section

The Modules section in WINSPCTR.LOG lists the modules that were running at the time of the exception, including the

- Path to the executable file
- Date stamp of the executable file
- File size
- Module name
- Module handle
- Reference count indicating how many times the module is in use

Three of the modules running when the BAD application caused an exception include

```
C:\WIN31\SYSTEM\KRNL386.EXE     Date: 03/02/1992   Size: 116132
   Module: KERNEL   hModule: 010F  reference count: 21
C:\WIN31\SYSTEM\SYSTEM.DRV      Date: 03/01/1992   Size: 2304
   Module: SYSTEM   hModule: 013F  reference count: 13
   ⋮
C:\C\BIN\WINSPCTR.EXE           Date: 06/02/1992   Size: 46256
   Module: WINSPCTR  hModule: 1397 reference count: 1
   ⋮
```

USER and GDI heap section

The USER and GDI (graphics device interface) heap information section in WINSPCTR.LOG shows what percentage of the USER and GDI heaps was available at the time of exception. For example,

```
USER     Free   91%
GDI      Free   83%
```

Because Windows has only 64K of internal heap space for applications to share, it's often helpful to keep track of how the space is used. If you find that USER and GDI are taking up a lot of heap space, check to see if you have deallocated resources you are not using. The Help | About box for Program Manager lists the lower of these values as the amount of free System Resources.

System Information section

The System Information section in WINSPCTR.LOG shows the windows version and mode you're running, including

- CPU type
- Largest free block of contiguous linear memory in the system
- Total linear address space in pages
- Amount of free memory pages in the linear address space
- Number of pages in the system swap file

The System Information section for a 486 system might look like this:

```
System info: Running in enhanced mode under Windows 3.1 debug version
CPU: 80486
Largest free memory block: 3457024 bytes
Total linear memory space: 19696 K
Free linear memory space : 18212 K
Swap file Pages:           0 (0 K)
```

Processing WinSpector data

DFA is a utility that takes a WINSPCTR.BIN file and Turbo Debugger information (either in the .EXE, .DLL or .TDS files) and translates the binary data into a useful form by generating a file that contains not only stack trace information similar to the log file but also function names, line numbers, and local and global variables.

DFA post-processes Turbo Debugger information that WinSpector gathered at the time of the exception. If you check the PostMortem dump option (see page 134), WinSpector creates a WINSPCTR.BIN file at the time of the exception. You can use DFA.EXE to translate the binary data in WINSPCTR.BIN into usable information stored in a text file called DFA.OUT.

Because only one WINSPCTR.BIN file is written per Windows session, make sure you run DFA promptly. For example, if you get three UAEs in succession, WinSpector will write three reports to the log file, but binary data will exist for only the first report. It's best to run DFA immediately after receiving the first UAE. You might then want to rename the DFA.OUT file and delete the WINSPCTR.BIN and WINSPCTR.LOG files before continuing.

DFA output

DFA writes a file only if Turbo Debugger information exists for the file in the stack frame. The DFA output file (DFA.OUT) has a stack trace similar to the one in the WinSpector log file, except that it contains

- Function names
- Line numbers
- Local and global variables
- Data segments and their values (including the stack segment)

Using DFA with WINSPCTR.LOG

When DFA is used with the WINSPCTR.LOG file alone, it gives minimal stack trace information, such as addresses. If Turbo Debugger information (contained in a .EXE, .DLL, or .TDS file) is present, source file names and line numbers are added to the report.

Using DFA with WINSPCTR.BIN

When used with the WINSPCTR.BIN file, DFA

- Adds Stack-based variables to the log, including local variables, parameters passed to the function, structures and arrays.
- Lists variable types, values, and addresses by function.

If Turbo Debugger information is present, for each stack frame, DFA reports

- In section one, the
  - Source file
  - Line number
  - Local variables
  - Parameters

- In section two, the
  - Module name for the task with the fault
  - File names
  - Logical segments
  - The segments' selectors
  - Whether the segments are data or code segments
- In section three, the
  - Global variables
  - Static variables
  - The variables' values at the time of the exception

Format

```
DFA [option] WINSPCTR.LOG [WINSPCTR.BIN]
```

When WINSPCTR.LOG (required) is present, you get source file and line numbers. When WINSPCTR.BIN (optional) is present, you get additional variable information.

Table 8.2
DFA options

Option	What it does
/O[outputfile]	Renames the output file from the DFA.OUT default
/D	Forces DFA to write a hex dump of the saved data segments

Other WinSpector tools

WinSpector has three utilities you can use to enhance the information about an exception:

- EXEMAP.EXE creates a .MAP file from a Windows .EXE file. The .MAP file is needed to create a .SYM file, which expands error reporting for the original .EXE.
- TMAPSYM.EXE, used in conjunction with EXEMAP.EXE, creates a .SYM file from a .MAP file.
- BUILDSYM.EXE uses EXEMAP.EXE and TMAPSYM.EXE to create a .SYM file from a Windows .EXE file.

Using EXEMAP.EXE

EXEMAP creates .MAP files for Windows executables. A .MAP file can be used to create a .SYM file, which can then be used by WinSpector to expand its error reporting. If you are using .DLLs or other programs for which you don't have the source code, this information can be especially useful.

To create a .MAP file from an .EXE, type `EXEMAP filename.EXE newname.MAP`. If you don't type a new name, EXEMAP creates a .MAP file with the same name as the .EXE.

Although the resulting .MAP file isn't as complete as one generated by the link phase of the compile process, it does include addresses for exported public functions.

Using TMAPSYM.EXE

TMAPSYM creates .SYM files from existing .MAP files (created either by the compiler or by the EXEMAP utility). The resulting .SYM files make available to WinSpector the public functions, variable names, and functions in the entry table of the executable. Constants and line-number information, however, are not included in a TMAPSYM-generated .SYM file.

To create a .SYM file from a .MAP file, type `TMAPSYM filename.MAP` (you must type the .MAP extension).

Using BUILDSYM.EXE

BUILDSYM creates .SYM files from .EXE files. It has the same output as using both EXEMAP and TMAPSYM, since it automatically runs them, but it deletes the .MAP files from your directory. BUILDSYM supports wildcards, so you can create .SYM files for part or all of a directory by entering a single command.

To run BUILDSYM, both EXEMAP and TMAPSYM must be in the same directory as BUILDSYM or in your search path. BUILDSYM places the .SYM files it creates in the current directory. For WinSpector to find a .SYM file, the file must be in the same directory as the executable that caused the exception.

BUILDSYM performs the following tasks:

- Verifies that the files are Windows files, and if not, leaves them alone.
- Calls EXEMAP to create .MAP files.
- Verifies that .MAP files were created.
- Calls TMAPSYM and passes the names of the new .MAP files so TMAPSYM can create .SYM files.
- Deletes the .MAP files because they are no longer needed.

To create a .SYM file from an .EXE, type `BUILDSYM filename.EXE`.

Conveniently, you can use DOS wildcards in the *filename* portion of the syntax. For example, type `BUILDSYM *.EXE` to create .SYM files for all the .EXE files in the current directory.

C H A P T E R 9

Using the linker: TLINK

Appendix A, "Error messages," lists linker messages generated by TLINK and by the built-in IDE linker.

TLINK and TLINK32 are command-line tools that combine object modules (.OBJ files) and library modules (.LIB files) to produce executable files. The IDE uses built-in versions of the linkers. Because the compiler automatically calls the linker, you don't need to use TLINK *unless* you suppress the linking stage of compiling (see the **–c** compiler option). Unless otherwise specified, instructions and options for TLINK also apply to TLINK32.

TLINK basics

TLINK and TLINK32 options are case-sensitive.

TLINK uses a configuration file called TLINK.CFG, a response file (optional), and command-line options to link object modules, libraries, and resources into an executable file (.EXE or .DLL). The IDE linker uses the options specified in the Project Options dialog box in the Linker section. The syntax for TLINK is

```
TLINK [@respfile][options] startupfile myobjs, exename, [mapfile],
[mylibs] runtimelib [import], [deffile], [resfiles]
```

where

- `options` are TLINK options that control how TLINK works. For example, options specify whether to produce an .EXE or a DLL file. TLINK options must be preceded by either a slash (/) or a hyphen (-). To turn off a default option, place a hyphen after the option (for example, –P-). Table 9.3 lists the TLINK options.

See Tables 9.1 and 9.2 to determine which module to use.

- `startupfile` is a Borland initialization module for executables or DLLs that arranges the order of the various segments of the program. The initialization module must appear first in the object file list. If it isn't first, the program segments might not be placed in memory properly, which could cause some frustrating program bugs. Failure to link the correct initialization module usually results in a long list of error messages telling you that certain identifiers are unresolved, or that no stack has been created.

- `myobjs` are the .OBJ files you want linked. Specify the path if the files aren't in the current directory.
- `exename` is the name you want given to the executable file (.EXE or .DLL). If you don't specify an executable file name, TLINK derives the name of the executable by appending .EXE or .DLL to the first object file name listed. Be sure you give an explicit name for the executable file name on the TLINK command line. Otherwise, your program name will be something like C02.EXE—which probably isn't what you wanted.
- `mapfile` (optional) is the name you want given to the map file. If you don't specify a name, the map file name is given the same as *exefile* (but with the .MAP extension).
- `mylibs` (optional) are the library files you want included at link time. If these files aren't in the current directory or the search path (see the **/L** option) then you must include their paths.
- `runtimelib` is the Borland run-time library. If no libraries are included, none are linked.
- `importlib` is the Windows import library, which provides access to the Microsoft Windows API functions.
- `deffile` is the module-definition file (.DEF) for a Windows executable. If you don't specify a .DEF file, TLINK creates an application based on default settings.
- `resfiles` are a list of .RES files to bind to the executable.

TLINK assumes or appends these extensions to file names that have none:

- .OBJ for object files
- .EXE for executable files
- .DLL for dynamic-link libraries
- .MAP for map files
- .LIB for library files
- .DEF for module-definition files
- .RES for resource files

TLINK.CFG

The IDE uses linker options specified in project options and style sheets. See Chapter 2 for more information on setting options for project.

TLINK uses a configuration file called TLINK.CFG (or TLINK32.CFG) for options that you'd normally type at the command-line (note that TLINK.CFG can only be options, not file names). Configuration files let you save options you use frequently, so that you don't have to continually retype them.

TLINK looks for TLINK.CFG in the current directory, then in the directory from which TLINK was loaded.

The following TLINK.CFG file tells TLINK to look for libraries first in the directory C:\BC4\LIB and then in C:\WINAPPS\LIB, to include debug information in the executables it creates, to create a detailed segment map, and to produce a Windows executable (.EXE not .DLL).

TLINK	TLINK32
`/Lc:\bc4\lib;c:\winapps\lib`	`/Lc:\bc4\lib;c:\winapps\lib`
`/v /s`	`/v /s`
`/Twe`	`/Tpe`

Response files

The command-line compilers also use response files. See page 45 for more information.

Response files are ASCII files of options and file names for TLINK.EXE (and TLINK32.EXE) that you would normally type at the command line. Response files let you have a longer command line than most operating systems allow. Response files can include the same information as configuration files (command-line options), but they can also contain file names.

Unlike the command line, a response file can be several lines long. To use more than one line in your response file, end each line with a plus character (+). Note that if a line ends with an option that uses the plus to turn it on (such as **/v+**), the **+** isn't treated as a line continuation character (to continue the line, use **/v++**).

If you separate command-line components (such as .OBJ files from .LIB files) by lines in a response file, you must leave out the comma used to separate them on the command line. For example,

```
/c c0ws+
   myprog,myexe
   mymap
   mylib cws
```

leaves out the commas you'd have to type if you put the information on the command line:

```
TLINK /c c0ws myprog,myexe,mymap,mylib cws
```

To use response files,

1. Type the command-line options and file names into an ASCII text file and save the file.
2. Type `TLINK @[path]RESFILE.RSP`, where `RESFILE.RSP` is the name of your response file.

You can specify more than one response file as follows:

```
tlink /c @listobjs,myexe,mymap,@listlibs
```

If you use a response file in addition to command-line options, the command-line options will override any options in the response file. For example, if you include **–v** in a response file, but you use **–v-** at the command-line, TLINK uses the command-line option **–v-**.

Using TLINK with BCC.EXE

BCC always starts TLINK with the **/c** (case-sensitive link) option.

You can pass options and files to TLINK through the command-line compilers (BCC.EXE and BCC32.EXE) by typing file names on the command line with explicit .OBJ and .LIB extensions. For example,

```
BCC mainfile.obj sub1.obj mylib.lib
```

links MAINFILE.OBJ, SUB1.OBJ, and MYLIB.LIB and produces the executable MAINFILE.EXE.

BCC starts TLINK with the files C0WS.OBJ, CWS.LIB, and IMPORT.LIB (initialization module, run-time library, and Windows import library). BCC32 starts TLINK32 with the files C0W32.OBJ, CW32.LIB, and IMPORT32.LIB by default.

Linking libraries

You must always link the Borland C++ run-time library that contains the standard C/C++ library functions for the type of application you are linking. You must also include the appropriate import library (IMPORT.LIB for 16-bit Windows applications, IMPORT32.LIB for console applications, or IMPRTW32.LIB for 32-bit Windows applications).

Table 9.1 describes the 16-bit Windows 3.*x* libraries and .OBJ files provided by Borland. See the *Library Reference* for a complete list of Windows libraries and the *DOS Reference* for a complete list of DOS libraries and startup files.

Table 9.1 Borland 16-bit libraries and startup files

Libraries and .OBJs	**Description**
C*n*.LIB	Run-time library for DOS applications, where *n* is S, C, M, L or H to indicate Small, Compact, Medium, Large or Huge memory model.
CW*n*.LIB	Run-time library for Windows 3.*x* applications, where *n* is S, C, M, or L to indicate Small, Compact, Medium, or Large memory model.
CRTLDLL.LIB	Run-time library for Windows 3.*x* applications to link in as a .DLL.
IMPORT.LIB	Import library for Windows 3.*x* API functions.
C0*n*.OBJ	Startup code for DOS .EXE applications, where *n* is T, S, C, M, L, or H to indicate Tiny, Small, Compact, Medium, Large or Huge memory model.
C0W*n*.OBJ	Startup code for Windows 3.*x* applications, where *n* is S, M, C, or L to indicate Small, Medium, Compact, or Large memory model.
C0D*n*.OBJ	Startup code for Windows 3.*x* .DLL modules, where *n* is S, M, or L to indicate Small, Medium, or Large memory model.
MATHWS.LIB	If your program uses floating-points, you must include a math library. MATHWS.LIB is for small and tiny models.
MATHWC.LIB	Math library for compact models.

Table 9.1: Borland 16-bit libraries and startup files (continued)

MATHWM.LIB	Math library for medium models.
MATHWL.LIB	Math library for large models.

Table 9.2 describes the 32-bit libraries and .OBJ files provided by Borland; these are used by TLINK32. See the *Library Reference* for a complete list of libraries.

Table 9.2 Borland 32-bit libraries and startup files

Libraries and .OBJs	Description
CW32.LIB	Run-time library for Win32 applications.
IMPORT32.LIB	Import library for console applications and 32-bit Windows applications.
C0X32.OBJ	Startup code for console applications.
C0W32.OBJ	Startup code for Win32 applications.
C0D32.OBJ	Startup code for 32-bit DLL modules.

TLINK options

Unless otherwise specified, options work with both TLINK and TLINK32. Options are case-sensitive and must be preceded by either a slash (/) or a hyphen (-). To turn off a default option, place a hyphen after the option at the command-line (for example, **–P-** or **/P-**). You can place options anywhere in the command line. You don't need spaces after options (**/m/f/c** is the same as **/m /f /c**), but you must separate options and files with a space.

Table 9.3 lists the TLINK command-line options and the IDE equivalent options (note that not all command-line options have an IDE equivalent). Command-line default options are marked by a bullet (■). A more detailed explanation of options, including the IDE option names, follows the table.

Table 9.3: TLINK options

Default	Option	IDE Linker option	For	Description
	/3	Linker|16-bit Linker|Enable 32-bit processing	16-bit	Accepts and links 32-bit code produced by TASM or a compatible assembler.
	/a*x*		32-bit	Specifies application type, where
		Target Attributes|Target Model		**/aa** targets Windows applications
		<none>		**/ap** targets console applications.
	/A:*dd*	16-bit|Segment Alignment	16/32-bit	Specifies page alignment within .EXE file.
	/B:*xxxxxx*	32-bit Linker|Image based address	32-bit	Specifies image base address (in hexadecimal).
	/c	General|Case-sensitive link	16/32-bit	Treats case as significant in symbols.

Table 9.3: TLINK options (continued)

	/C	General|Case-sensitive exports, imports	16-bit	Treats case as significant in EXPORTS and IMPORTS section of module-definition file.
	/d	Warnings|Warn duplicate symbol in .LIB	16-bit	Warns you if there are duplicate symbols in libraries.
	/E	16-bit Linker|Process extended dictionaries	16-bit	Enables processing of extended dictionaries in libraries.
	/E*nn*	32-bit Linker|Maximum linker errors	32-bit	Specifies maximum errors before termination.
■	**/e**	16-bit|Process extended dictionaries (uncheck)	16-bit	Ignores extended dictionaries in libraries. This is the opposite of the **/E** option.
	/f	16-bit Linker|Inhibit optimizing far to near	16-bit	Inhibits optimization of far calls to near data.
	/G*x*		16/32-bit	"Goodies" options where *x* is n, r, or m.
	/Gn	16-bit Linker|Discard nonresident name table	16-bit	Discard nonresident name table.
	/Gr	16-bit|Transfer resident to nonresident table	16-bit	Transfer Resident names to nonresident names table.
	/Gm	Map File|Print mangled names in map file	16/32-bit	Put Mangled names in map file.
	/i	16-bit Linker|Initialize segments	16-bit	Initializes all segments.
	/l	Map File|Include source line numbers	16-bit	Includes source line numbers (lowercase L).
	/L	Directories|Library (*not under Linker in IDE*)	16/32-bit	Specifies library search paths.
	/m	Map File|Public	16/32-bit	Creates map file with publics.
	/n	General|Default Libraries	16-bit	Don't use default libraries.
	/o	Overlay module (Node attributes dialog box)	16-bit	Overlays modules or libraries.
■	**/P**	General|Pack code segments	16-bit	Packs code segments.
	/Rk	Resource|Pack fast load area (*not under Linker*)	16-bit	Sends options to RLINK.EXE.
	/Rv	<none>	32-bit	Verbose resource binding.
	/Re*xxxx*	<none>	32-bit	Renames the executable to *xxxx*.
	/S:*xxxxxx*	32-bit Linker|Stack size	32-bit	Specifies stack size (in hexadecimal).
	/s	Map File|Detailed	16-bit	Creates detailed map of segments.
	/t	<none>	16-bit	Creates a tiny-model DOS .COM file.
	/Td*x*		16-bit	Specifies application target, where
		<none>		**/Tdc** means build a DOS .COM file.
■		TargetExpert Platform		**/Tde** means build a DOS .EXE file.
	/Tp*x*	TargetExpert Platform	32-bit	Specifies application target, where
■				**/Tpe** means build a 32-bit .EXE file.
				/Tpd means build a 32-bit DLL.
	/Tw*x*	TargetExpert Target Type	16-bit	Specifies Windows 3.*x* target application, where
■				**/Twe** builds a Windows .EXE file.
				/Twd builds a Windows DLL.
	/v	General|Include debug information	16/32-bit	Includes full symbolic debug information.
	/w*xxx*	Warnings (see page 153 for information)	32-bit	Enable or disable warnings (see page 153).
	/x	Map File|Off	16/32-bit	Doesn't create a map file.
	/ye	<none>	16-bit	Uses expanded memory for swapping.
	/yx	<none>	16-bit	Configures TLINK's use of extended memory swapping.

/3 (32-bit code) lets you link 32-bit DOS object modules produced by TASM or a compatible assembler. This option increases the memory requirements for TLINK and slows down linking.

/a (application type) lets you specify the type of EXE image:

- **/aa** targets Windows applications.
- **/ap** targets console applications that can be run in a window.

/A:*dd* (align pages) specifies page alignment for code and data within the executable file where *dd* must be a decimal power of 2. For example, if you specify an alignment value of **/A:12**, the sections in the image are stored on 4096-byte boundaries. The operating system seeks pages for loading based on this alignment value. The default is **/A:9**, which means sections are aligned on 512-byte boundaries within the executable file.

/B:*xxxxxx* (base address) specifies an image base address for an application. If this option is used, internal fixups are removed from the image, and the requested load address of the first object is set to the hexadecimal number given with the option. All successive objects are aligned on 64K linear address boundaries. This option makes applications smaller on disk, and improves both load-time and run-time performance since the operating system no longer has to apply internal fixups. Because NT loads all .EXE images at 64K, you're advised to link all .EXEs with **/B:0x10000**.

/c (case sensitivity) makes the case significant in public and external symbols.

/C (case sensitivity) makes the case significant in the EXPORTS and IMPORTS sections in module-definition files.

/d (duplicate symbols) warns you if a symbol appears in more than one library file. If the symbol must be included in the program, TLINK uses the symbol from the first file containing the symbol that is specified on the command line (or in a response file). This option also warns you if symbols appear in both an .OBJ file and a .LIB file (TLINK uses the first one linked in and ignores the others).

/E*nn* (maximum errors) specifies the maximum number of errors the linker reports before terminating. **/E0** (the default) reports an infinite number of errors (that is, as many as possible).

/E (extended dictionaries) processes extended dictionaries. The library files in Borland C++ contain an extended dictionary with information that lets TLINK use less memory and link faster with those libraries. You can add the extended dictionary to other library files using TLIB's **/E** option (see the

TLIB section on page 170). Avoid using **/E** with programs linked with libraries that have been built without an extended dictionary (third-party libraries that have been provided without source code, for example). To use extended dictionaries, all linked libraries must have extended dictionaries.

/e (ignore extended dictionaries) ignores extended dictionaries. This is the opposite of the **/E** option, and is the default.

/f (inhibit far optimizations) inhibits the optimization of far calls to near data.

/Gx (Goodies) are options where *x* can be
  n = Discard nonresident name table.
  r = Transfer resident names to nonresident table.
  m = (TLINK32) Put mangled names in map file; this can help you identify how names are mangled.

/i (uninitialized trailing segments) outputs uninitialized trailing segments into the executable file even if the segments don't contain data records.

/l (line numbers) creates a section in the .MAP file for source-code line numbers. Linked .OBJ files must be compiled with debug information using **–y** or **–v**. If you use **/x** to suppress map file creation, the **/l** (lowercase L) option has no effect.

/L (library search paths) lets you list directories for TLINK to search if you don't type an explicit path for a library or the C or C++ initialization module. TLINK searches the current directory first (where you typed TLINK). For example,

```
TLINK /Lc:\BC4\lib;c:\mylibs splash logo,,,utils .\logolib
```

first searches the current directory for UTILS.LIB, then searches C:\BC4\LIB, then C:\MYLIBS. Because LOGOLIB explicitly names the current directory, TLINK doesn't search the libraries specified with the **/L** option to find LOGOLIB.LIB.

/m (detailed map file) creates a more complete map than TLINK normally does by adding a list of sorted public symbols to the map file. This kind of map file is useful in debugging. Many debuggers can use the list of public symbols, which lets you refer to symbolic addresses when you're debugging. If you don't specify map file options (**/m**, **/s**, or **/x**), then the option Map File | Segments is used. See also **/s**.

/M (map mangled) maps with mangled public names.

/n (ignore default libraries) ignores default libraries specified by some compilers. Use this option when linking modules that are written in another language.

/o (overlays) overlays code in all the modules or libraries that follow the option on the command line (this option works for DOS applications only). Use **/o-** on the command line to turn off overlays. If you specify a class name after this option, all the segments in that class are overlaid (you can do this for multiple classes). If you don't specify any name after this option, all segments of classes ending with CODE are overlaid. This option uses the default overlay interrupt number of 3FH. To specify a different interrupt number, use **/o#*xx***, where *xx* is a two-digit hexadecimal number.

/P (pack code segments) combines as many code segments as possible in one physical segment up to (and never greater than) the code-segment packing limit. TLINK starts a new segment if it needs to. The default code-segment packing limit is 8,192 bytes (8K). To change it, use /P=*n* where *n* specifies the number of bytes between 1 and 65,536. You would probably want the limit to be a multiple of 4K under 386 enhanced mode because of the paging granularity of the system.

Although the optimum segment size in 386 enhanced mode is 4K, the default code-segment packing size is 8K because typical code segments are from 4K to 8K and 8K might pack more efficiently.

Because each maintained segment has some system overhead, code-segment packing typically increases performance. **/P-** turns off code-segment packing (useful if you've turned it on in the configuration file and want to disable it for a particular link).

/s (detailed segment map) creates a map file with the same features as the **/m** option, but adds a detailed segment map. If you don't specify map file options (**/m**, **/s**, or **/x**), then the option Map File | Segments is used. For each segment in each module, this map file includes the address, length in bytes, class, segment name, group, module, and ACBP information. If the same segment appears in more than one module, each module appears as a separate line. Except for the ACBP field, the information in the detailed segment map is self-explanatory.

The ACBP field encodes the A (alignment), C (combination), and B (big) attributes into a set of four bit fields, as defined by Intel. TLINK uses only three of the fields, the A, C, and B fields. The ACBP value in the map is printed in hexadecimal. The following values of the fields must be OR'ed together to arrive at the ACBP value printed.

Field	Value	Description
The A field (alignment)	00	An absolute segment.
	20	A byte-aligned segment.
	40	A word-aligned segment.
	60	A paragraph-aligned segment.
	80	A page-aligned segment.
	A0	An unnamed absolute portion of storage.
The C field (combination)	00	Cannot be combined.
	08	A public combining segment.
The B field (big)	00	Segment less than 64K.
	02	Segment exactly 64K.

With the **/s** option, public symbols with no references are flagged "idle." An idle symbol is a publicly-defined symbol in a module that wasn't referenced by an EXTDEF record by any other module included in the link. For example, this fragment from the public symbol section of a map file indicates that symbols *Symbol1* and *Symbol3* aren't referenced by the image being linked:

```
0002:00000874   Idle   Symbol1
0002:00000CE4          Symbol2
0002:000000E7   Idle   Symbol3
```

32 **/S:*xxxxxx* (stack size)** sets the application stack size in hexadecimal where *xxxxxx* is a hexadecimal string. Specifying the stack size with **/S** overrides any stack size setting in a module-definition file.

/t (tiny model DOS .COM file) creates a DOS tiny-model .COM file (this option works the same as **/Tdc**, except you can use **/t** with BCC.EXE). DOS .COM files can't exceed 64K in size, have segment-relative fixups, or define a stack segment. They must have a starting address of 0:100H. If you change the file extension (to .BIN, for example), the starting address can be either 0:0 or 0:100H. The linker can't generate debugging information for .COM files, so you'll need to debug it as an .EXE, then recompile and link as a .COM file.

/Td*x* (DOS target) produces a DOS .EXE (**/Tde**) or a DOS .COM (**/Tdc**) file.

32 **/Tp*x* (protected target)** produces a protected mode .EXE (**/Tpe**) or .DLL file (**/Tpd**).

16 **/Tw*x* (target type)** produces a Windows .EXE (**/Twe**) or .DLL file (**/Twd**). This option isn't necessary if you include a module-definition file with an EXETYPE Windows statement because TLINK creates an application (.EXE) if the module-definition file has a NAME statement or a DLL if the module-definition file has a LIBRARY statement.

/v (debugging information) includes debugging information in the executable file. If this option is found anywhere on the command line, debugging information is included in the executable file for all object modules that contain debugging information. You can use the **/v+** and **/v–** options to selectively enable or disable debugging information on a module-by-module basis (but not on the same command line as **/v**). For example, the command

```
TLINK mod1 /v+ mod2 mod3 /v- mod4
```

includes debugging information for modules *mod2* and *mod3*, but not for *mod1* and *mod4*.

/w*xxx* (warning control) turns on (**/w*xxx***) or off (**/w-*xxx***) TLINK warnings, where *xxx* can be one of the following (defaults mean TLINK will send the warning without you specifically turning it on):

Table 9.4
TLINK32 warnings

Default	/w option	IDE description
■	**bdl**	Using based linking in DLLs (might cause the DLL to malfunction)
	def	No .DEF file; using defaults
	dpl	Warn duplicate symbol in .LIB
■	**dup**	Duplicate symbol (warning for .OBJs)
■	**ent**	No entry point
	imt	Import doesn't match previous definition
	inq	Extern not qualified with _ _import
■	**srf**	Self-relative fixup overflowed
		"No stack" warning

/x (no map file) tells TLINK to not generate a map file. TLINK usually creates map files that list segments in the program, the program start address, and any TLINK warning or error messages (the Map File | Segments option, which has no command-line option, is on by default).

/ye (expanded memory) controls TLINK's use of expanded memory for I/O buffering. If TLINK needs more memory for active data structures (while reading object files or writing the executable file), it either clears buffers or swaps them to expanded memory.

When reading files, TLINK clears the input buffer so that its space can be used for other data structures. When creating an executable, it writes the buffer to its correct place in the executable file. In both cases, you can substantially increase the link speed by swapping to expanded memory. By default, swapping to expanded memory is enabled, and swapping to extended memory is disabled. If swapping is enabled and no appropriate memory exists in which to swap, then swapping doesn't occur.

This option has several forms, shown below

- **/ye** or **/ye+** enables expanded memory swapping (this is the default).
- **ye-** disables expanded memory swapping (this is off by default).

/yx (extended memory) controls TLINK's use of extended memory for I/O buffering. By default, TLINK can use up to 8MB of extended memory. You can change TLINK's use of extended memory with one of the following forms of this option:

- **/yx+** uses all available extended memory up to 8MB.
- **/yx*n*** uses only up to *n* KB extended memory.

Module-definition file reference

IMPDEF creates module-definition files, and IMPLIB creates import libraries from module-definition files. See Chapter 10 for more information on these tools.

This section describes module-definition files and the statements that appear in them. A module-definition file provides information to TLINK about the contents and system requirements of a Windows application. More specifically, a module-definition file

- Names the .EXE or .DLL.
- Identifies the application type.
- Lists imported and exported functions.
- Describes the code and data segment attributes, and lets you specify attributes for additional code and data segments.
- Specifies the size of the stack.
- Provides for the inclusion of a stub program.

See the *Programmer's Guide* for uses and examples of module-definition files.

CODE statement

CODE defines the default attributes of code segments. Code segments can have any name, but must belong to segment classes whose name ends in CODE (such as CODE or MYCODE). The syntax is

```
TLINK                                TLINK32
CODE [FIXED|MOVEABLE]                [PRELOAD | LOADONCALL]
     [DISCARDABLE|NONDISCARDABLE]    [EXECUTEONLY | EXECUTEREAD]
     [PRELOAD|LOADONCALL]
```

- FIXED (the default) means the segment remains at a fixed memory location; MOVEABLE means the segment can be moved.
- PRELOAD means code is loaded when the calling program is loaded. LOADONCALL (the default) means the code is loaded when called by the program.

- DISCARDABLE means the segment can be discarded if it is no longer needed (this implies MOVABLE). NONDISCARDABLE (the default) means the segment can't be discarded.
- EXECUTEONLY means a code segment can only be executed. EXECUTEREAD (the default) means the code segment can be read and executed.
- PRELOAD means the segment is loaded when the module is first loaded. LOADONCALL (the default) means the segment is loaded when called.

DATA statement

DATA defines attributes of data segments. The syntax is

```
DATA [NONE | SINGLE | MULTIPLE]
     [READONLY | READWRITE]
     [PRELOAD | LOADONCALL]
     [SHARED | NONSHARED]
```

- NONE means that there is no data segment created. This option is available only for libraries. SINGLE (the default for .DLLs) means a single data segment is created and shared by all processes. MULTIPLE (the default for .EXEs) means a data segment is created for each process.
- READONLY means the data segment can be read only. READWRITE (the default) means the data segment can be read and written to.
- PRELOAD means the data segment is loaded when a module that uses it is first loaded. LOADONCALL (the default) means the data segment is loaded when it is first accessed (this is ignored for 32-bit applications).
- SHARED (the default for 16-bit .DLLs) means one copy of the data segment is shared among all processes. NONSHARED (the default for programs and 32-bit .DLLs) means a copy of the data segment is loaded for each process needing to use the data segment.

DESCRIPTION statement

DESCRIPTION (optional) inserts text into the application module and is typically used to embed author, date, or copyright information. The syntax is

```
DESCRIPTION 'Text'
```

Text is an ASCII string delimited with single quotes.

EXETYPE statement

EXETYPE defines the default executable file (.EXE) header type for 16-bit application. You can leave this section in for 32-bit application for backward compatibility, but if you need to change the EXETYPE, see the NAME statement on page 158. The syntax for EXETYPE is

```
EXETYPE  WINDOWS
```

EXPORTS statement

EXPORTS defines the names and attributes of functions to be exported. The EXPORTS keyword marks the beginning of the definitions. It can be followed by any number of export definitions, each on a separate line. The syntax is

```
EXPORTS
     ExportName [Ordinal] [RESIDENTNAME] [Parameter]
```

- *ExportName* specifies an ASCII string that defines the symbol to be exported:

  ```
  EntryName [=InternalName]
  ```

  InternalName is the name used within the application to refer to this entry. *EntryName* is the name listed in the executable file's entry table and is externally visible.

- *Ordinal* defines the function's ordinal value as follows:

  ```
  @ordinal
  ```

  where *ordinal* is an integer value that specifies the function's ordinal value.

  When an application or DLL module calls a function exported from a DLL, the calling module can refer to the function by name or by ordinal value. It's faster to refer to the function by ordinal because string comparisons aren't required to locate the function. To use less memory, export a function by ordinal (from the point of view of that function's DLL) and import/call a function by ordinal (from the point of view of the calling module).

  When a function is exported by ordinal, the name resides in the nonresident name table. When a function is exported by name, the name resides in the resident name table. The resident name table for a module is in memory whenever the module is loaded; the nonresident name table isn't.

- RESIDENTNAME specifies that the function's name must be resident at all times. This is useful only when exporting by ordinal (when the name wouldn't be resident by default).

- *Parameter* is an optional integer value that specifies the number of words the function expects to be passed as parameters.

IMPORTS statement

IMPORTS defines the names and attributes of functions to be imported from DLLs. Instead of listing imported DLL functions in the IMPORTS statement, you can either

- Specify an import library for the DLL in the TLINK command line, or
- Include the import library for the DLL in the project manager in the IDE.

If you're programming for 32-bits, you must use **__import** to import any function, class, or data you want imported; for 16-bits you must use **__import** with classes. See the *Programmer's Guide* for more information on using **__import**.

The IMPORTS keyword marks the beginning of the definitions; it is followed by any number of import definitions, each on a separate line. The syntax is

```
IMPORTS
      [InternalName=]ModuleName.Entry
```

- *InternalName* is an ASCII string that specifies the unique name the application uses to call the function.
- *ModuleName* specifies one or more uppercase ASCII characters that define the name of the executable module containing the function. The module name must match the name of the executable file. For example, the file SAMPLE.DLL has the module name SAMPLE.
- *Entry* specifies the function to be imported—either an ASCII string that names the function or an integer that gives the function's ordinal value.

LIBRARY statement

LIBRARY defines the name of a DLL module. A module-definition file can contain either a LIBRARY statement to indicate a DLL or a NAME statement to indicate a program.

A library's module name must match the name of the executable file. For example, the library MYLIB.DLL has the module name MYLIB. The syntax is

```
LIBRARY LibraryName [INITGLOBAL | INITINSTANCE]
```

- *LibraryName* (optional) is an ASCII string that defines the name of the library module. If you don't include a *LibraryName*, TLINK uses the file name with the extension removed. If the module-definition file includes neither a NAME nor a LIBRARY statement, TLINK assumes a NAME statement without a *ModuleName* parameter.
- INITGLOBAL means the library-initialization routine is called only when the library module is first loaded into memory. INITINSTANCE means

the library-initialization routine is called each time a new process uses the library.

NAME statement

NAME is the name of the application's executable module. The module name identifies the module when exporting functions. For 32-bit applications, NAME must appear before EXETYPE. If NAME and EXETYPE don't specify the same target type, the type listed with NAME is used. The syntax is

```
NAME ModuleName [WINDOWSAPI] | [WINDOWCOMPAT]
```

ModuleName (optional) specifies one or more uppercase ASCII characters that name the executable module. The name must match the name of the executable file. For example, an application with the executable file SAMPLE.EXE has the module name SAMPLE.

If *ModuleName* is missing, TLINK assumes the module name matches the file name of the executable file. For example, if you don't specify a module name and the executable file is named MYAPP.EXE, TLINK assumes the module name is MYAPP.

If the module-definition file includes neither a NAME nor a LIBRARY statement, TLINK assumes a NAME statement without a *ModuleName* parameter.

WINDOWAPI is a Windows executable, and is equivalent to the TLINK32 option **/aa**.

WINDOWCOMPAT is a Windows-compatible character-mode executable, and is equivalent to the TLINK32 option **/ap**.

SEGMENTS statement

SEGMENTS defines the segment attributes of additional code and data segments. The syntax is

```
SEGMENTS
     SegmentName [CLASS 'ClassName'] [MinAlloc]
     [SHARED | NONSHARED]
     [PRELOAD | LOADONCALL]
```

- *SegmentName* is a character string that names the new segment. It can be any name, including the standard segment names _TEXT and _DATA, which represent the standard code and data segments.
- *ClassName* (optional) is the class name of the specified segment. If no class name is specified, TLINK uses the class name CODE.
- *MinAlloc* (optional) is an integer that specifies the minimum allocation size for the segment. TLINK and TLINK32 ignore this value.

- SHARED (the default for 16-bit .DLLs) means one copy of the segment is shared among all processes. NONSHARED (the default for programs and 32-bit .DLLs) means a copy of the segment is loaded for each process needing to use the data segment.
- PRELOAD means that the segment is loaded immediately; LOADONCALL means that the segment is loaded when it is accessed or called (this is ignored by TLINK32). The resource compiler might override the LOADONCALL option and preload segments instead.

STACKSIZE statement

STACKSIZE defines the number of bytes needed by the application for its local stack. An application uses the local stack whenever it makes function calls. Don't use the STACKSIZE statement for dynamic-link libraries. The syntax is

```
STACKSIZE bytes
```

bytes is an integer value that specifies the stack size in bytes.

STUB statement

STUB appends a DOS executable file specified by *FileName* to the beginning of the module. The executable stub displays a warning message and terminates if the user attempts to run the executable stub in the wrong environment (running a Windows application under DOS, for example).

Borland C++ adds a built-in stub to the beginning of a Windows application unless a different stub is specified with the STUB statement. You shouldn't use the STUB statement to include WINSTUB.EXE because the linker does this automatically. The syntax is

```
STUB "FileName"
```

FileName is the name of the DOS executable file to be appended to the module. The name must have the DOS file name format.

If the file named by *FileName* isn't in the current directory, TLINK searches for the file in the directories specified by the PATH environment variable.

Module-definition file defaults

The module-definition file isn't strictly necessary to produce a Windows executable under Borland C++.

If no module-definition file is specified, the following defaults are assumed:

```
CODE        PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE MULTIPLE (for applications)
            PRELOAD MOVEABLE SINGLE (for DLLs)
HEAPSIZE    4096
STACKSIZE   5120 (1048576 for TLINK32)
```

To replace the EXETYPE statement, the Borland C++ linker can discover what kind of executable you want to produce by checking settings in the IDE or options on the command line.

You can include an import library to substitute for the IMPORTS section of the module definition file.

You can use the **_export** keyword in the definitions of export functions in your C and C++ source code to remove the need for an EXPORTS section. Note, however, that if **_export** is used to export a function, that function is exported by name rather than by ordinal (ordinal is usually more efficient).

If you want to change various attributes from the default, you'll need to have a module-definition file.

C H A P T E R 10

Using resource tools

This chapter describes the Borland resource tools.

- BRCC.EXE and BRCC32.EXE are the Borland resource compilers. They compile resource script files (.RC files) and produce the binary .RES file.
- RLINK.EXE and RLINK32.DLL (through TLINK32.EXE) are the Borland resource linkers that bind resources, in .RES file form, to an .EXE file, and mark the resulting .EXE file as a Windows executable.
- BRC.EXE and BRC32.EXE are shells through which both BRCC or BRCC32 and RLINK or RLINK32 (through TLINK32) can be started in a single step.

All 32-bit resource tools work exactly like the 16-bit tools unless specified in this chapter.

Windows programs provide a familiar and standard user interface for all applications. The components of the user interface are known as resources and can include:

- Menus
- Dialog boxes
- Pointers
- Icons
- Bitmaps
- Strings
- Accelerator keys
- Fonts

Resources are defined external to your source code, and then attached to the executable file during linking. The application calls resources into memory only when needed, thus minimizing memory usage.

Resource script (.RC) files are text files that describe the resources a particular application will use. BRCC and RC use the .RC file to compile the resources into a binary format resource (.RES) file. RLINK then attaches the .RES file, which contains your resources, to your executable (this is called resource linking).

BRCC.EXE: The resource compiler

BRCC is a command-line version of Resource Workshop's resource compiler. It accepts a resource script file (.RC) as input and produces a resource object file (.RES) as output. BRCC uses the following command-line syntax:

```
BRCC [options] <filename>.RC
```

Table 10.1 lists all BRCC options. Note that options are not case-sensitive (-r is the same as –R).

Table 10.1 BRCC (Borland resource compiler)

Switch	Description
@responsefile	Takes instructions from the specified command file.
–d<name>[=<string>]	Defines a preprocessor symbol.
–fo<filename>	Renames the output .RES file. (By default, BRCC creates the output .RES file with the same name as the input .RC file.)
–i<path>	Adds one or more directories (separated by semicolons) to the include search path.
–r	This switch is ignored. It is included for compatibility with other resource compilers.
–v	Prints progress messages (verbose).
–x	Deletes the current include path.
–? or –h	Displays switch help.
–30	Builds Windows 3.0-compatible .RES files.
–31	Builds Windows 3.1-compatible .RES files.
–w32	Builds Win32-compatible .RES files.

Like Resource Workshop's resource compiler, BRCC predefines common resource-related Windows constants such as WS_VISIBLE and BS_PUSHBUTTON. Also, two special compiler-related symbols are defined: RC_INVOKED and WORKSHOP_INVOKED. These symbols can be used in the source text in conjunction with conditional preprocessor statements to control compilation. For example, the following construct can greatly speed up compilation:

```
#ifndef WORKSHOP_INVOKED
#include "windows.h"
#endif
```

The following example adds two directories to the include path and produces a .RES file with the same name as the input .RC file.

```
brcc -i<dir1>;<dir2> <filename>.RC
```

This example produces an output .RES file with a name different from the input .RC file name:

```
brcc -fo<filename>.RES <filename>.RC
```

RLINK: the resource linker

RLINK combines a .RES file with an .EXE file to produce a new Windows executable. RLINK accepts as input one or more object files (.RES) and a single Windows executable file. RLINK links the resources by fixing up stringtables and messagetables and then binding these linked resources to the executable.

RLINK uses the following command-line syntax:

```
rlink [options] <filename>.RES <filename>.EXE
```

RLINK accepts these options:

Table 10.2
RLINK switches

Switch	Description
@<filename>	Takes instructions from the specified command file.
–d	Removes resources from the .EXE file (no .RES file is specified).
–fe<filename>	Renames the output .EXE file.
–fi<filename>	Renames the input .RES file.
–k	Don't reorder segments for fastload.
–v	Prints progress messages (verbose listing).
–vx	Lists resources but does not bind to EXE file.
–? or –h	Displays switch help.
–30	Builds Windows 3.0-compatible .RES files.
–31	Builds Windows 3.1-compatible .RES files.

The following example binds the resources in the .RES file into the .EXE file.

```
rlink <filename>.RES <filename>.EXE
```

The next example links the resources in the two .RES files and binds them to the .EXE file.

```
rlink -fi<filename>.RES <filename>.RES <filename>.EXE
```

The next example combines the program code in the input .EXE file with the resources in the input .RES file and produces an output .EXE file with a new name.

```
rlink -fe<filename>.EXE <filename>.RES <filename>.EXE
```

The final example takes input from an .RLK command file. It then links the resources in three .RES files and binds them to the .EXE file.

```
rlink @<filename>.RLK
```

The command file (<filename>.RLK) contains

```
-fi<filename>.RES
-fi<filename>.RES
<filename>.RES
<filename>.EXE
```

BRC.EXE: the resource shell

The Borland Resource Compiler (BRC) is a resource compiler shell. It invokes either BRCC or RLINK or both, depending on the command-line syntax. The command-line syntax for BRC is as follows:

```
brc [switches] <filename>.RC [<filename>.EXE]
```

BRC accepts these switches:

Table 10.3 BRC switches

Switch	Description
–d<name>=string	Defines a symbol you can test with the #IFDEF preprocessor directive.
–fo<filename>	Renames the .RES file.
–fe<filename>	Renames the .EXE file.
–fi<filename>	Specifies additional .RES files.
–i<path>	Adds one or more directories (separated by semicolons) to the include search path.
–k	Don't create fastload area.
–r	Creates a .RES file only. The compiled .RES file is not added to the .EXE.
–v	Prints progress messages (verbose listing).

Table 10.3: BRC switches (continued)

–x	Directs the compiler to ignore the INCLUDE environment variable when it searches for include or resource files.
–31	Builds Windows 3.1-compatible .RES files.
–w32	Builds Win32-compatible .RES files.

Depending on your task, there are several variations on the basic BRC command-line syntax:

- The following statement compiles the .RC file, creates a .RES file, and adds the .RES file to the executable file.

  ```
  brc <filename>.RC [<filename>.EXE]
  ```

  BRC automatically seeks an .EXE file with the same name as the .RC file. You need to specify the .EXE file only if its name is different from the .RC file's.
- The following statement creates a .RES file, but not an .EXE file. If you name an .EXE file in the command line, BRC ignores it.

  ```
  brc -r <filename>.RC
  ```
- The following statement adds an existing .RES file to an executable file. The .EXE file name is required only if it differs from the .RES file name.

  ```
  brc <filename>.RES [<filename>.EXE]
  ```

C H A P T E R 11

Using libraries

This chapter describes several tools that let you work with library files. You can use these tools from either the IDE or the command line.

- IMPLIB creates import libraries, and IMPDEF creates module definition files (.DEF files). Import libraries and module definition files provide information to the linker about functions imported from dynamic-link libraries (DLLs).
- TLIB is a utility that manages libraries of individual .OBJ (object module) files. A library is a convenient tool for dealing with a collection of object modules as a single unit.

Using IMPLIB: The import librarian

The IMPLIB utility creates import libraries. IMPLIB takes as input DLLs, module definition files, or both, and produces an import library as output. The IDE uses IMPLIB as a translator for a DLL target (see Chapter 2 for information on the project manager and targets). When you add a DLL as a target, the project manager compiles and links the DLL's dependent files to create the .DLL file, then runs IMPLIB to create a .LIB file. You can also run IMPLIB from the IDE (see page 20 for information on using tools from the IDE).

Import libraries contain records. Each record contains the name of a DLL, and specifies where in the DLL the imported functions reside. These records are bound to the application by TLINK or the IDE linker, and provide Windows with the information necessary to resolve DLL function calls. An import library can be substituted for part or all of the IMPORTS section of a module definition file.

If you've created a Windows application, you've already used at least one import library, IMPORT.LIB, the import library for the standard Windows DLLs. (IMPORT.LIB is linked automatically when you build a Windows application in the IDE and when using BCC to link. You have to explicitly link with IMPORT.LIB only if you're using TLINK to link separately.)

An import library lists some or all of the exported functions for one or more DLLs. IMPLIB creates an import library directly from DLLs or from module definition files for DLLs (or a combination of the two).

To create an import library for a DLL, type

```
IMPLIB Options LibName [ DefFiles... | DLLs... ]
```

A DLL can also have an extension of .EXE or .DRV, not just .DLL.

where *Options* is an optional list of one or more IMPLIB options (see Table 11.1), *LibName* is the name for the new import library, *DefFiles* is a list of one or more existing module definition files for one or more DLLs, and *DLLs* is a list of one or more existing DLLs. You must specify at least one DLL or module definition file.

Table 11.1
IMPLIB options

Options must be lowercase and preceded by either a hyphen or a slash.

Option	Description
–c	Accepts case-sensitive symbols. If you have two symbols that differ only in case (like MYSYM and mysym) and you don't use **–c**, IMPLIB uses the first symbol and treats the second one as a duplicate.
–i	Tells IMPLIB to ignore WEP, the Windows exit procedure required to end a DLL. Use this option if you are specifying more than one DLL on the IMPLIB command line.
–w	No warnings.

Using IMPDEF: The module-definition file manager

Import libraries provide access to the functions in a Windows DLL. See page 167 for more details.

IMPDEF takes as input a DLL name, and produces as output a module definition file with an export section containing the names of functions exported by the DLL. The syntax is

```
IMPDEF DestName.DEF SourceName.DLL
```

This creates a module definition file named *DestName*.DEF from the file *SourceName*.DLL. The resulting module definition file would look something like this:

```
LIBRARY     FileName

DESCRIPTION 'Description'

EXPORTS
            ExportFuncName              @Ordinal
               ⋮
            ExportFuncName              @Ordinal
```

where *FileName* is the DLL's root file name, *Description* is the value of the DESCRIPTION statement if the DLL was previously linked with a module definition file that included a DESCRIPTION statement, *ExportFuncName* names an exported function, and *Ordinal* is that function's ordinal value (an integer).

Classes in a DLL

IMPDEF is useful for a DLL that uses C++ classes. If you use the **_export** keyword when defining a class, all of the non-inline member functions and static data members for that class are exported. It's easier to let IMPDEF make a module definition file for you because it lists all the exported functions, and automatically includes the member functions and static data members.

Since the names of these functions are mangled, it would be tedious to list them all in the EXPORTS section of a module definition file simply to create an import library from the module definition file. If you use IMPDEF to create the module definition file, it includes the ordinal value for each exported function. If the exported name is mangled, IMPDEF also includes that function's unmangled, original name as a comment following the function entry. So, for instance, the module definition file created by IMPDEF for a DLL that used C++ classes would look something like this:

```
LIBRARY     FileName

DESCRIPTION 'Description'

EXPORTS
            MangledExportFuncName  @Ordinal ; ExportFuncName
               :
            MangledExportFuncName  @Ordinal ; ExportFuncName
```

where *FileName* is the DLL's root file name, *Description* is the value of the DESCRIPTION statement if the DLL was previously linked with a module definition file that included a DESCRIPTION statement, *MangledExportFuncName* provides the mangled name, *Ordinal* is that function's ordinal value (an integer), and *ExportFuncName* gives the function's original name.

Functions in a DLL

IMPDEF creates an editable source file that lists all the exported functions in the DLL. You can edit this .DEF file to contain only those functions that you want to make available to a particular application, then run IMPLIB on the edited .DEF file. This results in an import library that contains import information for a specific subset of a DLL's export functions.

Suppose you're distributing a DLL that provides functions to be used by several applications. Every export function in the DLL is defined with **_export**. Now, if all the applications used all the DLL's exports, then you could use IMPLIB to make one import library for the DLL. You could deliver that import library with the DLL, and it would provide import information for all of the DLL's exports. The import library could be linked to any application, thus eliminating the need for the particular application to list every DLL function it uses in the IMPORTS section of its module definition file.

But let's say you want to give only a few of the DLL's exports to a particular application. Ideally, you want a customized import library to be linked to that application—an import library that provides import information only for the subset of functions that the application uses. All of the other export functions in the DLL are hidden to that client application.

To create an import library that satisfies these conditions, run IMPDEF on the compiled and linked DLL. IMPDEF produces a module definition file that contains an EXPORT section listing all of the DLL's export functions. You can edit that module definition file, remove the EXPORTS section entries for those functions you don't want in the customized import library, and then run IMPLIB on the module definition file. The result is an import library that contains import information for only those export functions listed in the EXPORTS section of the module definition file.

Using TLIB: the Turbo Librarian

When it modifies an existing library, TLIB always creates a copy of the original library with a .BAK extension.

The libraries included with Borland C++ were built with TLIB. You can use TLIB to build and modify your own libraries, or to modify the Borland C++ libraries, libraries furnished by other programmers, or commercial libraries you've purchased. You can also use TLIB to

- Create a new library from a group of object modules.
- Add object modules or other libraries to an existing library.
- Remove object modules from an existing library.
- Replace object modules from an existing library.
- Extract object modules from an existing library.
- List the contents of a new or existing library.

See the section on the **/E** option (page 172) for details.

TLIB can also create (and include in the library file) an extended dictionary, which can be used to speed up linking.

Although TLIB is not essential for creating executable programs with Borland C++, it saves you time on large development projects.

Why use object module libraries?

When you program in C and C++, you often create a collection of useful functions and classes. Because of C and C++'s modularity, you are likely to split those functions into many separately compiled source files. Any particular program might use only a subset of functions from the entire collection.

An object module library manages a collection of functions and classes. When you link your program with a library, the linker scans the library and automatically selects only those modules needed for the current program.

The TLIB command line

The TLIB command line takes the following general form, where items listed in square brackets are optional:

```
tlib [@respfile] [option] libname [operations] [, listfile]
```

For an online summary of TLIB options, type `TLIB` and press *Enter.*

In the IDE, you can create a library as a target in a project file. Using TargetExpert, choose Static Library for the target type (see Chapter 2 for help using the project manager). TLIB is the default translator for a library file and it uses options you set in the Project Options dialog box under the Librarian section (choose Options | Project from the main menu).

Table 11.2: TLIB options

Option	Librarian (IDE option)	Description
@respfile		The path and name of the response file you want to include. You can specify more than one response file.
libname		The DOS path name of the library you want to create or manage. Every TLIB command must be given a *libname*. Wildcards are not allowed. TLIB assumes an extension of .LIB if none is given. Use only the .LIB extension because the command-line compilers (BCC and BCC32) and the IDE require the .LIB extension to recognize library files. **Note:** If the named library does not exist and there are *add* operations, TLIB creates the library.
/C	Case-sensitive library	The case-sensitive flag. This option is not normally used; see page 172 for a detailed explanation.
/E	Create extended dictionary	Creates Extended Dictionary; see page 172 for a detailed explanation.
/P*size*	Library Page Size	Sets the library page size to *size*; see page 172 for a detailed explanation.
/0	Purge comment records	Removes comment records from the library.
operations		The list of operations TLIB performs. Operations can appear in any order. If you only want to examine the contents of the library, don't give any operations.
listfile	Generate list file	The name of the file that lists library contents. It must be preceded by a comma. No listing is produced if you don't give a file name. The listing is an alphabetical list of each module. The entry for each module contains an alphabetical list of each public symbol defined in that module. The default extension for the *listfile* is .LST. You can direct the listing to the screen by using the *listfile* name CON, or to the printer by using the name PRN.

Using response files

When you use a large number of operations, or if you find yourself repeating certain sets of operations over and over, you will probably want to use response files. A response file is an ASCII text file (which can be created with the Borland C++ editor) that contains all or part of a TLIB command. Using response files, you can build TLIB commands larger than would fit on one command line. Response files can

- Contain more than one line of text; use the ampersand character (**&**) at the end of a line to indicate that another line follows.
- Include a partial list of commands. You can combine options from the command line with options in a response file.
- Be used with other response files in a single TLIB command line.

Using case-sensitive symbols: The /C option

TLIB maintains a dictionary of all public symbols defined in the modules of the library. When you add a module to a library, its symbol must be unique. If you try to add a module to the library that duplicates a symbol, TLIB displays an error message and doesn't add the module.

Don't use **/C** if you plan to use the library with other linkers or let other people use the library.

Because some linkers aren't case-sensitive, TLIB rejects symbols that differ only in case (for example, the symbols *lookup* and *LOOKUP* are treated as duplicates). TLINK, however, *can* distinguish case, so if you use your library only with TLINK, you can use the TLIB **/C** option to add a module to a library that includes symbols differing only in case.

Creating an extended dictionary: The /E option

To increase the linker's capacity for large links, you can use TLIB to create an extended dictionary and append it to the library file. This dictionary contains, in a compact form, information that is not included in the standard library dictionary and that lets the linker (TLINK) preprocess library files so that any unnecessary modules are not preprocessed.

To create an extended dictionary for a library that you're modifying, use the **/E** option when you start TLIB to add, remove, or replace modules in the library. You can also use the **/E** option to create an extended dictionary for an existing library that you don't want to modify. For example, if you type `TLIB /E mylib` the linker appends an extended dictionary to the specified library.

If you use **/E** to add a library module containing a C++ class with a virtual function, you'll get the error message `Library contains COMDEF records--extended dictionary not created.`

Setting the page size: The /P option

Every DOS library file contains a dictionary that appears at the end of the .LIB file, following all of the object modules. For each module in the library, the dictionary contains a 16-bit address of that particular module within the .LIB file; this address is given in terms of the library page size (it defaults to 16 bytes).

The library page size determines the maximum combined size of all object modules in the library, which cannot exceed 65,536 pages. The default (and minimum) page size of 16 bytes allows a library of about 1 MB in size. To create a larger library, use the **/P** option to increase the page size. The page size must be a power of 2, and it cannot be smaller than 16 or larger than 32,768.

All modules in the library must start on a page boundary. For example, in a library with a page size of 32 (the lowest possible page size higher than the default 16), an average of 16 bytes is lost per object module in padding. If you attempt to create a library that is too large for the given page size, TLIB issues an error message and suggests that you use **/P** with the next available higher page size.

Removing comment records: The /0 option

Use the **/0** option to remove comment records, which reduces the size of a library. For example, you might have debugging or browsing information in a library, but you no longer need to use that information; the **/0** option removes that information.

The operation list

The operation list describes what actions you want TLIB to do. It consists of a sequence of operations given one after the other. Each operation consists of a one- or two-character action symbol followed by a file or module name. You can put whitespace around either the action symbol or the file or module name but not in the middle of a two-character action or in a name.

You can put as many operations as you like on the command line, up to DOS's COMMAND.COM-imposed line-length limit of 127 characters. The order of the operations is not important. TLIB always applies the operations in a specific order:

To replace a module, remove it, then add the replacement module.

1. All extract operations are done first.
2. All remove operations are done next.
3. All add operations are done last.

Wildcards are never allowed in file or module names.

TLIB finds the name of a module by stripping any drive, path, and extension information from the given file name. TLIB always assumes reasonable defaults. For example, to add a module that has an .OBJ extension from the current directory, you need to supply only the module name, not the path and .OBJ extension.

TLIB recognizes three action symbols (–, +, *), which you can use singly or combined in pairs for a total of five distinct operations. The action symbols and what they do are listed here:

Table 11.3
TLIB action symbols

To create a library, add modules to a library that does not yet exist.

You can't directly rename modules in a library. To rename a module, extract and remove it, rename the file just created, then add it back into the library.

Action symbol	Name	Description
+	**Add**	TLIB adds the named file to the library. If the file has no extension, TLIB assumes an extension of .OBJ. If the file is itself a library (with a .LIB extension), then the operation adds all of the modules in the named library to the target library. If a module being added already exists, TLIB displays a message and does not add the new module.
-	**Remove**	TLIB removes the named module from the library. If the module does not exist in the library, TLIB displays a message. A remove operation needs only a module name. TLIB lets you enter a full path name with drive and extension included, but ignores everything except the module name.
*	**Extract**	TLIB creates the named file by copying the corresponding module from the library to the file. If the module does not exist, TLIB displays a message and does not create a file. If the named file already exists, it is overwritten.
-* *-	**Extract & Remove**	TLIB copies the named module to the corresponding file name and then removes it from the library.
-+ +-	**Replace**	TLIB replaces the named module with the corresponding file.

Examples

These examples demonstrate some of the things you can do with TLIB:

- To create a library named MYLIB.LIB with modules X.OBJ, Y.OBJ, and Z.OBJ, type `tlib mylib +x +y +z`.
- To create a library named MYLIB.LIB and get a listing in MYLIB.LST too, type `tlib mylib +x +y +z, mylib.lst`.
- To replace module X.OBJ with a new copy, add A.OBJ and delete Z.OBJ from MYLIB.LIB, type `tlib mylib -+x +a -z`.
- To create a new library (ALPHA) with modules A.OBJ, B.OBJ...G.OBJ using a response file:
  - First create a text file, ALPHA.RSP, with

    ```
    +a.obj +b.obj +c.obj &
          +d.obj +e.obj +f.obj &
          +g.obj
    ```

  - Then use the TLIB command, which produces a listing file named ALPHA.LST: `tlib alpha @alpha.rsp, alpha.lst`

CHAPTER 12

Using MAKE

MAKE.EXE is a command-line project-manager utility that helps you quickly compile only those files in a project that have changed since the last compilation. (MAKER is a real-mode version of MAKE.) If you work in the IDE, you should use the IDE's project manager (see Chapter 2).

This chapter covers the following topics:

- MAKE basics
- Makefile contents
- Using explicit and implicit rules
- Using MAKE macros
- Using MAKE directives

MAKE basics

MAKE uses rules from a text file (MAKEFILE or MAKEFILE.MAK by default) to determine which files to build and how to build them. For example, you can get MAKE to compile an .EXE file if the date-time stamps for the .CPP files that contain the code for the .EXE are more recent than the .EXE itself. MAKE is very useful when you build a program from more than one file because MAKE will recompile only the files that you modified since the last compile.

Two types of rules (explicit and implicit) tell MAKE what files depend on each other. MAKE then compares the date-time stamp of the files in a rule and determines if it should execute a command (the commands usually tell MAKE which files to recompile or link, but the commands can be nearly any operating system command).

MAKE accepts * and ? as wildcards.

The general syntax for MAKE is

```
MAKE [options...] [targets[s]]
```

To get command-line help for MAKE, type `MAKE -?` or `MAKE -h`.

where `options` are MAKE options that control how MAKE works, and `targets` are the names of the files in a makefile that you want MAKE to build. Options are separated from `MAKE` by a single space. Options and targets are also separated by spaces.

If you type MAKE at the command prompt, MAKE performs the following default tasks:

To place MAKE instructions in a file other than MAKEFILE, see the section titled "MAKE options."

1. MAKE looks in the current directory for a file called BUILTINS.MAK (this file contains rules MAKE always follows unless you use the **–r** option). If it can't find the file in the current directory, it looks in the directory where MAKE.EXE is stored. After loading BUILTINS.MAK, MAKE looks for a file called MAKEFILE or MAKEFILE.MAK. If MAKE can't find any of these files, it gives you an error message.
2. When MAKE finds a makefile, it tries to build *only* the first target file in the makefile (although the first target can force other targets to be built). MAKE checks the time and date of the dependent files for the first target. If the dependent files are more recent than the target file, MAKE executes the target commands, which update the target. See the section called "Using makefiles" for more information on instructions in makefiles.
3. If a dependent file for the first target appears as a target elsewhere in the makefile, MAKE checks its dependencies and builds it before building the first target. This chain reaction is called linked dependency.
4. If the MAKE build process fails, MAKE deletes the target file it was building. To get MAKE to keep a target when a build fails, see the **.precious** directive on page 192.

You can stop MAKE by using *Ctrl+Break* or *Ctrl+C*.

BUILTINS.MAK

BUILTINS.MAK contains standard rules and macros that MAKE uses before it uses a makefile (you can use the **–r** option to tell MAKE to ignore BUILTINS.MAK). Use BUILTINS.MAK for instructions or macros you want executed each time you use MAKE. Here's the default text of BUILTINS.MAK:

```
#
# Borland C++ - (C) Copyright 1992 by Borland International
#
CC = BCC
AS = TASM
RC = RC
.asm.obj:
      $(AS) $(AFLAGS) $&.asm
.c.exe:
      $(CC) $(CFLAGS) $&.c
.c.obj:
      $(CC) $(CFLAGS) /c $&.c
.cpp.obj:
      $(CC) $(CPPFLAGS) /c $&.cpp
```

```
.rc.res:
     $(RC) $(RFLAGS) /r $&
.SUFFIXES: .exe .obj .asm .c .res .rc
```

Using TOUCH.EXE

Sometimes you'll want to force a target file to be recompiled or rebuilt even though you haven't changed it. One way to do this is to use the TOUCH utility. TOUCH changes the date and time of one or more files to the current date and time, making it "newer" than the files that depend on it.

You can force MAKE to rebuild a target file by *touching* one of the files that target depends on. To touch a file (or files), type the following at the command prompt:

You can use wildcards * and ? with TOUCH.

```
touch filename [filename...]
```

TOUCH updates the file's creation date and time.

Important! Before you use TOUCH, make sure your system's internal clock is set correctly. If it isn't, TOUCH and MAKE won't work properly.

MAKE options

Command-line options control MAKE behavior. Options are case-sensitive. Type options with either a preceding – or /. For example, to use a file called PROJECTA.MAK as the makefile, type `MAKE -fPROJECTA.MAK` (a space after **–f** is optional). Many of the command-line options have equivalent directives that are used in the makefile (see page 188 for more information on directives).

Table 12.1: MAKE options

Option	Description
–h or –?	Displays MAKE options and shows defaults with a trailing plus sign.
–B	Builds all targets regardless of file dates.
–D*macro*	Defines *macro* as a single character, causing an expression **!ifdef** *macro* written in the makefile to return true.
[-D]*macro*=[*string*]	Defines *macro* as *string*. If *string* contains any spaces or tabs, enclose *string* in quotation marks. The **–D** is optional.
–I*directory*	Searches for include files in the current directory first, then in *directory*.
–K	Keeps temporary files that MAKE creates (MAKE usually deletes them). See also KEEP on page 179.
–N	Executes MAKE like Microsoft's NMAKE (see the section following this table for more information).
–U*macro*	Undefines previous definitions of *macro*.
–W	Writes the current specified non-string options to MAKE.EXE making them defaults.
–f*filename*	Uses *filename* or *filename*.MAK instead of MAKEFILE (space after **–f** is optional).

Table 12.1: MAKE options (continued)

–a	Checks dependencies of include files and nested include files associated with .OBJ files and updates the .OBJ if the .H file changed. See also **–c**.
–c	Caches autodependency information, which can improve MAKE's speed. Use with –a; don't use if MAKE changes include files (such as using TOUCH from a makefile or creating header or include files during the MAKE process).
–d*directory*	Used with **–S** to specify the drive and directory MAKE uses when it swaps out of memory. The option is ineffective when used with the MAKER.
–e	Ignores a macro if its name is the same as an environment variable (MAKE uses the environment variable instead of the macro).
–i	Ignores the exit status of all programs run from MAKE and continues the build process.
–m	Displays the date and time stamp of each file as MAKE processes it.
–n	Prints the commands but doesn't actually perform them, which is helpful for debugging a makefile.
–p	Displays all macro definitions and implicit rules before executing the makefile.
–q	Returns 0 if the target is up-to-date and nonzero if is is not (for use with batch files).
–r	Ignores any rules defined in BUILTINS.MAK.
–s	Suppresses onscreen command display.
–S	Swaps MAKER out of memory while commands are executed, reducing memory overhead and allowing compilation of large modules. This option has no effect on MAKER.

Setting options on as defaults

The **–W** option lets you set some MAKE options on as defaults so that each time you use MAKE, those options are used. To set MAKE options, type

```
make  -option[-]  [-option][-]  . . . -W
```

For example, you could type `MAKE -m -W` to always view file dates and times. Type `MAKE -m- -W` to turn off the default option. When MAKE asks you to write changes to MAKE.EXE, type `Y`.

Caution! The **–W** option doesn't work when the DOS Share program is running. The message `Fatal: unable to open file MAKE.EXE` is displayed. The **–W** option doesn't work with the following MAKE options:

- **–D***macro*
- **–D***macro=string*
- **–d***directory*
- **–U***symbol*
- **–f***filename*
- **–?** or **–h**
- **–I***directory*

Compatibility with Microsoft's NMAKE

Use the **–N** option if you want to use makefiles that were originally created for Microsoft's NMAKE. The following changes occur when you use **–N**:

- MAKE interprets the **<<** operator like the **&&** operator: temporary files are used as response files, then deleted. To keep a file, either use the **–K** command-line option or use KEEP in the makefile.

  MAKE usually deletes temporary files it creates.

  ```
  <<TEMPFILE.TXT!
  text
  :
  !KEEP
  ```

  If you don't want to keep a temporary file, type NOKEEP or type only the temporary file name. If you use NOKEEP with a temporary file, then use the **–K** option with MAKE, MAKE deletes the temporary file.
- The **$d** macro is treated differently. Use **!ifdef** or **!ifndef** instead.
- Macros that return paths won't return the last \. For example, if `$(<D)` normally returns `C:\CPP\`, the **–N** option makes it return `C:\CPP`.
- Unless there's a matching **.suffixes** directive, MAKE searches rules from bottom to top of the makefile.
- The **$*** macro always expands to the target name instead of the dependent in an implicit rule.

Using makefiles

A makefile is an ASCII file of instructions for MAKE.EXE. MAKE assumes your makefile is called MAKEFILE or MAKEFILE.MAK unless you use the **–f** option (see page 177).

MAKE either builds targets you specify at the MAKE command line or it builds *only* the first target it finds in the makefile (to build more than one target, see the section "Symbolic targets.") Makefiles can contain:

- Comments
- Explicit rules
- Implicit rules
- Macros
- Directives

Symbolic targets

A symbolic target forces MAKE to build multiple targets in a makefile (you don't need to rely on linked dependencies). The dependency line lists all the targets you want to build. You don't type any commands for a symbolic target.

In the following makefile, the symbolic target *allFiles* builds both FILE1.EXE and FILE2.EXE.

```
allFiles: file1.exe file2.exe     #Note this target has no commands.
file1.exe: file1.obj
   bcc file1.obj
file2.exe: file2.obj
   bcc file2.obj
```

Rules for symbolic targets

Observe the following rules with symbolic targets:

- Symbolic targets don't need a command line.
- Give your symbolic target a unique name; it can't be the name of a file in your current directory.
- Name symbolic targets according to the operating system rules for naming files.

Explicit and implicit rules

The explicit and implicit rules that instruct MAKE are generally defined as follows:

- Explicit rules give MAKE instructions for specific files.
- Implicit rules give general instructions that MAKE follows when it can't find an explicit rule.

Rules follow this general format:

```
Dependency line
   Commands
   :
```

The dependency line is different for explicit and implicit rules, but the commands are the same (for information on linked dependencies see page 176).

MAKE supports multiple rules for one target. You can add dependent files after the first explicit rule, but only one should contain a command line. For example,

```
Target1: dependent1 dep2 dep3 dep4 dep5
Target1: dep6 dep7 dep8
  bcc -c $**
```

Explicit rule syntax

Explicit rules are instructions to MAKE that specify exact file names. The explicit rule names one or more targets followed by one or two colons. One colon means one rule is written for the target; two colons mean that two or more rules are written for the target.

Explicit rules follow this syntax:

Braces must be included if you use the paths parameter.

```
target [target...]:[:][{path}] [dependent[s]...]
   [commands]
      ⋮
```

- target — The name and extension of the file to be updated (*target* must be at the start of the line—no spaces or tabs are allowed). One or more targets must be separated by spaces or tabs. Don't use a target's name more than once in the target position of an explicit rule in a makefile.
- path — A list of directories, separated by semicolons and enclosed in braces, that points to the dependent files.
- dependent — The file (or files) whose date and time MAKE checks to see if it is newer than `target` (dependent *must* be preceded by a space). If a dependent file also appears in the makefile as a target, MAKE updates or creates the target file before using it as a dependent for another target.
- commands — Any operating system command. Multiple commands are allowed in a rule. Commands must be indented by at least one space or tab (see the section on commands on page 183).

If the dependency or command continues on to the next line, use the backslash (\) at the end of the line after a target or a dependent file name. For example:

```
MYSOURCE.EXE: FILE1.OBJ\
              FILE2.OBJ\
              FILE3.OBJ
     bcc file1.obj file2.obj file3.obj
```

Single targets with multiple rules

A single target can have more than one explicit rule. You must use the double colon :: after the target name to tell MAKE to expect multiple explicit rules. The following example shows how one target can have multiple rules and commands.

```
.cpp.obj:
   bcc -c -ncobj $<

.asm.obj:
   tasm  /mx $<, asmobj\\

mylib.lib :: f1.obj f2.obj
   echo Adding C files
   tlib mylib -+cobj\f1 -+cobj\f2

mylib.lib :: f3.obj f4.obj
   echo Adding ASM files
   tlib mylib -+asmobj\f3 -+asmobj\f4
```

Implicit rule syntax

An implicit rule starts with either a path or a period and *implies* a target-dependent file relationship. Its main components are file extensions separated by periods. The first extension belongs to the dependent, the second to the target.

If implicit dependents are out-of-date with respect to the target or if they don't exist, MAKE executes the commands associated with the rule. MAKE updates explicit dependents before it updates implicit dependents.

Implicit rules follow this basic syntax:

```
[{source_dirs}].source_ext[{target_dirs}].target_ext:
     [commands]
```

■ {source_dirs}	The directory of the dependent files. Separate multiple directories with a semicolon.
■ .source_ext	The dependent file-name extension.
■ {target_dirs}	The directory of the target (executable) files. Separate multiple directories with a semicolon.
■ .target_ext	The target file-name extension. Macros are allowed here.
■ :	Marks the end of the dependency line.
■ commands	Any operating system command. Multiple commands are allowed. Commands must be indented by one space or tab (see the section on commands on page 183).

If two implicit rules match a target extension but no dependent exists, MAKE uses the implicit rule whose dependent's extension appears first in the .SUFFIXES list. See the ".suffixes" section on page 192.

Explicit rules with implicit commands

A target in an explicit rule can get its command line from an implicit rule. The following example shows an implicit rule and an explicit rule without a command line.

```
.c.obj:
     bcc -c $<    #This command uses a macro $< described later.

myprog.obj:       #This explicit rule uses the command: bcc -c myprog.c
```

See page 186 for information on default macros.

The implicit rule command tells MAKE to compile MYPROG.C (the macro **$<** replaces the name `myprog.obj` with `myprog.c`).

Commands syntax

Commands can be any operating system command, but they can also include MAKE macros, directives, and special operators that operating systems can't recognize (note that | can't be used in commands). Here are some sample commands:

```
cd..

bcc -c mysource.c

COPY *.OBJ C:\PROJECTA

bcc -c $(SOURCE)     #Macros are explained later in the chapter.
```

Commands follow this general syntax:

```
[prefix...] commands
```

Command prefixes

Commands in both implicit and explicit rules can have prefixes that modify how MAKE treats the commands. Table 12.2 lists the prefixes you can use in makefiles; each prefix is explained in more detail following the table.

Table 12.2
Command prefixes

Option	**Description**
@	Don't display *command* while it's being executed.
–*num*	Stop processing commands in the makefile when the exit code returned from *command* exceeds *num*. Normally, MAKE aborts if the exit code is nonzero. No white space is allowed between – and *num*.
–	Continue processing commands in the makefile, regardless of the exit code returned by them.
&	Expand either the macro **$****, which represents all dependent files, or the macro **$?**, which represents all dependent files stamped later than the target. Execute the command once for each dependent file in the expanded macro.

Using @

The following command uses the modifier @, which prevents the command from displaying onscreen when MAKE executes it.

```
diff.exe : diff.obj
   @bcc diff.obj
```

Using –num and –

The **–num** and – modifiers control MAKE processing under error conditions. You can choose to continue with the MAKE process if an error occurs or only if the errors exceed a given number.

In the following example, MAKE continues processing if BCC isn't run successfully:

```
target.exe : target.obj
target.obj : target.cpp
  bcc -c target.cpp
```

Using &

The **&** modifier issues a command once for each dependent file. It is especially useful for commands that don't take a list of files as parameters. For example,

```
copyall : file1.cpp file2.cpp
   &copy $** c:\temp
```

results in COPY being invoked twice as follows:

```
copy file1.cpp c:\temp
copy file2.cpp c:\temp
```

Without the **&** modifier, COPY would be called only once.

Command operators

You can use any operating system command in a MAKE commands section. MAKE uses the normal operators (such as +, –, and so on), but it also has other operators you can use.

Table 12.3
Command operators

Operator	Description
<	Take the input for use by *command* from *file* rather than from standard input.
>	Send the output from *command* to *file.*
>>	Append the output from *command* to *file.*
<<	Create a temporary, inline file and use its contents as standard input to *command.*
&&	Create a temporary file and insert its name in the makefile.
delimiter	Any character other than # and \ used with **<<** and **&&** as a starting and ending delimiter for a temporary file. Any characters on the same line and immediately following the starting delimiter are ignored. The closing *delimiter* must be written on a line by itself.

Debugging with temporary files

Temporary files can help you debug a command set by placing the actual commands MAKE executes into the temporary file. Temporary file names start at MAKE0000.@@@, where the 0000 increments for each temporary file you keep. You must place delimiters after **&&** and at the end of what you want sent to the temporary file (! is a good delimiter).

The following example shows **&&** instructing MAKE to create a file of the input to TLINK.

```
prog.exe: A.obj B.obj
```

```
TLINK /c &&!
c0s.obj $**
prog.exe
prog.map
maths.lib cs.lib
!
```

The response file created by **&&** contains these instructions:

```
c0s.obj a.obj b.obj
prog.exe
prog.map
maths.lib cs.lib
```

Using MAKE macros

Macros are case-sensitive: MACRO1 is different from Macro1.

A MAKE macro is a string that is expanded (used) wherever the macro is called in a makefile. Macros let you create template makefiles that you can change to suit different projects. For example, to define a macro called LIBNAME that represents the string "mylib.lib," type `LIBNAME = mylib.lib`. When MAKE encounters the macro `$(LIBNAME)`, it uses the string `mylib.lib`.

If MAKE finds an undefined macro in a makefile, it looks for an operating-system environment variable of that name (usually defined with SET) and uses its definition as the expansion text. For example, if you wrote `$(path)` in a makefile and never defined *path*, MAKE would use the text you defined for PATH in your AUTOEXEC.BAT. (See the manuals for your operating system for information on defining environment variables.)

Defining macros

The general syntax for defining a macro in a makefile is `MacroName = expansion_text`.

- *MacroName* is case-sensitive and is limited to 512 characters.
- *expansion_text* is limited to 4096 characters consisting of alphanumeric characters, punctuation, and white space.

Each macro must be on a separate line in a makefile. Macros are usually put at the top of the makefile. If MAKE finds more than one definition for a *macroName*, the new definition replaces the old one.

Macros can also be defined using the command-line option **–D** (see page 177). More than one macro can be defined by separating them with spaces. The following examples show macros defined at the command line:

```
make -Dsourcedir=c:\projecta
make command="bcc -c"
make command=bcc option=-c
```

The following differences in syntax exist between macros entered on the command line and macros written in a makefile.

Table 12.4 Command line vs. makefile macros

Syntax	Makefile	Command line
Spaces allowed before and after =	Yes	No
Space allowed before *macroName*	No	Yes

Using a macro

To use a macro in a makefile, type `$(MacroName)` where MacroName is the name of a defined macro. You can use braces {} and parentheses () to enclose the MacroName.

MAKE expands macros at various times depending on where they appear in the makefile:

- Nested macros are expanded when the outer macro is invoked.
- Macros in rules and directives are expanded when MAKE first looks at the makefile.
- Macros in commands are expanded when the command is executed.

String substitutions in macros

MAKE lets you temporarily substitute characters in a previously defined macro. For example, if you defined a macro called SOURCE as `SOURCE = f1.cpp f2.cpp f3.cpp`, you could substitute the characters .OBJ for the characters .CPP by using `$(SOURCE:.CPP=.OBJ)`. The substitution doesn't redefine the macro.

Rules for macro substitution:

- Syntax: `$(MacroName:original_text=new_text)`.
- No whitespace before or after the colon.
- Characters in *original_text* must exactly match the characters in the macro definition; this text is case-sensitive.

MAKE now lets you use macros within substitution macros. For example,

```
MYEXT=.C
SOURCE=f1.cpp f2.cpp f3.cpp
$(SOURCE:.cpp=$(MYEXT))        #Changes f1.cpp to f1.C, etc.
```

Default MAKE macros

MAKE contains several default macros you can use in your makefiles. Table 12.5 lists the macro definition and what it expands to in explicit and implicit rules.

Table 12.5: Default macros

Macro	Expands in implicit:	Expands in explicit:	Example
$*	path\dependent file	path\target file	C:\PROJECTA\MYTARGET
$<	path\dependent file+ext	path\target file+ext	C:\PROJECTA\MYTARGET.OBJ
$:	path for dependents	path for target	C:\PROJECTA
$.	dependent file+ext	target file + ext	MYSOURCE.C
$&	dependent file	target file	MYSOURCE
$@	path\target file+ext	path\target file+ext	C:\PROJECTA\MYSOURCE.C
$**	path\dependent file+ext	all dependents file+ext	FILE1.CPP FILE2.CPP FILE3.CPP
$?	path\dependent file+ext	old dependents	FILE1.CPP

Table 12.6
Other default macros

Macro	Expands to:	Comment
__MSDOS__	1	If running under DOS.
__MAKE__	0x0370	MAKE's hex version number.
MAKE	make	MAKE's executable file name.
MAKEFLAGS	options	The options typed at the command line.
MAKEDIR	directory	Directory where MAKE.EXE is located.

Modifying default macros

When the default macros listed in Table 12.5 don't give you the exact string you want, macro modifiers let you extract parts of the string to suit your purpose.

To modify a default macro, use this syntax:

```
$(MacroName [modifier])
```

Table 12.7 lists macro modifiers and provides examples of their use.

Table 12.7
File-name macro modifiers

Modifier	Part of file name expanded	Example	Result
D	Drive and directory	$(<D)	C:\PROJECTA\
F	Base and extension	$(<F)	MYSOURCE.C
B	Base only	$(<B)	MYSOURCE
R	Drive, directory, and base	$(<R)	C:\PROJECTA\MYSOURCE

Using MAKE directives

MAKE directives resemble directives in languages such as C and Pascal, and perform various control functions, such as displaying commands onscreen before executing them. MAKE directives begin either with an exclamation point or a period. Table 12.8 lists MAKE directives and their corresponding command-line options (directives override command-line options). Each directive is described in more detail following the table.

Table 12.8
MAKE directives

Directive	Option	Description
.autodepend	–a	Turns on autodependency checking.
!elif		Acts like a C **else if**.
!else		Acts like a C **else**.
!endif		Ends an **!if**, **!ifdef**, or **!ifndef** statement.
!error		Stops MAKE and prints an error message.
!if		Begins a conditional statement.
!ifdef		If defined that acts like a C **ifdef**, but with macros rather than **#define** directives.
!ifndef		If not defined.
.ignore	–i	MAKE ignores the return value of a command.
!include		Specifies a file to include in the makefile.
!message		Lets you print a message from a makefile.
.noautodepend	–a-	Turns off autodependency checking.
.noIgnore	–i-	Turns off **.Ignore**.
.nosilent	–s-	Displays commands before MAKE executes them.
.noswap	–S-	Tells MAKE not to swap itself out of memory before executing a command.
.path.ext		Tells MAKE to search for files with the extension .ext in *path* directories.
.precious		Saves the target or targets even if the build fails.
.silent	–s	Executes without printing the commands.
.suffixes		Determines the implicit rule for ambiguous dependencies.
.swap	–S	Tells MAKE to swap itself out of memory before executing a command.
!undef		Clears the definition of a macro.

.autodepend

Autodependencies occur in .OBJ files that have corresponding .CPP, .C, or .ASM files. With **.autodepend** on, MAKE compares the dates and times of all the files used to build the .OBJ. If the dates and times of the files used to build the .OBJ are different from the date-time stamp of the.OBJ file, the .OBJ file is recompiled. You can use **.autodepend** or **–a** in place of linked dependencies (see page 176 for information on linked dependencies).

!error

This is the syntax of the **!error** directive:

```
!error message
```

MAKE stops processing and prints the following string when it encounters this directive:

```
Fatal makefile exit code: Error directive: message
```

Embed **!error** in conditional statements to abort processing and print an error message, as shown in the following example:

```
!if !$d(MYMACRO)
#if MYMACRO isn't defined
!error MYMACRO isn't defined
!endif
```

If MYMACRO in the example isn't defined, MAKE prints the following message:

```
Fatal makefile 4: Error directive: MYMACRO isn't defined
```

Summing up error-checking controls

Four different controls turn off error checking:

- The **.ignore** directive turns off error checking for a selected portion of the makefile.
- The **–i** command-line option turns off error checking for the entire makefile.
- The ***–num*** command operator, which is entered as part of a rule, turns off error checking for the related command if the exit code exceeds the specified number.
- The – command operator turns off error checking for the related command regardless of the exit code.

!if and other conditional directives

The **!if** directive works like C **if** statements (see the *Programmer's Guide* if you don't understand **if** statements). As shown here, the syntax of **!if** and the other conditional directives resembles compiler conditionals:

```
!if condition      !if condition      !if condition      !ifdef macro
     :                  :                  :                  :
!endif             !else              !elif condition    !endif
                        :                  :
                   !endif             !endif
```

The following expressions are equivalent:
`!ifdef` *macro* and `!if $d(`*macro*`)`
`!ifndef` *macro* and `!if !$d(`*macro*`)`

These rules apply to conditional directives:

- One **!else** directive is allowed between **!if**, **!ifdef**, or **!ifndef** and **!endif** directives.
- Multiple **!elif** directives are allowed between **!if**, **!ifdef**, or **!ifndef** and **!else** directives and **!endif**.
- You can't split rules across conditional directives.
- You can nest conditional directives.
- **!if**, **!ifdef**, and **!ifndef** must have matching **!endif** directives within the same source file.

The following information can be included between **!if** and **!endif** directives:

- Macro definition
- Explicit rule
- Implicit rule
- **!include** directive
- **!error** directive
- **!undef** directive

Condition in **if** statements represents a conditional expression consisting of decimal, octal, or hexadecimal constants and the operators shown in Table 12.9.

Table 12.9
Conditional operators

Operator	Description	Operator	Description
–	Negation	?:	Conditional expression
~	Bit complement	!	Logical NOT
+	Addition	>>	Right shift
–	Subtraction	<<	Left shift
*	Multiplication	&	Bitwise AND
/	Division	|	Bitwise OR
%	Remainder	^	Bitwise XOR
&&	Logical AND	>=	Greater than or equal*
||	Logical OR	<=	Less than or equal*
>	Greater than	==	Equality*
<	Less than	!=	Inequality*

*Operator also works with string expressions.

MAKE evaluates a conditional expression as either a simple 32-bit signed integer or as a character string.

!include

This directive is like the **#include** preprocessor directive for the C or C++ language—it lets you include the text of another file in the makefile:

```
!include filename
```

You can enclose *filename* in quotation marks ("") or angle brackets (<>) and nest directives to unlimited depth, but writing duplicate **!include** directives in a makefile isn't permitted—you'll get the error message *cycle in the include file.*

Rules, commands, or directives must be complete within a single source file; you can't start a command in an **!include** file, then finish it in the makefile.

MAKE searches for **!include** files in the current directory unless you've specified another directory with the **–I** option.

!message

The **!message** directive lets you send messages to the screen from a makefile. You can use these messages to help debug a makefile that isn't working the way you'd like it to. For example, if you're having trouble with a macro definition, you could put this line in your makefile:

```
!message The macro is defined here as: $(MacroName)
```

When MAKE interprets this line, it will print onscreen `The macro is defined here as: .CPP`, if the macro expands to .CPP at that line. Using a series of **!message** directives, you can debug your makefiles.

.path.ext

The **.path.ext** directive tells MAKE where to look for files with a certain extension. The following example tells MAKE to look for files with the .c extension in C:\SOURCE or C:\CFILES and to look for files with the .obj extension in C:\OBJS.

```
.path.c = C:\CSOURCE;C:\CFILES
.path.obj = C:\OBJS
```

.precious

If a MAKE build fails, MAKE deletes the target file. The **.precious** directive prevents the file deletion, which is desired for certain kinds of targets such as libraries. When a build fails to add a module to a library, you don't want the library to be deleted.

The syntax for **.precious** is:

```
.precious: target [target] . . . [target]
```

.suffixes

The **.suffixes** directive tells MAKE the order (by file extensions) for building implicit rules.

The syntax of the **.suffixes** directive is:

```
.suffixes: .ext [.ext]  [.ext] . . . [.ext]
```

.ext represents the dependent file extension in implicit rules. For example, you could include the line `.suffixes: .asm .c .cpp` to tell MAKE to interpret implicit rules beginning with the ones dependent on .ASM files, then .C files, then .CPP files, regardless of what order they appear in the makefile.

The following example shows a makefile containing a **.suffixes** directive that tells MAKE to look for a source file (MYPROG.EXE) first with an .ASM extension, next with a .C extension, and finally with a .CPP extension. If MAKE finds MYPROG.ASM, it builds MYPROG.OBJ from the assembler file by calling TASM. MAKE then calls TLINK; otherwise, MAKE searches for MYPROG.C to build the .OBJ file, and so on.

```
.suffixes: .asm .c .cpp

myprog.exe: myprog.obj
tlink myprog.obj

.cpp.obj:
  bcc -P $<
.asm.obj:
   tasm /mx $<
.c.obj:
   bcc -P- $<
```

!undef

The syntax of the **!undef** directive is:

```
!undef MacroName
```

!undef (undefine) clears the given macro, *MacroName*, causing an **!ifdef** *MacroName* test to fail.

Using macros in directives

The macro **$d** is used with the **!if** conditional directive to perform some processing if a specific macro is defined. The **$d** is followed by a macro name, enclosed in parentheses or braces, as shown in the following example.

```
!if $d(DEBUG)               #If DEBUG is defined,
bcc -v f1.cpp f2.cpp        #compile with debug information;
!else                       #otherwise (else)
bcc -v- f1.cpp f2.cpp       #don't include debug information.
!endif
```

Caution! Don't use the **$d** macro when MAKE is invoked with the **–N** option.

Null macros

An undefined macro causes an **!ifdef** *MacroName* test to return false; a null macro returns true. A null macro is a macro defined with either spaces to the right of the equal sign (=) or no characters to the right of the equal sign. For example, the following line defines a null macro in a makefile:

```
NULLMACRO =
```

One of the following lines can define a null macro on the MAKE command line:

```
NULLMACRO=""
```

or

```
-DNULLMACRO
```

P A R T II

Using Resource Workshop

This section of the *User's Guide* describes how to use Resource Workshop, a tool that integrates the entire process of designing and compiling resources for applications running under Microsoft Windows, Version 3.0 and later. Resource Workshop,

- Works with resources in either text or binary format. It includes graphics-oriented editors that let you edit binary files and a text editor that lets you edit the files as resource scripts.
- Makes it easy to manage hundreds of resources stored in dozens of files.
- Includes multilevel Undo and Redo features that let you step back through changes you've made.
- Includes all the compilers you need and lets you compile your resources only when you need to.
- Decompiles binary resource files, so you can change a program's resources even if you don't have access to the source code.
- Includes features that automatically check for errors, making it easy to test resources for errors like incorrect syntax and duplicate resource IDs.

There are four online files for Resource Workshop. These files include advanced information that most users don't need.

■ BWCCAPI.TXT	Describes the BWCC Application Program Interface (API).
■ CUSTCNTL.TXT	Describes how to create custom control classes. It includes sample code for C++ and Pascal. You should be familiar with this material if you plan to create your own custom control classes.
■ BWCCSTYL.TXT	Describes some of the considerations that went into designing Borland-style dialog boxes with Borland Windows Custom Controls.
■ RWINI.TXT	Describes options you can set only in the WORKSHOP.INI file.

C H A P T E R 13

Resource Workshop basics

This chapter provides an overview of Resource Workshop and Windows resources. It explains resources and resource types. It also covers how to use the different types of resource files in Resource Workshop and how these files fit together in a project.

Understanding Windows resources

Resources define the visible portions of your Windows program. For example, when you open a dialog box and click a button to accomplish a task with a program, you're interacting with that program's resources. In addition to dialog boxes and buttons, other types of resources you can use in your Windows programs include icons, cursors, bitmaps, menus, and keyboard accelerators.

In general, resources in a Windows application are separate from the program code, letting you make significant changes to the user interface without even opening the file that contains the program code.

Also, because different applications can use the same set of resources, you don't have to reinvent all your favorite dialog boxes, icons, and customized cursors. Instead, you can use them over and over.

Resource Workshop supports the following types of Windows resources:

- *Dialog boxes.* A window (usually a popup window) that communicates information to the user and lets the user select choices, such as files to open, colors to display, text to search for, and so on.
- *Menus.* Windows programs usually include a menu bar that lists the names of individual menus. A typical menu contains one or more menu items (commands). For example, most Windows programs have a File menu with commands for creating, opening, saving, or printing files.
- *Accelerators.* Keyboard combinations (or hot keys) that a user presses to perform a task in an application. For example, a Windows program can include the accelerator *Shift+Ins* for the Paste command, which the user presses to paste text or images from the Clipboard into a file the program

has open. Accelerators typically appear in the menu to the right of the commands to which they're linked.

- *String tables.* These tables contain text (like descriptions, prompts, and error messages) that's displayed as part of a Windows program. Because these text strings are Windows resources that are separate from the program (instead of strings embedded in the program), you or others can edit and translate messages displayed by a program without having to make any changes to the program's source code.
- *Bitmaps.* A binary representation of a graphic image in a program. Windows itself uses lots of bitmaps. For example, the images representing various controls on a typical window, such as scroll bar arrows, the Control-menu symbol, and the Minimize symbol, are all bitmaps. Each bit, or group of bits, in the bitmap represents one pixel of the image.
- *Icons.* Small bitmaps—64×64 (supported in Windows 3.1 on very high-resolution devices), 32×32, or 16×32 pixels in size—that represent minimized windows. You create icons using the Bitmap editor (described in Chapter 19).
- *Cursors.* Small bitmaps, 32×32 pixels in size, that represent the position of the mouse on the screen. Windows programs use customized cursors to indicate what type of task the user is currently performing. You create cursors using the Bitmap editor (see Chapter 19).

  You can see an example of customized cursors in the Resource Workshop Bitmap editor, which appears if you edit a bitmapped resource (bitmap, icon, cursor, or font). Each time you choose a new paint tool and move the cursor to the image you're working on, the cursor takes a shape that represents its current function.
- *Fonts.* Although *font* usually refers to a set of text characters, in Resource Workshop it most commonly refers to a set of bitmaps that are always stored and used together.

  Windows programs use text fonts to define the typeface, size, and style of text. For example, a font that a program can display onscreen or print on a printer is **10-point Times Roman bold**. In this case, the *typeface* is Times Roman, the *size* is 10 points, and the *style* is bold.

  You can use Resource Workshop to modify the way existing fonts appear or to create your own fonts.
- *User-defined and rcdata resources.* These resources are essentially the same in Resource Workshop. They consist of any data you want to add to your executable file. For example, if you have a large block of initialized, read-only data, such as a text file, you can add it to your executable file as a user-defined resource.

One common reason for adding user-defined resources to an application is to help manage memory. Many Windows applications use the medium memory model, which includes a single data segment. If you have a relatively large amount of data and you don't want that data to take up permanent residence in memory, you can save the data as a user-defined resource defined to be discardable data. The user-defined resource then takes up memory only when your program needs to use it.

- *Version information.* Resource Workshop supports the VERSIONINFO resource, the version stamper for Windows 3.1 .EXE files. For a detailed description of VERSIONINFO, see the Resource Workshop online Help files or your Windows 3.1 programming materials.

For most resource types, Resource Workshop gives you a choice: a powerful, easy-to-use, graphics-oriented resource editor, or a text editor for writing resource scripts. This choice makes Resource Workshop accessible to all levels of Windows programmers.

In addition to its own graphics and text editors, Resource Workshop lets you specify an external editor to use for editing project files that won't compile when they're being opened.

Types of resource files

A file you create and edit with Resource Workshop can be in either binary or text format. In addition, Resource Workshop can generate standard Windows file formats, which means you can use Resource Workshop files with programs that generate binary code from resource script files.

- *Resource compiler script files.* A resource compiler (.RC) file is a resource script (text) file containing definitions of one or more resources. The file can contain resources defined in script form and references to other files containing resources.

  In general, you should base all your Resource Workshop projects on at least one .RC file.
- *Compiled resource files.* A compiled resource (.RES) file contains one or more compiled resources.

  Typically, when creating a Windows program, you compile all resources for an application into a single .RES file, and then you bind the .RES file to the executable file as part of the linking process. However, with Resource Workshop if you don't want to produce a .RES file, you don't have to, because Resource Workshop can compile resource files and bind them directly to an executable file.

- *Executable and dynamic-link library files.* An executable (.EXE) or dynamic-link library (DLL) file is the ultimate destination for all resources you define with Resource Workshop. Usually, you compile an .RC file into a .RES file, then use your compiler to bind the .RES file to the executable or DLL file. You can also use Resource Workshop to bind the resources directly to the executable or DLL file and bypass the Microsoft Resource Compiler altogether.
  - *.DRV files.* A .DRV file is a Windows device driver, a special case of a DLL. You can edit the resources in a .DRV file the same way you can edit resources in any DLL.

  If you want to change the resources in a compiled binary file (an executable file, a DLL file, or a .RES file), Resource Workshop will decompile the file and let you make changes, and then save the resources back to the original binary file.
- *Dialog files.* A dialog (.DLG) file is a resource script (text) file that typically contains descriptions of one or more dialog boxes. There is, however, no requirement that a .DLG file contain dialog boxes; it can contain any of the resources found in an .RC file.

Bitmapped resource files

There are four kinds of bitmapped resource files:

- A bitmap (.BMP) file contains a bitmap resource in binary format.
- An icon (.ICO) file contains an icon in binary format.
- A cursor (.CUR) file contains a customized cursor in binary format.
- Font files take two forms, binary and font library.
  - A binary font (.FNT) file contains the definition of a customized font in binary format. You can use the Resource Workshop Bitmap editor to design a font and store it in an .FNT file.
  - A font library (.FON) file is a resource-only dynamic-link library that contains a font directory and one or more fonts. You must create .FON files outside Resource Workshop. However, once you've created an .FON file, you can use Resource Workshop to modify the file.

CHAPTER 14

Working with projects, resources, and identifiers

This chapter describes the Project window and explains how to control the display of resources in that window and how to choose resources from that window for editing. It also explains how to add and delete resources from projects, how to save them, and how to work with identifiers.

A *project* is a collection of one or more resources. A project file (typically .RC) either contains one or more resources or references files containing resources, or both.

Creating a new project

To create a new project,

If you have a project open, Resource Workshop closes it first. If there are unsaved changes, Resource Workshop asks if you want to save those changes before closing.

1. Choose File | New project.

   Decide what type of file you want to base your project on. A typical project is based on an .RC file, because this type of file lets you work with all kinds of resources. However, you can also choose one of the following:

   - .RC, to create a resource script file
   - .RES, to work with a binary resource file
   - .CUR, to create a project containing a cursor
   - .ICO, to create a project containing an icon
   - .BMP, to create a project containing a bitmap
   - .FNT, to create a project containing only fonts

2. Click the project file type you want, then click OK.

   Resource Workshop displays your new, untitled project in the Project window. For .RC projects, you'll need to specify a default header file where Resource Workshop stores identifiers.

Opening an existing project

An existing project can be one that you created with Resource Workshop or an .RC file you created with other resource development software. You can also work with the resources in any application developed for Windows 3.0 or higher, even if you don't have access to the source code. If you have access only to an executable file, Resource Workshop can decompile the resources bound to that file so that you can make changes to them.

To open an existing project,

1. Choose File | Open Project. Resource Workshop displays the Open Project dialog box.
2. Specify the file you want to open by doing either of the following:

   - Type the path and file name, then press *Enter.*
   - Choose a file type, then choose a file from the Files list and click OK.

   What Resource Workshop does next depends on whether the project is a binary file or a file containing resource data.

   - If the project is a binary file (an executable file, a .RES file, or a DLL file), Resource Workshop decompiles the resources and shows you its progress on the left side of the status bar at the bottom of the display.
   - If the project is an .RC file or other file containing resource data (as is usually the case), Resource Workshop reads the file directly (since decompiling isn't necessary), then it compiles the file and each resource, showing you its progress in the Compile Status dialog box.

     If the compiler encounters an error, Resource Workshop displays the Compiler Error dialog box, which shows you the error and highlights the line where the error occurred.

     To edit the file using the external text editor specified in File | Preferences, press the Run Editor button in the Compiler Error dialog box. When the editor appears, make your changes, then save your changes and exit the editor. You must then reload the project.
3. Once the project is compiled or decompiled, Resource Workshop displays the Project window with all the resources listed in it.

Using the Project window

Once you've opened a new or existing project, Resource Workshop displays the Project window.

The Project window acts as a file management tool, making it easy to get an overall view of a project. Even if the project contains a large number of resources, you can quickly scan through them by scrolling through the Project window.

For a new project, the window is empty, and you have to put resources into it by creating them or adding them as files (more on these subjects later).

For an existing project, you can see

- The complete list of files in the project.
- The types of resources contained in each file.
- If the file contains resource data (is not an .EXE, .RES, or .DLL file), the identifiers (#**define**s or constants) associated with the resources. Identifiers are discussed on page 210.

Embedded and linked resources

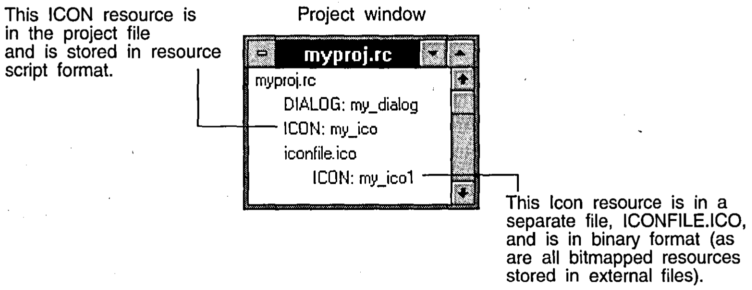
The resources in your project file can be *embedded* in the file or *linked* to it.

- An embedded resource is stored in resource script form in the project file. It exists only as part of the project in which it's stored, and it can't be used in other projects.
- A linked resource is a separate file that is referenced in the project file. Linked resources such as .RC, .DLG, or binary-format can be used in other projects.

Displaying the Project window

Use the View menu to determine how the Project window displays information. By default, the Project window displays the selected resource in a preview window. Use View | Hide preview to turn off the preview (preview is off in Figure 14.1). You can also set where the preview displays in relation to the project outline by choosing View | Show vertical preview or View | Show horizontal preview.

Figure 14.1
Project window showing resources by file


This ICON resource is in the project file and is stored in resource script format.
Project window
myproj.rc
myproj.rc
DIALOG: my_dialog
ICON: my_ico
iconfile.ico
ICON: my_ico1
This Icon resource is in a separate file, ICONFILE.ICO, and is in binary format (as are all bitmapped resources stored in external files).

Show Identifiers

Displays any identifiers (#**define**s or constants) in the project. Identifiers are discussed on page 210.

Show Resources

Lists the types and names of resources, like BITMAP: airplane. In most cases you'll leave this option checked.

Uncheck the Show Resources option if you want to see file names only without a list of resources contained in those files, or if you're just interested in looking at identifiers. With Show Resources off, none of the resources defined in the project file can be selected for editing.

Show Items

Shows another level of detail in the Project window. When Show Items is on, Resource Workshop displays items within individual resources (for example, POPUPs and MENUITEMs defined in a menu resource).

Show Unused Types

Lists all the resource types, whether or not they are used in the project.

Show Unused Types is available only when you are viewing By Type.

Selecting a resource

To select a resource, use the mouse or the arrow keys to highlight it in the Project window.

- If you've chosen View | By Type, look for the resource type first. The resource is listed by name under the resource type.
- If you've chosen View | By File, look for the resource under its file name, if you know it. The resource name is preceded by the resource type and a colon.

  For example, in the myproj.rc Project window in Figure 14.1 on page 203, the Icon resource my_ico1 is listed under the file iconfile.ico as ICON:my_ico1.

Working with resources

This section describes typical tasks performed on resources.

Loading a resource

To load a resource for naming or editing, you can do either of the following:

- Double-click the resource name (not the file type) in the Project window. Resource Workshop automatically starts the appropriate graphical resource editor, if one is available. If a resource editor is not available (for example, for a user-defined resource), Resource Workshop starts the internal text editor.

- Select the resource name in the Project window and then choose either Resource | Edit (to open a graphical editor) or Resource | Edit as Text (to open the text editor).

Resource editors

Each resource editor is explained in the chapter on that resource.

When you double-click one of the following resource types in the Project window (or select it and choose Resource | Edit), Resource Workshop loads the appropriate resource editor:

- If the resource is a dialog box, menu, accelerator, or string table, Resource Workshop loads an editor specifically designed to work with that type of resource. For descriptions of these editors, see Chapters 15 through 18.
- If the resource is a bitmapped resource (icon, bitmap, cursor, or font), Resource Workshop displays the resource in the Bitmap editor.

The internal text editor

User-defined resources and two predefined resource types—rcdata and VERSIONINFO—can be edited only with the internal text editor.

If the resource is in binary format (like a dialog box or menu), Resource Workshop decompiles it to let you work with the resource script.

See the online Help index for a description of the resource script language.

The internal text editor is similar to the Windows Notepad editor. It uses the *Del, Home, End, PgUp, PgDn, Tab,* and *Backspace* keys as you would expect and is always in insert mode.

When you enter text, don't spend any time formatting it, because Resource Workshop is likely to rearrange the text for you when it compiles the resource.

If Resource Workshop finds any errors, it tells you what they are and puts you back in the text editor so you can correct them.

If you want to edit the resource script directly without the assistance of Resource Workshop, you can open the source file with an editor of your choice and edit·that file.

➡ When Resource Workshop loads the resource, it recompiles the file and reformats the resource script. If it encounters any comments in the script when you open the resource for editing, it displays a dialog box warning you that the comments will be deleted.

Adding a resource

You can add a resource directly to the project (an embedded resource) or as a file reference (a linked resource).

Adding an embedded resource

To add a resource to your project, open the project you want to work with and then

1. Choose Resource | New to display the New Resource dialog box.
2. Choose the header files you want the resource and its identifier to appear in. Note that you can only choose from header files attached to your project (use File | Add to project to add a new header file to your project).
3. Double-click the type of resource you want to create.

- If you're creating an accelerator, menu, dialog box, or string table, Resource Workshop puts an entry for the resource in the Project window and opens the appropriate resource editor.
- If you're creating an rcdata or VERSIONINFO resource, Resource Workshop puts an entry for the resource in the Project window and opens the internal text editor.
- If you're creating a bitmapped resource (an icon, cursor, bitmap, or font), Resource Workshop asks if you want to save the resource as source text or in binary format. Choose Source text to embed the object.
- If the type of resource you want to create isn't listed, you can press the New Type button and create a user-defined type for your resource. See Chapter 23 for information on user-defined resources.

Adding a linked resource

You can add a resource by reference (linked). To add an existing resource by reference,

1. Choose File | Add to Project. Resource Workshop displays the Add File to Project dialog box.
2. Either type the name of the file containing the resource in the File Name text box, or double-click the file name if it's listed in the Files box.

   If the file isn't in the current directory or is of a different type from the current type, you can select it in either of these ways:

   You can also enter file search criteria by selecting the file type from the File Type list box.

   - You can type the file's full path and name into the File Name text box.
   - You can change directories by using the Directories list box. Then enter your search criteria in the File Name box and press *Enter* (for example, use *.CUR to find all cursor files), or select a type from the File Type list box. When the file name you want appears in the Files list box, double-click to select the file.

3. In the second drop-down list (under the File Type drop-down list), you see the current project file listed, which is most likely where you will put the reference to the new file. If your project contains more than one .RC file and you want to put the reference elsewhere, scroll down the list to find the name of the file in which you want to place the reference.
4. Press *Enter* or click OK to add the file to the project. Resource Workshop puts an entry that points to this file in the Project window.

   If you choose View | By File, you'll see the file name listed and under it the resource name. Any changes you make in the project to this resource are reflected in the original resource file.

To add a new resource by reference,

1. Enter the name of the file you want to create in the File Name box of the Add File to Project dialog box. If you're not using one of the standard resource file extensions (like .CUR), use the File Type box to identify the type of resource you're creating.
2. Click OK. A dialog box appears saying the file doesn't exist, but asks if you want to create it.
3. Click OK to create the file. Resource Workshop inserts a reference to the file in the Project Window.

Moving a resource

You can move resources from one file in a project to another file in that project by using Resource | Move. Both files must be part of the same project.

1. Select a resource in the project file by either editing it or highlighting it in the project resources list.
2. Choose Resource | Move. A dialog box appears displaying the resource name and the file that it is currently in (labeled Old).
3. Select a New file (the file you want the resource to appear in), then click OK.

Copying resources between projects

There are two ways you can copy a resource from the current project to another project:

- One way is to save the resource as a file, close the current project, open the other project, and add the resource as a file to the new project. If there are any identifiers in the resource, you have to be careful that they're preserved when you add the resource to the new project.
- An easier way is to have two copies of Resource Workshop open, one for each project, and to use the Windows Clipboard to copy the resource

from one project and paste it to the next. This method is not only faster than the first method, but it also saves all the identifiers.

To copy a resource using the second method,

1. Open two copies of Resource Workshop, one with the project containing the resource you want to copy (the source project) and another with the project you want to copy the resource to (the target project).
2. Be sure the target project has a reference to an identifier file that will receive any identifiers in the new resource. (If necessary, choose File | Add to Project and add the appropriate type of identifier file.)

Don't use the *Ctrl+Ins* hot key—it won't copy the resource properly.

3. Select the source project, then select the resource you want to copy in the Project window. Choose Edit | Copy to copy it to the Windows Clipboard.
4. Select the target project and then, with the Project window active, choose Edit | Paste to paste the resource into the project. Resource Workshop displays the Paste Resource dialog box.
5. The Paste Resource Into list box should contain the name of the target project. Make sure the Paste Identifiers Into list box contains the name of the identifier file that will receive any identifiers in the resource (if necessary, scroll the drop-down list and choose the correct file name), then press *Enter* or click OK to paste in the new resource.

Deleting a resource

To delete a resource from a project, select the resource in the Project window, then choose either Edit | Cut or Edit | Delete to remove it. (Edit | Cut lets you paste the resource elsewhere.)

Renaming a resource

To rename a resource,

1. Choose Resource | Rename. Resource Workshop displays the Rename Resource dialog box.
2. In the New Name text entry box, type the new resource name, then press *Enter.*

See page 213 for a discussion of creating identifiers when you rename resources.

3. Resource Workshop asks if you want to create a new identifier by that name.
   - If you click Yes, Resource Workshop renames the resource and assigns it the identifier value you specify.
   - If you click No, Resource Workshop renames the resource without creating an identifier. See page 210 for a discussion of the advantages of creating identifiers.

Specifying resource memory options

Resource Workshop lets you specify how each resource's memory is managed. If you are a new Windows programmer, you might want to use the defaults.

To specify memory options, select the resource in the Project window and then choose Resource | Memory Options. The Resource Memory Options dialog box appears.

Uncheck any memory options you don't want. (See the description of the four options in Table 14.1.)

➡ If you set memory options for an Icon resource, those options apply to all the images in that resource.

Here's what each option in the Resource Memory Options dialog box does:

Table 14.1 Resource memory options

Option	Description
Load on Call	Loads the resource into memory only when it's needed. Choosing Load On Call can reduce the amount of time required to load your program. If you uncheck this option, you'll activate Preload, which means that Windows will load the resource into memory when it first loads the program. You need to preload a resource only if you know Windows needs the resource as soon as the application begins to execute.
Moveable	Lets Windows move the resource segment in memory to make room for other memory allocations. If you uncheck this option, the resource segment occupies a fixed block of memory.
Discardable	Lets Windows discard the resource segment from memory when it's no longer needed. Windows can load the resource into memory again when necessary. If you uncheck this option, you'll activate Nondiscardable. Windows won't be able to remove the cursor segment from memory while the application is running, and, if Pure isn't checked, you'll be able to modify the resource from within your application. Note that Nondiscardable is not compatible with the RC.EXE.
Pure	Prevents the resource segment in memory from being modified. Usually, you'll want to leave this option checked. See the Windows documentation for information about this option.

Using identifiers

Windows requires that every resource be associated with a unique name or a unique integer (called a *resource ID*). By default, Resource Workshop assigns a name to each new resource—for example, DIALOG_1 for a dialog box or MENU_1 for a menu resource.

The default name isn't very descriptive, and referring to a resource by name alone decreases the application's efficiency at run time. To overcome these shortcomings, you can rename the resource and assign it an *identifier* (a C **#define** or a Pascal constant). Using identifiers gives you both descriptive names and run-time efficiency.

- You can rename the resource to an integer value. Using an integer value has the same effect on the application's run-time efficiency as assigning an identifier, with the drawback that the resource ID still isn't very descriptive. DIALOG: 225 doesn't tell you which dialog box it is.
- You can rename the resource, but elect not to create an identifier. The result is a resource with a descriptive name and code that is easier to work with than an identifier (the Windows parameters expect a far pointer to a **char**), but you'll lose some of the application's run-time efficiency.

Components of resource identifiers

A resource identifier consists of two parts: a text literal (the identifier name) and a value (typically an integer). For example, the following statement declares an identifier with a name of dlg_OpenFile and a value of 100:

```
#define dlg_OpenFile 100
```

Resource Workshop lists the resource in the Project window as DIALOG: dlg_OpenFile, which readily identifies it as the Open File dialog box.

Identifiers must be unique within a resource type. Only the first 31 characters are significant; Resource Workshop ignores any characters past the 31st character. You can change identifier values in text boxes. For example, in the Menu editor you can type an Item Id as `CM_FILENEW=125`. When you move off the field, Resource Workshop changes the value of the CM_FILENEW identifier to 125.

Assigning an integer value to an identifier speeds up calls to the resource at run time, but you won't be able to use the short integer value directly as a parameter. You must either typecast the integer into a long pointer to **char** or use a macro to do the typecasting for you.

If you write your program in C or C++, you can use the MAKEINTRESOURCE macro. If you write your program in Pascal, you can use the MakeIntResource type (a pointer to *char*).

The MAKEINTRESOURCE macro looks like a function call but actually does a typecast on the identifier. For example, to use dlg_OpenFile as a parameter in a C or C++ program, enter it as the following expression:

```
MAKEINTRESOURCE(dlg_OpenFile)
```

If you're working with a .RES file, an executable file, or a DLL, Resource Workshop decompiles all resource IDs in the file into integer values. You can't add identifiers to this type of file, but you can save the file as an .RC file and then assign identifiers to its resources. See the section "Working with binary files" on page 217.

Identifier files

When you open a new project, the first thing you should do is specify a file in which to store your identifiers.

Store your identifiers in one or more header (.H) files that use #**define**s to assign values to the identifier names.

This manual refers to header files, units, and include files as *identifier files*.

You can use a text editor or word processor to create your identifier files, but you can also create them with Resource Workshop, as described in the next section and in the section "Adding identifiers" on page 213.

Creating identifier files

After you open a new project (File | New Project) and give it a name (File | Save Project), add the identifier file by taking these steps:

1. Choose File | Add to Project. Resource Workshop displays the Add File to Project dialog box.
2. Click the File Type list box down-arrow button to display a list of file types you can add to your project.
3. If you are writing your application in C, choose

   ```
   H c header
   ```

4. In the File Name text box, type a name for the identifier file.
5. Click OK to exit the Add File to Project dialog box. Resource Workshop creates the identifier file at this time.

C header files

If your program is written in C, store your identifiers in a .H header file. The #**define**s in the file assign integer values to the identifier names.

The following is a sample from a typical header file:

```
/****************************************************
 *     Selected #defines from RWCDEMOC.H            *
 ****************************************************/

#define  bmp_StatusBar              101
#define  cm_About_CUA               145
#define  id_ClearWindow             229
#define  dlg_About                  104
#define  dlg_FileNew                106
#define  sth_Edit                    15
#define  men_Main                   100
#define  acc_Main                   100
#define  ico_RWCDemo                100
#define  sth_EditClear               13
#define  ScribbleWindow             100
#define  FileWindow                 101
#define  GraphWindow                102
```

In addition to **#define**s, you can also store type and structure definitions, program code, and comments in a header file. Resource Workshop ignores all data in the header file except for the **#define**s and any preprocessor directives.

Automatic identifier management

Resource Workshop can automatically create and delete identifiers for you. To turn on automatic identifier management, choose File | Preferences and check Generate identifiers automatically. Note that if you're using AppExpert from the Borland C++ IDE, Generate identifiers automatically will be checked and unselectable; you won't be able to turn this option off because AppExpert relies on the automatic identifiers syntax.

With automatic identifiers on, every time you create a resource item that uses an identifier (menu items, for example), Resource Workshop creates a unique identifier for that item and places it in the header file for that resource (.RH or .RC). Also, if you delete any items, the identifier is deleted.

Resource Workshop uses an identifier prefix, which you can change. Here's a list of the default prefixes and the menu commands to change them (note that you must be editing the resource type to get the correct menu to display):

Table 14.2
Identifier prefixes

Resource	Prefix	Menu command
String table	IDS_	String Table|Change identifier prefix
Menu	CM_	Menu|Change identifier prefix

Table 14.2: Identifier prefixes (continued)

Accelerator	CM_	Accelerator|Change identifier prefix
Dialog	IDC_	Options|Change identifier prefix

Working without an identifier file

If you don't add an identifier file to your project, you can still create identifiers for your resources. Resource Workshop stores these identifiers in the active project file—as #**define**s.

This practice is *not* recommended for the following reasons:

- If you decide later to use a separate identifier file, you'll have to use a text editor to cut the #**define**s from the resource scripts and paste them into the identifier file.
- Because the resource script file supports only #**define**s, you'll have to take an extra editing step to create a Pascal unit or constant file.

Adding identifiers

There are two ways to add identifiers to your project:

- By renaming your resource
- With the Identifiers dialog box

By renaming resources

As described on page 208, you can rename your resources as you create them. When Resource Workshop encounters a new resource name, it automatically asks if you want to create an identifier for that name.

For example, let's say you've just added a new menu resource to your project. By default, Resource Workshop gives it the name MENU_1. (It happens to be the first menu in the project.) Because it's a file menu, you decide to change its name to Menu_File, so you choose Resource | Rename to display the Rename Resource dialog box.

When you enter a new name in the Rename Resource dialog box and click OK, Resource Workshop asks if you want to create a new identifier for that name. If you click OK, Resource Workshop displays the New Identifier dialog box.

First, make sure the right identifier file name appears in the File list box. Then type an appropriate integer value into the Value text box, and click OK. The next time you save your project, the identifier name and value will be saved to your .H, .PAS, or .INC file.

By using the Identifiers dialog box

Using the Identifiers dialog box, you can add an identifier to your project before you create the resource it will be associated with. To add an identifier,

1. Choose Resource | Identifiers to display the Identifiers dialog box.
2. Click the New button. Resource Workshop displays the New Identifier dialog box.
3. Use the File list box to specify the file in which the identifier is to be stored.
4. Type the resource name in the Name text box.
5. Type the ID value in the Value text box.
6. Click the OK button.

   Note that the new resource name now appears in the Identifiers list box, and that its Value is given as (unused).

In addition to adding identifiers, you can use the Identifiers dialog box to edit, delete, or list identifiers and to start a resource editor. To display the Identifiers dialog box, choose Resource | Identifiers.

Editing identifiers

You can change an identifier value by following these steps:

1. Choose Resource | Identifiers to display the Identifiers dialog box.
2. Select the identifier whose value you want to change.
3. Click the Change button. Resource Workshop displays the Change Identifier Value dialog box.
4. Type a new value in the New Value text box and click OK.

The new identifier value will be written to your .H, .PAS, or .INC file the next time you choose File | Save Project.

You can also move and rename resources using the Identifiers dialog box.

Deleting identifiers

If an identifier is not used in your project, you should delete it from the .H, .PAS, or .INC file. Here are three reasons you might have an unused identifier:

- You assigned an identifier to a resource and then deleted the resource.
- You added an identifier to the project and then never used it.
- You renamed a resource that already had an integer identifier value.

To delete an identifier,

1. Choose Resource | Identifiers to display the Identifiers dialog box.
2. Select the identifier you want to delete.

   If the selected identifier is not associated with a resource (either because the resource was deleted or the identifier was never used), the Usage box says (unused).

   You *can* delete an identifier that is still in use.

   If, however, the identifier is still associated with a resource, the Usage box automatically highlights the type and name of the associated resource.
3. Click the Delete button.

   If the identifier is unused, it is deleted immediately. No warning dialog box is displayed.

   If the identifier is still in use, Resource Workshop displays a warning dialog box that says "#define [or Constant] is used. Delete anyway?" To delete the identifier, click the Yes button. If you don't want to delete the identifier, click the No button.
4. The next time you choose File | Save Project, Resource Workshop updates the identifier file, removing the deleted identifier.

Listing identifiers

To list the identifiers in your project,

1. Choose Resource | Identifiers to display the Identifiers dialog box.
2. Choose either All or Single File in the View group.

   If you choose Single File, select the file whose identifiers you want to see from the file name list box in the View group.
3. Scroll the Identifiers list box to the identifiers you want to see. When you highlight an identifier in the list box, its name and integer value appear in the Name and Value boxes above the list box.

Starting a resource editor

You can use the Identifiers dialog box to start a resource editor with a preselected resource already loaded.

To start a resource editor from the Identifiers dialog box,

1. Scroll the Identifiers list box until the resource you want is highlighted.

   The resource's type and name appear in the Usage list box.
2. Double-click on the highlighted type and name in the Usage list box.

   Resource Workshop starts the appropriate editor with that resource already loaded.

Setting preferences

To set preferences, , choose File | Preferences. See the online Help for information on items not discussed in this section (click the Help button in the Preferences dialog box).

Undo Levels

Resource Workshop has a multilevel Undo and Redo feature that lets you correct actions in any of the resource editors. Depending on the amount of available memory in your computer, you can undo or redo up to 99 actions. The default number of levels is ten.

For each undo, press *Alt+Backspace* or choose Edit | Undo; for each redo, press *Shift+Alt+Backspace* or choose Edit | Redo. You can work your way back and forth through your edit session this way through as many levels as you have set in the Undo Levels option.

➡ The number of undo levels can be limited by available system memory. A memory-intensive resource—like a large bitmap with several flood fills—can cause fewer undo levels to be available.

Text Editor

When Resource Workshop loads a project file, it compiles all the resources in the file. If the compiler encounters an error, it stops compiling, notifies you there was an error, and asks if you want to edit the file by using the external text editor you have specified in this option.

The default external text editor is the Windows Notepad editor. If you specify another editor, it must be one that runs under Windows. (A DOS editor with a .PIF file will work.)

Multi-Save

The .RES and Executable preferences control how a project is to be saved when you select File | Save Project. These preferences are enabled only when a resource compiler (.RC or .DLG) project is open because they apply only to a specific project. Regardless of the Multi-Save settings, the project always gets saved in its original format as well. (For example, if the project is an .RC file, the resources in the file are always saved as resource scripts in addition to any Multi-Save options.)

- *.RES.* Compiles the current project's resources and saves them in .RES format (in binary format).
- *Executable.* Compiles the current project's resources and binds them to the executable file specified in this option (can be an .EXE or .DLL file).
- *Make Backups When Saving Files.* If you check the Backups option, Resource Workshop creates an additional set of backup files each time you save a project. Backup files have a tilde (~) as the first character in

the file extension. For example, when you save MYPROJ.BMP, the backup file is called MYPROJ.~BM.

Target Windows Version

These radio buttons let you target your resource for specific versions of Windows. Note, however, that .RES and .EXE files targeted for Windows 3.1 are not backward-compatible with Windows 3.0.

If the .RES file is targeted to Windows 3.1, you cannot use the Windows 3.0 version of RC.EXE to bind your resources to the application. You must use Borland tools or version 3.1 of the Microsoft Resource Compiler.

➡ This option is available only if no project is currently open.

Working with binary files

Resource Workshop lets you open executable files, .RES files, and DLLs as projects so you can customize their user interfaces. For example, you might want to translate your application interface into another language.

When you load one of these files, Resource Workshop decompiles the resources in the file and shows them to you as though they were part of a regular .RC file. When you're finished with your changes, Resource Workshop compiles the resources again into binary code and stores them in the original file.

All resource IDs in binary files must be integers.

Because the resources in a decompiled binary file aren't stored in resource script form, you can't assign identifiers to the resource IDs.

You can, however, save the project as an .RC file. The resources can then be saved as resource scripts, and you can assign identifiers to them.

If you're customizing the user interface of a program and have access only to the executable file or DLL, you might also want to save your changes in a separate .RC file so you can apply the changes to the next version of the program. The resources in your .RC file must have the same resource IDs as their counterparts in the new version and must otherwise be compatible with the new version.

When you save the project as an .RC file, Resource Workshop doesn't automatically save the resources back to the original file unless you've entered the original file name as a Multi-Save option in the Preferences dialog.

To save a binary file as a project and add identifiers, do the following:

1. Choose File | Open Project and select the executable, .RES, or DLL file from the Open Project dialog box.
2. Choose File | Save File As. In the Save File As dialog box, select RC Resource Script from the File Type list box. Enter the name of the new .RC file.

   When you press click OK to save the file, Resource Workshop automatically places you in the .RC file.
3. Choose File | Preferences and enter the name of the original binary file as a Multi-Save option.
   - If the original binary file was a .RES file, check .RES and enter the name in that text box.
   - If the original binary file was an executable or DLL file, check Executable and enter the name in that text box.
4. Choose File | Add to Project and specify an identifier file to hold the new identifiers. If the file you specify doesn't exist, Resource Workshop creates a new one for you.

Be sure to preserve the current integer values of the resource IDs.

5. Make your changes to the resources and specify identifiers where you want them. For each new identifier, Resource Workshop asks if you want to save it in the identifier file.
6. When you quit and save the file, Resource Workshop saves both the .RC file and the binary file. If the binary file is an executable file or a DLL, the changed resources are bound into it and are available immediately when you run that program.

Creating 32-bit resources

By default, Resource Workshop creates 16-bit resources if you're running under 16-bit Windows or 32-bit resources if you're running under NT. You can change the type of resources you create. For example, if you're running under NT, you can also create 16-bit resources, and if you're running under Windows, you can also create 32-bit resources.

To change the type of resource project you create, close any open projects, then

1. Choose File | Preferences.
2. Check the Target Windows version option you want, then click OK.

3. Choose File | New project. Once you create a project, you can't change the Target type (for example, you can't change from Win32 to Windows 3.1).

You can also choose a major and minor language from the File | Preferences dialog box if you're targeting Win32. Note that the language you choose here must be the language used in NT, otherwise you might not view your resources correctly. For example, if you choose French and German as your resource languages, but you have NT set to English, you can build your resources in Resource Workshop, but when you run your program, you might not see your resources correctly. This is especially true for menu resources.

C H A P T E R 15

Creating dialog boxes

Dialog boxes give the user a way to interact with your application. A dialog box is a pop-up window that lets the user specify information (files to open, colors to display, text to search for, and so on).

The dialog box usually contains a number of *controls*, such as buttons, text boxes, and scroll bars. Controls usually let the user specify information, but can also be used to display static text and graphics in a dialog box.

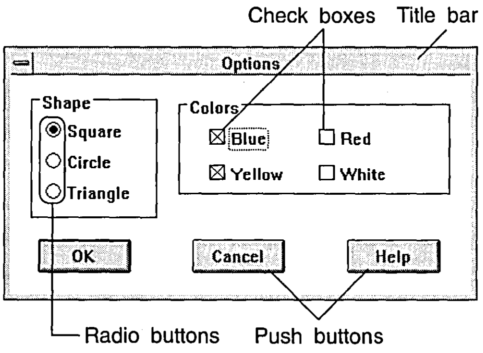


Figure 15.1
A typical dialog box

Starting the Dialog editor

Using the Dialog editor, you can create new dialog boxes or edit existing ones. How you start the Dialog editor depends on which task you want to perform.

Creating a new dialog box

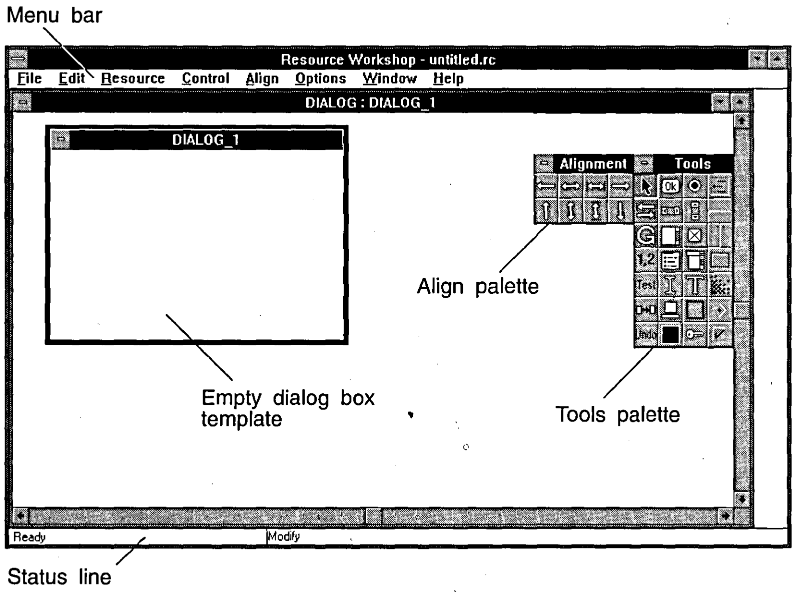
Chapter 14 describes how you open a project.

To create a new dialog box,

1. Choose File | New Project to start a new project or File | Open Project to load an existing project.
2. Choose Resource | New. The New Resource dialog box appears.
3. In the Resource Type list box, double-click DIALOG.

4. Another dialog box appears where you specify a template for your dialog box. Choose a dialog type, then click OK.
5. The Dialog editor opens displaying a template dialog of the type you chose in step 4. Figure 15.2 shows the components of the Dialog editor.

Figure 15.2 Dialog editor with empty dialog box


Menu bar
Resource Workshop - untitled.rc
File Edit Resource Control Align Options Window Help
DIALOG : DIALOG_1
DIALOG_1
Alignment
Tools
Align palette
Empty dialog box template
Tools palette
Ready
Modify
Status line

Editing an existing dialog box

To edit a dialog box that already exists in a project file,

1. Open the project that contains the dialog box you want to edit.
2. Locate the dialog box you want to edit in the Project window and double-click its name.

The Dialog editor appears as in Figure 15.2, except that the dialog box you selected appears in place of the empty dialog box template.

Using the Dialog editor

When you create a new dialog box, the Dialog editor presents you with an empty template. You can decide what type of window your dialog box will be, specify a caption, choose which fonts will be displayed in it, and so on. You can also add, change, group, reorder, move, resize, or delete dialog controls so that your dialog box functions the way you want it to.

Selecting a dialog box

To make changes to the dialog box window or to set the attributes of the dialog box, you must first *select* it. To select the dialog box, click its title bar or outer edge.

When a dialog box is selected, Resource Workshop displays a *selection frame* around it, and you can

- Move the dialog box by using the mouse or the arrow keys.
- Resize the dialog box in either of the following ways:
  - Drag the appropriate edge or corner.
  - Select the dialog box by clicking its title bar, and then choose Align | Size to display the Size Dialog dialog box. Enter width (CX) and height (CY) values in dialog units. (In a dialog unit, *y* equals $\frac{1}{8}$ of the font height, and *x* equals $\frac{1}{4}$ of the font width.) The size values apply to the outer dimensions of the dialog box.

Setting dialog box attributes

Using the Window Style dialog box, you can set the attributes of your new dialog box, including

- Caption, Class, and Menu
- Window type
- Frame style
- Dialog style
- Font

To display the Window Style dialog box, select the dialog box you want to set attributes for, then press *Enter*. You can also double-click on the title bar of the dialog box you're editing using the Selector tool (the mouse pointer will appear as an arrow).

Adding a caption

You can also change captions using the Properties box (choose Options| Show Properties).

To add a caption,

1. Double-click the dialog box title bar to open the Window Style dialog box.
2. Type the new name for your dialog box in the Caption box.
3. Under Frame Style, make sure Caption (the default) is selected.
4. Click *OK*.

The Window Style dialog box lets you choose a type for your dialog box:

- *Popup*. A pop-up window. Because most dialog boxes are popups, this is the default.
- *Child*. A child of the current window.

- *Overlapped.* An overlapped pop-up window that can be covered or partially covered by another. You'll want to define a dialog box as overlapped only when it's the main window in the application.

Choosing a frame style

The frame style of the dialog box determines the appearance of the dialog box frame and the title bar. Frame styles are defined as follows:

- *Dialog Frame.* A double border, without a title bar.
- *Border.* A single, thin border, without a title bar.
- *Caption.* A single, thin border and a title bar where a caption can be displayed (default).
- *No Border.* No border and no title bar.

Choosing a dialog style

Dialog styles determine what the dialog box looks like and how the user works with it. You can choose one or more of the following styles for your dialog box:

- *System Menu.* Includes a System menu box on the left side of the title bar (appears only if you've also chosen Caption).

  If the dialog box is defined as a child window, you'll get a Close box instead of a Control menu.
- *Thick Frame.* Places a thick frame around the dialog box. This option defines what the user will see when the dialog box appears within an application. If you want the dialog box to be resizable, use this option.

  (Don't confuse this option with the Thick Frame option in the Dialog editor Preferences command. That option defines what the dialog box looks like when you select it in the Dialog editor.)
- *Vertical Scroll.* Adds a vertical scroll bar to the dialog box frame.
- *Horizontal Scroll.* Adds a horizontal scroll bar to the dialog box frame.
- *Minimize Box.* Adds a Minimize button on the right side of the title bar. The Minimize button appears only if you've also chosen Caption for the dialog box frame style.
- *Maximize Box.* Adds a Maximize button on the right side of the title bar. The Maximize button appears only if you've also chosen Caption for the dialog box frame style.
- *Absolute Align.* Makes the dialog box coordinates relative to the display screen rather than the parent window.
- *System Modal.* Makes the dialog box system modal, meaning that the user can't switch to anything else until the dialog box is put away.
- *Local Edit.* Allocates any edit text controls included in this dialog box to the application's local heap.

Choose Local Edit if your application needs to use EM_SETHANDLE and EM_GETHANDLE messages.

- *Modal Frame.* Frames the window with a combination of the dialog frame and caption styles (default). Choose Modal Frame if you want users to be able to move the dialog box.
- *No Idle Messages.* Suppresses sending WM_ENTERIDLE messages to the application's main window. The dialog box must be modal for this option to take effect.
- *Clip Children.* Protects the client area of child windows from being drawn on by the dialog box window.
- *Clip Siblings.* Protects the siblings of this window. Drawing is restricted to this window. This option is not required for pop-up windows, but can be useful for child dialog windows.
- *Visible.* Makes a modeless dialog box visible before the return from CreateDialog. This option has no effect on modal dialog boxes (the usual kind of dialog box). By default, this option is not checked (NOT WS_VISIBLE).

Changing fonts

By default, dialog boxes use the 8-point MS Sans Serif font for text, but you can change the font and the point size. To choose a font for your dialog box,

1. Open the Window Style dialog box.
2. Click the Fonts button to open the Select font dialog box.
3. Use the Select font dialog box to choose a typeface and size for the text in your dialog box. The characters in the Text box at the bottom of the dialog box show the currently selected typeface and size. (Note that Windows accepts only bold text in dialog boxes.)

Including a menu

Because it's really a window, a dialog box can include a menu. For example, some applications use a dialog box for their main window, in which case the dialog box would need a menu.

To include a menu in your dialog box,

1. Create the menu as a separate resource and make sure it's part of the project (See Chapter 16). Note the resource name or numeric ID that identifies the menu.
2. Open the Dialog editor and load the dialog box to get the menu.
3. Open the Window Style dialog box.
4. In the Menu input box, type the menu's resource name or numeric ID.

In the standard drawing modes the Dialog editor doesn't display the menu. When you test the dialog box (see page 253), Resource Workshop displays a generic menu called "Menu" that includes a single menu item called "Item."

Assigning a custom class to a dialog box

If you're an experienced Windows programmer, you might want to assign a custom class to a dialog box so you can process dialog box messages with your own window procedures (instead of Windows procedures). Another reason for assigning a custom class might be to make the dialog box a Borland-style dialog box.

To assign a custom class to a dialog box,

1. Open the Window Style dialog box.
2. Type the class name in the Class input box.

Setting dialog box position

If your dialog box uses the WS_OVERLAPPED style (you checked Overlapped as the window type), you can let Windows position it on the screen. This option is generally used for dialog box frames that function as main windows. To give control of the dialog box's position to Windows,

1. Select the dialog box frame by clicking on its edge or the title bar.
2. Choose Align | Size.
3. In the Size Dialog dialog box, click the Set by Windows radio button.

   Resource Workshop grays out the X-coordinate value in the dialog box, indicating that Windows has control of the dialog box.
4. Click OK to exit the Size Controls dialog box.

Working with controls

Controls are the individual components of dialog boxes. They let users provide information to your application or receive information from it. Controls fall into these general categories:

- Buttons
- Scroll bars
- List boxes
- Edit text
- Static controls
- Combo boxes
- Custom controls

Families of controls

Each control you use comes from one of five family groups:

- Standard Windows controls, like push buttons, check boxes, list boxes, and radio buttons. Icons for the standard Windows controls appear in the second and third columns of the Tools palette. Standard Windows controls are always available.

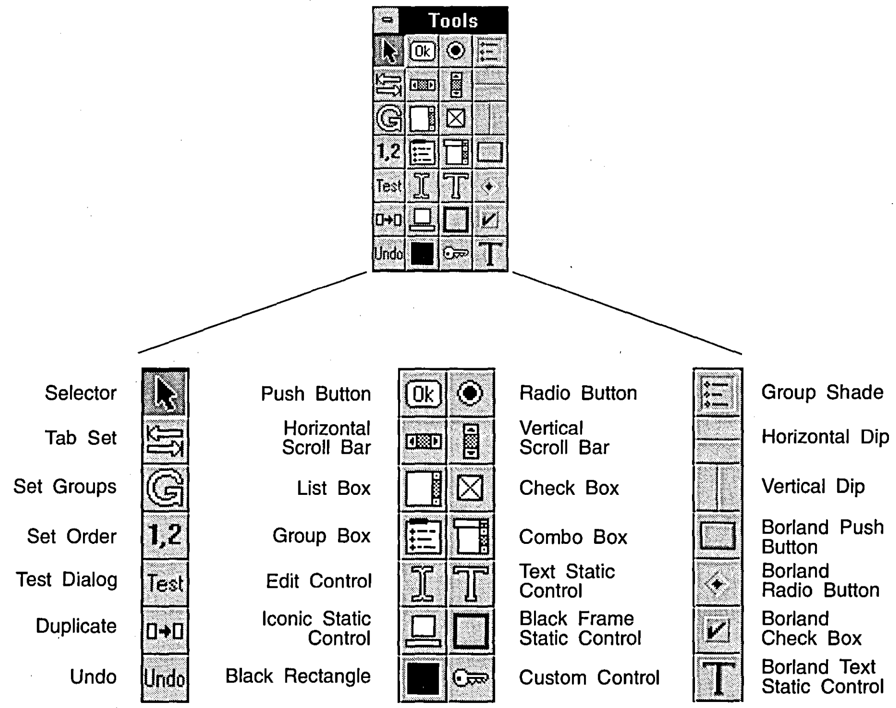
- Borland Windows Custom Controls (BWCC), which are also always available. These controls (including radio buttons, check boxes, and push buttons) offer both visual and functional enhancements over the standard Windows controls. Icons for BWCC appear in the fourth column of the Tools palette. The BWCC controls are described in Appendix B.
- Custom controls whose class is recognized by Resource Workshop. These controls are stored in a dynamic-link library (DLL) that includes the *ListClasses* function. When the DLL file is installed, the icons for these controls appear in one or more additional columns in the Tools palette, starting to the right of the BWCC icons. This type of control is described on page 251.
- Custom controls whose class is recognized by the Windows SDK dialog editor. These controls are stored in a DLL file that includes their bitmaps but does not include the *ListClasses* function. They are not represented in the Tools palette, but their names appear in the drop-down list of the New Custom Control dialog box (see page 252). When you add one of these controls to a dialog box, its bitmap appears on the screen when Resource Workshop is in WYSIWYG display mode.
- Custom controls whose class is not recognized by Resource Workshop or the Windows SDK dialog editor. Resource Workshop adds their names to the drop-down list in the New Custom Control dialog box, but they appear on the screen in WYSIWYG mode as gray rectangles.

Tools palette

The left column of the Tools palette contains tools that set the Dialog editor's current operating mode.

The remaining columns contain icons for the standard Windows controls, the BWCC controls, and any custom controls you've loaded and that Resource Workshop recognizes.

Figure 15.3
Tools palette



Here are brief descriptions of the standard Windows controls whose icons appear in the second and third columns of the Tools palette:

Push button	A rectangular button the user "presses" to select an action. Because push buttons always contain text, you must specify a caption for each one.
Radio button	A circular button with text to its left or right. When the button is selected, a solid dot fills the circle. Radio buttons are used in groups to represent related but mutually exclusive options.
Horizontal scroll bar	A horizontal rectangle with direction arrows on each end.
Vertical scroll bar	A vertical rectangle with direction arrows on each end.
List box	A rectangle usually containing a list of text strings. If you use the Owner-draw style, the list box can also contain a visual representation of a list of data. Usually, a user can browse through what's displayed

	in a list box, then select one or more items. You'll often see list boxes used in a File Open dialog box.
Check box	A rectangular button with text to its left or right. When a check box is selected, an X appears in the square. When a check box is not selected, the square is empty. Check boxes are often used to represent Boolean (on/off) states for individual options.
Group box	A rectangular box used to visually group other controls together. You can include a caption to display in the upper left corner of the group box.
Combo box	A combination of a list box and edit text control or a list box and static control.
Edit text control	A rectangle into which the user can enter text from the keyboard.
Text static control	Text that appears in the dialog box.
Iconic static control	An icon.
Black frame static control	An empty, rectangular frame that takes the color of the current window frame.
Black rectangle	A static control icon appearing as a rectangle that's the same color as the current window frame.
Custom control	A control whose class is different from the standard Windows or BWCC types and is not recognized by Resource Workshop.

Selecting controls

A number of Resource Workshop's editing options require that one or more controls be *selected*. For example, you must select a control before you can change its size, and you must select at least two controls before you can align them relative to each other. When a control is selected, it is surrounded by a selection frame.

To switch to *selection mode* so you can select controls, you must first click the Selector tool. The mouse pointer cursor becomes an arrow.

To select a single control, click the Selector inside the control or on its edge, depending on the current state of the Select Near Border check box in the Options | Preferences dialog box.

To select more than one control, you have two options:

- You can drag a *selection rectangle* around the controls you want to select, as follows:

  1. Choose the Selector tool.
  2. Click on one of the controls to ensure that the control (and not the entire dialog box) is selected.
  3. Drag a selection rectangle. Depending on the current state of the Selection Rectangle Surrounds check box in the Options | Preferences dialog box, the selection rectangle must either entirely surround the controls or just touch them.

  Resource Workshop places a selection frame around the selected controls. You can drag the selection frame to move the selected controls.
- You can Shift-click to select controls that would not be selected by dragging. (For example, a selection rectangle might include controls you don't want to select.) To Shift-click, click the first control, and then hold down the *Shift* key as you click the additional controls.

To add one or more controls to a group you've selected, or to delete one or more controls from the group, Shift-click the controls you want to add or delete.

To select all the controls in a dialog box, choose Edit | Select All. Resource Workshop places frames around each control in the dialog box and a selection frame around the group of controls. Edit | Select All does not select the dialog box window frame.

You can also select controls with the *Tab* key.

To see the sequence, click the Set Order tool.

- If a single control is selected, pressing *Tab* moves the selection to the next control in the sequence in which the controls were added. Pressing *Shift-Tab* moves the selection to the previous control in the sequence.

  In all cases, if the selected control is the last in the sequence, *Tab* moves the selection to the dialog box frame.
- If you've selected a group of controls by dragging a selection rectangle, *Tab* moves the selection to the next control in the sequence after the highest one in the group. The group is deselected.

  For example, if the selection frame contains controls 4, 5, 9, and 10, *Tab* moves the selection to control 11 (if it exists).
- If you selected a group of controls by *Shift*-clicking, *Tab* moves the selection to the next control in the sequence after the last control you selected.

For example, if you group-selected controls 9, 10, 4, and 5 in that order, *Tab* would move the selection to control 6.

Adding controls

The easiest way to add a new control to your dialog box is to

1. Click the control you want in the Tools palette. Your cursor will change to indicate the type of control you are placing.
2. Click where you want to place the control in the dialog box.

If you select a control from the Tools palette and then change your mind about placing it, choose the Selector tool. Your cursor will return to the familiar arrow shape, and you will be able to select controls in your dialog box. Pressing *Esc* before you place a control also returns you to selection mode.

You can also use the Dialog editor's Control menu (in the menu bar, not the window border) to add controls to your dialog box.

1. Use your mouse or press *Alt+C* to open the Control menu.
2. Choose the control type you want to add to your dialog box.
3. Click in the dialog box where you want the control placed.

Adding multiple copies of a control

You can place multiple, identical copies of a control in rows or columns. For example, you might want to place two columns of four check boxes in your dialog box. You could place each check box individually, but Resource Workshop gives you an easier way.

To place multiple copies of a control in rows or columns,

1. In your dialog box, select the control you want to duplicate.
2. Click the Duplicate tool or choose Edit | Duplicate. The Duplicate Control dialog box appears.
3. Specify the number of rows and columns you want, as well as the spacing in dialog units between the rows and columns.

   For example, if you want eight check boxes placed in two columns, you would specify four rows and two columns.
4. Choose OK. Neatly aligned, multiple copies of your control appear in your dialog box.

When only a single control is selected, the Duplicate tool has the same effect as Edit | Duplicate. For multiple selected controls, it has the same effect as Align | Array (see page 239).

Control properties (.VBX controls)

You can view the properties for controls you add to dialog boxes by using the Options | Show Properties menu. When Show Properties is on, a Properties dialog box appears listing the properties for the selected control.

A .VBX control can be edited only with the Properties dialog box. You should understand what properties your control uses so you can easily change them.

If the Properties dialog box isn't displayed, you can double-click a .VBX control to view it.

Moving controls

You can move a selected control by dragging anywhere inside its selection frame. You can also move grouped controls in the same manner. As you drag, the controls move together, maintaining their position relative to each other.

To move a control with the keyboard, *Tab* to select the control, then use the arrow keys to move the control, and press *Enter*. Press *Esc* instead of *Enter* to undo the move.

Resizing controls

You can change the size of a selected control by dragging the appropriate edge or corner. You can use the keyboard and mouse together to "fine tune" the size:

1. Select the control and move the mouse cursor over the appropriate part of the border.
2. When the mouse cursor becomes a double-headed arrow, hold the left mouse button down.
3. Press the appropriate arrow key to move the mouse cursor and the selection frame. Each press of the arrow moves the cursor a single dialog unit.

Dialog units are defined on page 223.

Locating and sizing controls at the same time

Using the Size Controls dialog box, you can specify a control's position and size at the same time.

1. Select the control you want.
2. Choose Align | Size or hold down the *Alt* key while double-clicking the mouse. The Size Controls dialog box appears.
3. To set the position of the control's upper left corner, specify its X- and Y-coordinates in dialog units. The coordinates 0,0 place the control in the upper left corner of the dialog box window, directly below the title bar.

To set the control's width and height, specify its CX and CY values in dialog units. See Figure 15.4.

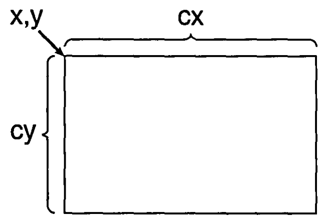


Figure 15.4 Dialog box coordinates

Aligning controls with a grid

You can display a grid on your dialog box and use it to align your controls.

To display a grid,

1. Choose Align | Grid. The Set Grid Attributes dialog box appears.
2. Specify the width and height of a grid cell (in dialog units).
3. Select the Grid Type. There are two options:
   - *Absolute.* Snaps the control to the nearest grid line.
   - *Relative.* Moves the control only in increments of the grid width horizontally and the grid height vertically. Therefore, if a control was not placed on a grid line originally, you will not be able to move it to a grid line with this option selected.

     For example, if you set the grid to be 4 × 4 and have a control with a position of (1,1), when you move the control, it will only go to positions that are 4 units away in either dimension. Possible coordinates would be (5,5), (5,9), (9,5), and so on.
4. Check the Show Grid option and choose OK.

Editing controls

To modify a control in your dialog box, double-click the control to display its Style dialog box. The options in this dialog box vary according to the type of control you're working with. For example, double-clicking a button control brings up the Button Style dialog box.

If you're using the keyboard, use *Tab* to select the control you want to edit. Press *Enter* to display the Style dialog box for the selected control.

Although each control type has its own Style dialog box, the dialog boxes have many options in common.

Table 15.1
Common options in Style dialog boxes

Option	Description
Caption	Lets you type the caption you want displayed with the control. Different type of controls display captions in different areas. For example, in a group box, the caption displays at the top left. In a push button, the caption displays inside the button. Not all controls display a caption. For example, a list box does not display the text specified in its caption. To the right of where you type the caption, check either Text or Number. Choose Text if you want the caption to be surrounded by quotation marks in the .RC or dialog file source code. Select Number if you don't want quotation marks.
Control ID	Lets you specify a unique identifier for the control. Control IDs can be a short integer or an integer expression. Type the control ID you want to assign to this control. By convention, static controls that are not modified at run time are assigned a control ID of –1. If you type an alphanumeric Control ID, Resource Workshop checks to see if an identifier exists as a *#define* or a *constant declaration.* If not, Resource Workshop asks if you want to create an identifier. See Chapter 14 for more information about identifiers.
Scroll Bar	Lets you choose whether you want horizontal or vertical scroll bars included with your control.

Most controls have certain attributes in common:

Table 15.2
Control attributes

Attribute	Description
Tab Stop	Lets the user press *Tab* to access this control.
Group	Identifies the first control within a group. See page 236 for details about grouping and accessing controls.
Visible	Determines whether the control is visible when the dialog box is first displayed. By default, this option is checked (WS_VISIBLE). If the option is not checked (NOT WS_VISIBLE), the control does not appear. The application can call the *ShowWindow* function at run time to make the control appear.
Disabled	Dims (grays) the control to indicate that it doesn't respond to user input.
Border	Draws a border around the control.

Each type of control Style dialog box has options that are specific to a particular type of control. These are mentioned in the discussion of the various types of controls, beginning on page 241.

Adding captions to controls

Although you can use a Style dialog box to add a caption to a control, you can also use the Properties dialog box.

To add a caption to a control, select the control and then do either of the following:

- Choose Options | Show Properties, then type a caption in the top text box.
- Double-click the control (dialog box) you want to add a caption to, then type a caption in the Caption text box.

Changing a control's class

If you are working with custom controls, you might find the Generic Control Style dialog box helpful. Display it by holding down *Ctrl* and double-clicking your control. Or use *Tab* to select the control, then hold down *Ctrl* and press *Enter*.

For more about custom controls, see page 251.

In the Generic Control Style dialog box, you can change the class of a control. You can also specify a caption, control ID, and style. If you type anything next to Info, the dialog box won't be compatible with the Microsoft Resource Compiler.

Specifying controls as tab stops

The Tab Set, Set Groups, and Set Order tools affect the user's keyboard interaction with the dialog box.

When using the keyboard, users typically press *Tab* to move from one control (or group of controls) to another. Some types of controls are automatically defined as tab stops when you add them to a dialog box. By setting the tab stop state of a control, you can manage the user's movement through the dialog box with the *Tab* key.

There are two ways to change tab stops:

- Use the Tab Set tool or the Set Tabs command.
- Use the Style dialog box.

To change or set tabs, you can use the Tab Set tool or the Set Tabs command.

1. Click the Tab Set tool or choose Options | Set Tabs. The cursor changes to the Tab Set icon. Resource Workshop surrounds any controls currently set as tab stops with a shaded outline.
2. To set a tab stop, click any control that is not surrounded by a shaded outline.

   To remove a tab stop, click a control that is surrounded by a shaded outline.
3. When you're through changing tab stops, click the Selector so you can return to editing your dialog box.

You can also use a Style dialog box to change a tab stop:

1. Open the Style dialog box for the control (double-click the control or select it and press *Enter*).
2. Check Tab Stop under Attributes to set a tab stop, or uncheck Tab Stop to toggle the setting off.

You can test your dialog box, as described on page 253, to see how your new tab stops work.

Grouping related controls

You can define groups of controls. This lets the user move among related controls using the arrow keys.

Defining groups is identical to specifying tab stops (described in the previous section). If you use the Set Groups tool or the Options | Set Groups command, Resource Workshop indicates that a control is marked as a group by surrounding it with a shaded outline. You can also define a group by setting the Group attribute in the control's Style dialog box.

Note, however, that you mark only the *first* control in each group with the Group attribute. Following the sequence in which you added the controls, Resource Workshop treats all subsequent controls as part of the group, until it encounters another control with the Group attribute.

There are two important things to remember about groups:

- The order in which you add the controls is important.
- When you select a "group" of controls by dragging a selection frame or by *Shift*-clicking, they are not a group in the sense used in this section; they are simply multiple selected controls. The Group attribute can be set only with the Set Group tool, the Options | Set Group command, or the Group check box in the Style dialog box.

Reordering controls (tab order)

You can specify the tab order in which users can access the controls in a dialog box. As noted in the previous section, the tab order is especially important when you've defined groups of related controls.

To specify the tab order of the controls in your dialog box,

1. Select the controls whose tab order you want to change.

   To specify the order for *all* controls in the dialog box, don't select any controls.
2. Click the Set Order tool. The mouse cursor turns into a Set Order icon.

Each control is numbered to show its current place in the overall order. If you chose just some of the controls in step 1, you'll see the order numbers only for those controls.

Note the `Next Item` prompt at the bottom of the Dialog editor. It tells you the order number that Resource Workshop assigns to the next control you click.

3. Click the items you want to assign new order numbers to. The Dialog editor displays a box around all the controls you've already picked.

   While assigning new order numbers, you can "step back" by re-clicking the last control you just clicked. The order will change to its previous number. You can continue to backtrack by clicking the controls in the reverse order that you originally clicked them.
4. When you finish assigning new order numbers, click the Selector so you can continue editing.

You can also order your controls with the menu command Options | Set Order. When you're done, choose Options | Modify Controls so you can continue to edit your dialog box. (You won't need to choose this command if you click all the selected controls.)

Aligning, resizing, and arranging controls

Once you've added controls to your dialog box, you can use the Dialog editor menu to align or resize controls and arrange them in rows and columns.

Aligning multiple controls

To align your controls with the Align command,

1. Select the controls you want to align.

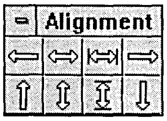
   Note that the selection frame surrounds all the selected controls. The individual controls that will be affected by the alignment options are indicated by shaded outlines.
2. Choose Align | Align. The Align controls dialog box appears.
3. Choose the Vertical Alignment and Horizontal Alignment options you want, then click OK to move the selected controls.

Table 15.3
Alignment options

Option	Description
Horizontal Alignment	
No Change	There is no change in horizontal alignment.
Left Sides	Aligns the controls so their left sides are on the left side of the selection frame.

Table 15.3: Alignment options (continued)

Centers	Aligns the controls so their horizontal centers are in the center of the selection frame.
Right Sides	Aligns the controls so their right sides are on the right side of the selection frame.
Space Equally	Moves the controls horizontally within the selection frame so the spaces between them are equal.
Center in Dialog	Moves the selection frame horizontally so it's centered in the dialog box. The relative position of the individual controls within the selection frame is unchanged.
Vertical Alignment	
No Change	There is no change in vertical alignment.
Tops	Aligns the controls so their tops are at the top of the selection frame.
Centers	Aligns the controls so their vertical centers are in the center of the selection frame.
Bottoms	Aligns the controls so their bottoms are at the bottom of the selection frame.
Space Equally	Moves the controls vertically within the selection frame so the spaces between them are equal.
Center in Dialog	Moves the selection frame vertically so it's centered in the dialog box. The relative position of the individual controls within the selection frame is unchanged.



Instead of using Align | Align Controls, you can use the Alignment palette. Select the controls you want to align and then choose a tool from the Alignment palette.

The one alignment option that does not appear in the Alignment palette is the Space Equally option. Of course, you can always use the Align Controls dialog box, but you can also space controls equally by "stretching" the selection frame:

1. Select the controls you want to space equally. A selection frame surrounds your selected controls.
2. Expand or "stretch" the selection frame by holding down the *Ctrl* key while dragging the edge of the selection frame in the direction you want to space your controls:
   - To space the controls equally in the horizontal direction, stretch the right or left border of the selection frame.

- To space the controls equally in the vertical direction, stretch the top or bottom border of the selection frame.

If you stretch a *corner* of the selection frame, the Form Controls into an Array dialog box appears, ready for you to arrange your controls in rows and columns.

Placing controls in columns and rows

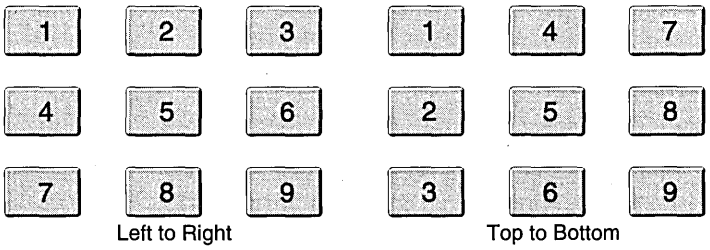
The Align | Array command arranges controls in columns and rows, aligning them horizontally or vertically, spacing them evenly horizontally or vertically, and renumbering them so they are all in sequence.

To use the Array command:

1. Select the controls you want to arrange in columns and rows.

   Note that the selection frame surrounds all the selected controls. The individual controls that will be affected by the Array command are indicated by shaded outlines.
2. If necessary, enlarge or reduce the size of the selection frame to enclose the area you want to fit the columns and rows into. For example, if you make the selection frame larger, the Array command will move the controls out to the new boundaries set by the selection frame.
3. Choose Align | Array or click the Duplicate tool.
4. Under Array Layout, specify the number of rows and columns you want.
5. Under Order, select how you want to order the controls in this group. The following figure shows how the two Order options would arrange nine controls into three columns.

Figure 15.5
Control order options



6. After you've chosen the options for the array, choose OK.

For multiple selected controls, the Duplicate tool has the same effect as Align | Array. When only a single control is selected, it has the same effect as Edit | Duplicate (see page 231).

Resizing multiple controls

You can resize multiple selected controls with the options in the Size Controls dialog box:

1. Select the controls you want to resize.

   Note that the selection frame surrounds all the selected controls. The individual controls that will be affected by the resizing options are indicated by shaded outlines.
2. Choose Align | Size. You'll see this dialog box:

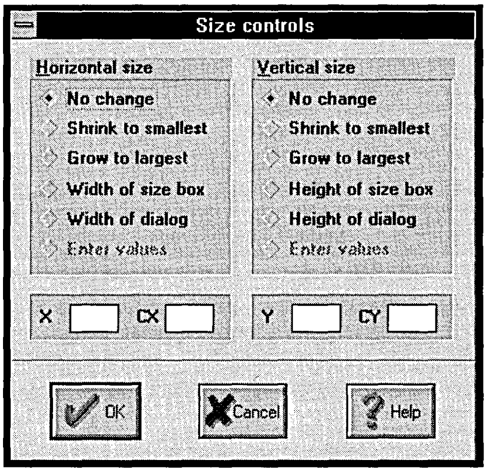


Figure 15.6
Size Controls dialog box

3. Choose the Vertical Size and Horizontal Size options you want and click OK to resize the selected controls.

Table 15.4
Size options

Option	Description
Horizontal Size	
No Change	There is no horizontal size change.
Shrink to Smallest	Reduces width of controls to match the least wide of the selected controls.
Grow to Largest	Increases width of controls to match the widest of the selected controls.
Width of Size Box	Resizes controls so they are as wide as the selection frame.
Width of Dialog	Resizes controls so they are as wide as the dialog box.
Vertical Size	
No Change	There is no vertical size change.

Table 15.4: Size options (continued)

Shrink to Smallest	Reduces height of controls to match the least tall of the selected controls.
Grow to Largest	Increases height of controls to match the tallest of the selected controls.
Height of Size Box	Resizes controls so they are as tall as the selection frame.
Height of Dialog	Resizes controls so they are as tall as the dialog box.

After you choose the vertical and horizontal sizing options you want, choose OK.

You can undo the sizing options by choosing Edit | Undo, by pressing *Alt+Backspace*, or by selecting the Undo tool.

Single-control sizing options

If you select a single control and then choose Align | Size, the following options are available:

- No Change (Horizontal and Vertical)
- Width of Dialog (Horizontal)
- Height of Dialog (Vertical)
- Enter Values (Horizontal and Vertical)

The No Change, Width of Dialog, and Height of Dialog options work as described in Table 15.4.

The Enter Values options let you specify both the size and position of the selected control. The X and Y values set the distance of the upper left corner of the control from the upper left corner of the dialog box (directly below the title bar). The CX and CY values set the width and height of the control. All values are measured in dialog units.

You can "undo" any editing you do in the Dialog editor, such as placing controls, aligning them, deleting controls, and so on, with the Undo tool or the Edit | Undo command. Undo works on commands that affect groups of controls as well as commands that change single controls.

Button controls

Button controls include radio buttons, push buttons, check boxes, and group boxes. You add buttons to your dialog box with the Tools palette (see page 228), and you use the Button Style dialog box to set the attributes of new buttons or to modify existing button controls. To display the Button Style dialog box, double-click the button control you want to modify.

Change the button type by choosing a new option under Button Type.

Table 15.5
Button types

Button type	Description
Push Button	A button containing text. The user clicks the button, which sends a BN_CLICKED message to the parent window.
Default Push Button	Identical to a push button, but also includes a bold border indicating that it's the default response if the user presses *Enter*.
Check Box	A rectangular button that can include text to the left or right of the button. The box is marked with an *X* when selected. It's the application's responsibility to check and uncheck the box.
Auto Check Box	Identical to a check box, but Windows does the checking and unchecking instead of the application.
3-State	Identical to a check box, but includes a third possible state: the button can be dimmed to show that its state is unknown or indeterminate. It's the application's responsibility to check, uncheck, and dim the box.
Auto 3-State	Identical to a 3-State button, but Windows does the checking and unchecking instead of the application program.
Radio Button	A circular button that has identifying text to the left or right. The circle is filled with a solid dot when selected. It's the application's responsibility to fill or clear the dot.
	Radio buttons must appear in groups. Usually, a group of radio buttons presents the user with a set of mutually exclusive options.
	When the user clicks a radio button, it sends a BN_CLICKED message to the parent window.
Auto Radio Button	Identical to a radio button, but Windows does the filling in or clearing of the dot instead of the application.
Group Box	A rectangular box that groups buttons together visually. You can also include a caption to display in the upper left corner of the group box.
User Button	Customized buttons for Windows 2.0 compatibility; we recommend that you don't use user button controls with Windows 3.0. Instead, you should use Owner Draw buttons.
Owner Draw	An option that allows the application itself to paint the button. The button sends a WM_DRAWITEM message to its parent when it needs painting.

The Alignment options determine if the text for the radio and check box buttons appears to the left or right of the button.

Push button Control ID values

Windows defines a set of control ID values for the standard push buttons used to exit dialog boxes. You can enter the ID name (which must be in uppercase letters) or ID value from Table 15.6. Note, however, that in contrast to the Borland Windows Custom Controls (described in Appendix B), changing the control ID of a standard Windows button does not automatically change the button text. For example, to create an OK button, you must change the control ID to IDOK *and* change the Caption string to OK.

Table 15.6 Predefined Windows button controls

ID name	ID value	Type
IDOK	1	OK
IDCANCEL	2	Cancel
IDABORT	3	Abort
IDRETRY	4	Retry
IDIGNORE	5	Ignore
IDYES	6	Yes
IDNO	7	No

Scroll bar controls

Windows, dialog boxes, and list boxes use scroll bars to indicate when there is more information than can currently be displayed. For example, if a file name list box can display ten names and there are twenty file names in the directory, a scroll bar indicates to the user the existence of the additional file names.

A scroll bar is a rectangle with direction arrows at each end. Between the arrows, a square icon (called the scroll box or *thumb*) indicates the approximate position of the display relative to the full range of information. For example, if the scroll box in a file name list box is halfway down the scroll bar, the user is looking at the file names halfway down the list of names.

You can add vertical and horizontal scroll bars anywhere you want in a dialog box.

To place scroll bars in your dialog box, use the Tools palette or the two scroll bar commands in the Controls menu. To set the scroll bars' attributes, display the Scroll Bar Style dialog box by double-clicking the scroll bar control you want to modify.

The Scroll Bar Style dialog box includes the common and the control-attribute options listed in Tables 15.1 and 15.2, as well as options that align the scroll bar inside the selection frame.

Table 15.7
Alignment options

Option	Description
None	The scroll bar fills the entire selection frame (default). If you resize the selection frame, you can change the scroll bar's proportions, making the arrow buttons and scroll box wider than usual.
Top Left	A horizontal scroll bar displays at the top of the selection frame and extends the full width of the frame. A vertical scroll bar displays at the left side of the selection frame and extends the full height of the frame. The scroll bar always appears in its standard width.
Bottom Right	A horizontal scroll bar displays at the bottom of the selection frame and extends the full width of the frame. A vertical scroll bar displays at the right side of the selection frame and extends the full height of the frame. The scroll bar always appears in its standard width.

List box controls

A list box is a rectangle containing a list of text strings. Usually, a user can browse through what's displayed in a list box, then select one or more items. The list box sends a message to the parent window about the selected item(s).

If the list of items exceeds the length or width of the list box, you can add scroll bars to the list box.

Other than the common options described on page 233, the List Box Style dialog box has Owner Drawing and List Box options.

Owner Drawing options let you determine whether the list contained in the list box should be drawn by the list box or the application. Choose one of the attributes in this table:

Table 15.8
Owner Drawing options

Option	Description
Not Owner Draw	The list box control draws the list (default).
Fixed	The application draws the list box in response to WM_DRAWITEM messages. The application can also respond to WM_COMPAREITEM, WM_DELETEITEM, and WM_MEASUREITEM messages. The list box control sends a WM_MEASUREITEM message to the application only when the list box is initially drawn, which fixes the list box item height.
Variable	The application draws the list box in response to WM_DRAWITEM messages. The application can also respond to WM_COMPAREITEM, WM_DELETEITEM, and WM_MEASUREITEM messages. The list box control sends a WM_MEASUREITEM message to the application for each item in the list box; each item can vary in height.

List Box options let you further define the list box. Choose one or more of the options in this table:

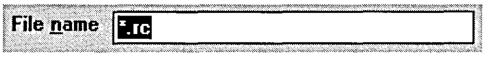
Table 15.9
List Box options

Option	Description
Notify	Sends an input message to the parent window when the user clicks on an item in the list (default).
Sort	Sorts the list alphabetically.
Multiple Select	Lets the user select more than one item at a time. The user can also toggle individual items on and off.
Don't Redraw	Prevents the list box from being redrawn when it is changed.
Tab Stops	Organizes the information in the list box in columns. The default column width is 32 dialog units or 8 characters. You should use *Tab* characters (\x09) to format the text. (If you want to change the column width, the application should set its own tab stops using the LB_SETTABSTOPS message.)
Integral Height	Causes the list box to decrease its height at run time, if necessary, to the nearest integral multiple of the current font height (default). For example, a list box might be drawn so that three items would display completely, but a fourth would be partially cut off. If Integral Height is turned on, the list box decreases its size at run time to the space required for three items (three times the font height).
Multi Column	Creates a list box in which the text wraps from column to column. The user scrolls the list box horizontally to display additional text. If you turn this option on, the application must send the LB_SETCOLUMNWIDTH message to set the width of all columns in pixels.
Pass Keyboard Input	Passes what the user types to the application.
Extend Select	When used with multiple-select list boxes, this attribute modifies the way multiple selection works, so that the user can select more than one item in the list.
Has Strings	If you've chosen either the Fixed or Variable Owner Drawing option, the list box stores text for each list item with the LB_INSERTSTRING or LB_ADDSTRING message. The list box can also retrieve list items from the message LB_GETTEXT.
Scroll Bar Always	(Windows 3.1 only) The list box always displays a vertical scroll bar, regardless of the number of items it contains. The WINDOWS.H constant for this style is LBS_DISABLENOSCROLL.

Edit text controls

An edit text control lets the user enter text from the keyboard. A common use of an edit text control is found in a File Open dialog box.

Figure 15.7
Edit text control



To set the attributes of an edit text control, double-click it in the dialog box. The Edit Text Style dialog box appears.

The Edit Text Style dialog box includes the common and control-attribute options listed in the tables beginning on page 233, as well as the following options:

- Alignment
- Scroll Bar
- Automatic Scroll
- Single- or multiple-line options
- Case sensitivity options
- Other options, including those specific to Windows 3.1

The following table describes the options in the Edit Text Style dialog box.

Table 15.10
Edit Text Style dialog box options

Option	Description
Alignment	
Left	Aligns multiple-line text to the left (default).
Right	Aligns multiple-line text to the right.
Center	Centers multiple-line text.
Scroll Bar options	
Horizontal	When this option is checked, the edit text control has a horizontal scroll bar at the bottom of its window.
Vertical	When this option is checked, the edit text control has a vertical scroll bar at the right edge of its window.
Automatic Scroll options	
Horizontal	When the user types a character at the right edge of the edit text boundary, the text automatically scrolls ten characters to the right. When the user presses *Enter*, the text scrolls back to the zero position.
Vertical	When the user presses *Enter* on the last line of the edit text control, the text scrolls up a full page. For example, if the control is five lines long, pressing *Enter* on the last line causes text to scroll up five lines; the cursor goes back to the top line.

Table 15.10: Edit Text Style dialog box options (continued)

	For this option to have any effect, you must also define the edit text control to allow for multiple lines. **Note:** Pressing the *Enter* key has no effect when Resource Workshop is in Test mode, but you will see this effect in your application.
Case options	
Case Insensitive	Displays text exactly as typed (default).
Upper Case	Displays all text in uppercase letters, regardless of how it's typed.
Lower Case	Displays all text in lowercase letters, regardless of how it's typed.
Line options	
Single Line	Limits the edit text to a single line (default).
Multiple Line	Lets the user type text on multiple lines. (To enable scrolling of multiple-line text, set the Vertical Automatic Scroll option to on.)
Other options	
Password	When Password is on, the letters being typed are not displayed. Instead, asterisks appear in their place. This is helpful for keeping passwords secret.
Convert OEM	Converts text typed into the control to the current OEM character set, then reconverts the text to ANSI. This option is useful in file input boxes because it ensures that any file name entered will be translatable into the OEM character set, which is used by the DOS file system.
Keep Selection	Keeps selected text highlighted, even when this control doesn't have keyboard focus. For example, if a user highlights text in an edit text control and then moves to another control, the text will no longer be highlighted, unless the edit text control has the Keep Selection attribute.

The following options are available only if you're running under Windows 3.1.

Table 15.11
Windows 3.1 styles

Type	**Description**
Read Only	The text is set to read-only. The WINDOWS.H constant for this style is ES_READONLY.
Want Return	The *Return* key forces a line break in a multiline edit text control that has keyboard focus. If the control doesn't have keyboard focus, the carriage return goes to the default push button. If the control doesn't have this flag, the user must press *Ctrl+Return* to create a line break. The WINDOWS.H constant for this style is ES_WANTRETURN.

Static controls

A static control displays text or art the user can't change. You can use static controls to label portions of your dialog box or to present information graphically.

The tools for these controls are illustrated in Figure 15.3 on page 228.

The static controls in the Tools palette include

- Static text
- Frame
- Icon
- Rectangle

You can set the attributes of static controls with the Static Style dialog box. To display the Static Style dialog box, double-click the static control in the dialog box.

The Static Style dialog box includes the common and control-attribute options listed in the tables beginning on page 233 (except for the scroll bar options), as well as several options specific to static controls.

The No Character Underline check box turns off character underlining. You can underline a text character in your static control by preceding it with an ampersand (&). If you check No Character Underline, underlining is disabled and ampersands are displayed as literal characters.

The Control Type options let you further define which kind of static control is displayed. Choose from the following options:

Table 15.12 Control Type options

Option	Description
Left Text	Displays text flush left within the control border (default). If text would extend past the edge of the frame, it automatically wraps to a new line.
	The text in all these styles uses the current Window Text color from the Control Panel.
Left Text—No Wrap	Displays text flush left within the control border. Any line of text that extends past the edge of the frame is clipped.
Centered Text	Displays text centered within the control border. If text would extend past the edge of the frame, it automatically wraps to a new line.
Right Text	Displays text flush right within the control border. If text would extend past the edge of the frame, it automatically wraps to a new line.
	In all static text but Simple Text, you can tab text by typing `\T`, and you can break text to a new line with `\R`.
Simple Text	Displays a single line of flush-left text. Doesn't take tab characters and can't be broken to a new line.
	Simple Text doesn't process the WM_CTLCOLOR message. In addition to receiving its text color from the Control Panel, its background uses the current Window Background color.

Table 15.12: Control Type options (continued)

White Rectangle	Displays a filled rectangle that uses the current Window Background color set in the Control Panel. The Windows default color for the Window Background is white.
Gray Rectangle	Displays a filled rectangle that uses the current screen background (Desktop) color set in the Control Panel. The Windows default color for the Desktop is gray.
Black Rectangle	Displays a filled rectangle that uses the current Window Frame color set in the Control Panel. The Windows default color for window frames is black.
White Frame	Displays an empty frame with a solid outline that uses the current Window Background color set in the Control Panel. The Windows default color for the window background is white.
Gray Frame	Displays an empty frame with a solid outline that uses the current screen background (Desktop) color set in the Control Panel. The Windows default color for the Desktop is gray.
Black Frame	Displays an empty frame with a solid outline that uses the current Window Frame color set in the Control Panel. The Windows default color for window frames is black.
	Note: When you add a frame to your dialog box, it might appear to be filled, using the current window background color. If you switch to Test mode, you'll see the frame as it will display at run time.
Icon	Displays an icon. Use the Edit Icon button to start the Bitmap editor so you can edit the icon.

Iconic static controls

Resource Workshop lets you display icons in a dialog box. The icon must be a part of the current project as an embedded or a linked resource.

To place an iconic static control in your dialog box,

1. Click the iconic static control in the Tools palette and drag it to your dialog box. Place the frame where you want the icon to appear.
2. Double-click inside the control's selection frame to display the Static Style dialog box.
3. In the Static Style dialog box, enter the name or identifier of the icon resource as the Caption and click the appropriate radio button—Text for a name, Number for an identifier.

   If the Caption text box does not contain the icon resource's name or identifier, the icon will not display.

   If you use an identifier—either its name or numeric value—as the Caption, you must select the Number option. For example, if you've created an identifier called GLOBE with a value of 1300, you can enter `GLOBE` or `1300` as the Caption, but in either case you must select Number.

   If the icon is called GLOBE and you didn't create an identifier, type `GLOBE` as the Caption and select the Text radio button.

4. Choose OK. The icon appears in your dialog box.
5. If you want to edit the icon, double-click the icon to display the Static Style dialog box once again. The Edit Icon button is now enabled. Click it to start the Bitmap editor so you can edit the icon.

Combo box controls

A combo box combines a list box (a control that lets the user browse and select strings) with either a static control (text a user can't change) or an edit text control (an area where a user can type).

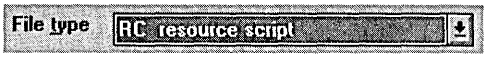


Figure 15.8
Combo box from Open File dialog box

You can set the attributes of a combo box using the Combo Box Style dialog box. To display the Combo Box Style dialog box, double-click the combo box in your dialog box.

The three Type options let you define the combo box.

Table 15.13
Combo box Type options

Option	Description
Simple	The drop-down list is always expanded to display items in the list, and the user can edit the items in the list (default).
Drop Down	When the dialog box is first displayed, the combo box consists of a single line of editable text. The user can click the down arrow to expand the list, and edit all items in the list.
Drop Down List	This option works just like a drop down, but the list is static. The user can select, but can't change anything in the list.

The Owner Drawing options let you determine whether the list contained in the list box should be drawn by the list box itself or by the application.

Table 15.14
Owner Drawing options

Option	Description
No	The list box control draws the list (default).
Fixed	The application draws the list box in response to WM_DRAWITEM messages. The application can also respond to WM_COMPAREITEM, WM_DELETEITEM, and WM_MEASUREITEM messages. The list box control sends the WM_MEASUREITEM message to the application only when the list box is initially drawn, which fixes the list box size.

Table 15.14: Owner Drawing options (continued)

Variable	The application draws the list box in response to WM_DRAWITEM messages. The application can also respond to WM_COMPAREITEM, WM_DELETEITEM, and WM_MEASUREITEM messages. The list box control sends the WM_MEASUREITEM message to the application for each item in the list box; the list box can therefore vary in size.
Has Strings	If you've chosen either Fixed or Variable, the list box stores text for each list item with the LB_SETTEXT message. The list box can also retrieve list items from LB_GETTEXT.

The Combo Box Style dialog box includes the common and control-attribute options listed in the tables beginning on page 233, as well as options specific to combo box controls.

Table 15.15
Combo box attributes

Option	Description
Vertical Scroll	Puts a vertical scroll bar in the list box.
Sorted	Automatically sorts items in a list box in alphabetical order.
Integral Height	Sizes the list box at run time so all items in the list are completely displayed (default). If you want to control the height of the list box precisely, uncheck this option.
OEM Conversion	Converts text the user types in to the current OEM character set, then reconverts the text to ANSI. This option is useful in file input boxes because it ensures that any file name entered will be translatable into the OEM character set, which is used by the DOS file system.
AutoHorizontal	Scrolls text to the left automatically when it exceeds the width of the control.
Vertical Scroll Always	(Windows 3.1 only) The combo box always displays a vertical scroll bar, regardless of the number of items it contains. The WINDOWS.H constant for this style is CBS_DISABLENOSCROLL.

Custom controls

If you want to use a control that doesn't fit into one of the predefined Windows types, you can use a *custom control*. The predefined controls discussed earlier in this chapter—list boxes, scroll bars, buttons, and so on—are called standard controls. They were developed by Microsoft and are part of Windows. A custom control, on the other hand, is any other window class you want to include in your dialog boxes.

There are two types of custom controls: those that you can install and those that are application-specific. You must implement installable custom controls using a dynamic-link library. Custom controls specific to an application are implemented in the application itself. Resource Workshop draws them as either gray boxes or empty frames.

Creating your own custom controls

If you want to create your own custom controls, you must design them and store them in dynamic-link library (DLL) files. Creating custom control classes is described in the online file CUSTCNTL.RW, which was copied to your hard disk by the Resource Workshop installation program.

Installing a control library (.DLL or .VBX)

Custom controls are stored in DLLs. To add custom controls to your dialog box, install the appropriate .DLL or .VBX files. Then the custom controls in that DLL will be available just like any standard Windows control.

To install a DLL file containing a custom control library,

1. Choose File | Install Control Library. You'll see the Install a New Control Library dialog box.
2. Specify the custom control .DLL or .VBX file.
3. Choose *OK.*

Now the controls contained in that DLL file are available for you to add to your dialog box. To edit certain custom controls (such as .VBX controls) display the Properties dialog box by either double-clicking the control or choosing Options | Show Properties.

Displaying custom controls

Before you add custom controls to your dialog box, choose Options | Preferences to see how your dialog boxes will be displayed. If the Drawing Type is set to Normal and the Draw Custom Controls as Frames option is checked, your custom controls will display as rectangular outlines. In that case, you might want to change either or both of these settings.

Adding a custom control

Once you've installed a DLL file containing custom controls, you can add any of those custom controls to your dialog boxes.

If your custom controls are of the type recognized by Resource Workshop (see page 227), their icons appear in the column (or columns) on the right side of the Tools palette, and you can select them directly from the palette.

If your custom controls are of the types not recognized by Resource Workshop, you must follow these steps:

1. Click the Custom Control tool or choose Control | Custom. The New Custom Control dialog box appears.

2. From the drop-down list next to class, choose the custom control you want. Resource Workshop displays a sample of the custom control you've selected in the middle of the dialog box.
3. When you've selected the custom control you want, choose OK. The mouse cursor becomes a cross hair, indicating that it's ready to place the custom control.
4. Click in the dialog box window where you want to place the custom control.

Testing a dialog box

To test your dialog box to see the effect of any changes you've made, select the Test tool or choose Options | Test Dialog.

You can press *Tab* and the arrow keys to see how you can move around your dialog box, or you can type text to see how text is scrolled in an edit text control. Check to see if your controls are in the order you want them.

When you test a dialog box, the status line at the bottom of the Dialog editor says `Test`.

To leave test mode and return to edit mode, do any of the following:

- Click the dialog box's OK or Cancel button.
- Choose Options | Test Dialog again.
- Press *Enter.*
- Click the Selector twice. (The first click switches focus from the dialog box to the Dialog editor.)

Viewing two dialog boxes

To view or compare two dialog boxes at the same time, follow these steps:

1. In the Project window, double-click the name of the first dialog box you want to view. The Dialog editor starts up and displays that dialog box.
2. Click the Test tool or choose Options | Test Dialog.
3. Click the Minimize button of the Dialog editor twice. (The first click switches focus from the dialog box to the Dialog editor window.) Your dialog box is now a floating, modeless dialog box you can move around like any window.
4. Return to the Project window and double-click the name of the second dialog box you want to view. A second Dialog editor starts up.

5. Again, click the Test tool or choose Options | Test Dialog in the second Dialog editor.
6. Click the Minimize button of the second Dialog editor twice.

Now you have two floating dialog boxes you can put side by side.

To exit Test mode, click the OK or Cancel button of your dialog box, or double-click its Control menu icon.

Customizing the Dialog editor

Resource Workshop lets you change the way some parts of the Dialog editor work. Choose Options | Preferences to display the Preferences dialog box.

- Status line units determine the unit of measurement the status line uses to display information.
  - *Dialog*. Uses the dialog unit as the unit of measurement on the status line. In a dialog unit, y equals $\frac{1}{8}$ of the font height, and x equals $\frac{1}{4}$ of the font width.
  - *Screen*. Uses a pixel as the unit of measurement on the status line.
- The Selection Border options let you change the appearance of the frame that surrounds selected controls.
  - *Thick frame*. The selection frame is thick, like the standard frame around a Windows application or dialog box window (default).
  - *Handles*. The selection frame is a rectangular outline with handles (small squares) at each corner and at the midpoint of each side.
- Drawing Type options determine how elements of your dialog box are displayed in the Dialog editor.
  - *Draft*. Draws each control as a rectangle with its control ID in the center. This option also lets you see how much space is occupied by the control's selection frame.
  - *Normal*. Draws standard Windows controls as they will appear at run time. Drawing of custom controls is determined by the Draw Custom Controls as Frames check box, described shortly.
  - *WYSIWYG* (the default option). With this option selected, Resource Workshop creates the dialog and control child windows and the controls draw themselves. This option is slowest, but the most accurate. Installable custom controls draw themselves.

- The Selection options "set the rules" for how you select controls. If you're working with closely spaced controls, you might want to turn these options on for greater precision and to avoid selecting controls inadvertently.
  - *Select Near Border.* If this option is checked, you must click *on* the control's border. If it is not checked, you can click anywhere inside the control's border.
  - *Selection Rectangle Surrounds.* If this option is checked, you must entirely surround the control (or controls) with the selection rectangle. If it is not checked, the selection rectangle need only touch the control (or controls).

For more information about the resource script language, use the Help system.

In the resource script language, each type of dialog control has a unique syntax. For example, centered static text uses the CTEXT statement. The CONTROL statement, however, can specify any type of dialog control. If you want Resource Workshop to generate only CONTROL statements in your resource script (rather than the specialized dialog control statements), select the Generate CONTROL Statements Only option.

The Draw Custom Controls as Frames check box is available only when the Drawing Type is set to Normal. When the option is checked, custom controls are drawn as empty rectangular outlines. When the option is unchecked, custom controls are drawn as gray rectangles with their text (if any) in a white rectangle in the center. Drawing custom controls as frames can speed up drawing of your dialog boxes on the screen.

If you check Ctl3d.dll, the Dialog editor uses the Windows 3-D look for controls such as radio buttons and check boxes.

C H A P T E R 16

Creating menus

Menus are lists of commands the user chooses from. Menu-driven applications remove the need for the user to remember a complex command-line syntax. Consequently, menus make an application easier to use.

Most Windows applications have a menu bar across the top of the screen that contains the names of the application's menus. Each menu contains a set of commands. For example, most Windows programs include a File menu with commands for creating, opening, saving, and printing files.

Resource Workshop's Menu editor makes it easy to create and edit menus for your application. Working with menus involves four basic steps:

1. Starting the Menu editor.

   If you are creating a new menu resource, the Menu editor presents you with a menu "template" to work on. If you're modifying an existing menu resource, it appears in the Menu editor.
2. Creating or editing the menu.
3. Testing the menu.
4. Saving the menu.

Menu terminology

This chapter uses the following terms to describe the elements of a menu resource:

- *Pop-up commands.* These commands cause menus to be displayed. Pop-up commands can appear in the menu bar, like the standard Windows File and Help menu names. They can also appear inside pop-up menus, where they cause another menu (called a "cascading menu") to be displayed.
- *Pop-up menus.* The rectangular boxes containing lists of application commands from which a user can choose. They come in two forms:

- *Drop-down menus.* These menus are displayed from the menu bar or from within a menu. They are tied to a pop-up command and are always displayed from that command's name. For example, the File drop-down menu always appears directly below the File pop-up command in the menu bar.
- *Floating menus.* These menus can appear anywhere in the application window. Their position is controlled by the *TrackPopupMenu* function.

See your compiler documentation for information about the *TrackPopupMenu* function.

- *Menu items.* The commands that appear in the menus—like Open, Save, or Print.
- *Menu separators.* The lines that divide the menu items into logical groups. Separators don't do anything other than make the menu easier to read and use. You can't edit menu separators.

Starting the Menu editor

The next two sections describe how to start the Menu editor to create a new menu or edit an existing one.

Creating a new menu

To create a new menu,

Chapter 14 describes how to open a project.

1. Make sure you've already opened the project you want to add the menu to. You can choose File | New Project to create a new project or File | Open Project to open an existing project.
2. Once you've opened a project, choose Resource | New to create a new resource for that project. The New Resource dialog box appears.
3. In the New Resource dialog box, scroll the Resource Type list to MENU, then either double-click MENU or click it and then click OK.

Resource Workshop displays the Menu editor with a default menu in it that you can begin editing.

Editing an existing menu

To edit an existing menu, open the project in which the menu is stored and do one of the following:

- Double-click the menu name in the Project window.
- Highlight the menu name and choose Resource | Edit.

Resource Workshop displays the Menu editor with the menu you have chosen loaded into it.

Menu editor screen

The Menu editor uses three panes to display editing information: an Outline pane that's similar to source script, a Test Menu pane, and an Attribute pane that lets you edit the currently highlighted line in the outline. You can change pane positions using the View command (see Table 16.1).

Attribute pane

The Attribute pane is where you edit pop-up commands and menu items, assign ID values, and set attributes for your menus and menu items. You can also define accelerator keys associated with each menu command. The statement you're editing is highlighted in the Outline pane.

Test Menu pane

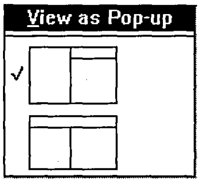
The Test Menu pane displays your menu and lets you test it.

The pop-up menu for the default Pop-up command contains a single command, Item. The Menu editor automatically updates this test menu as you make changes to the outline.

You can use the View menu to change how the test menu is displayed—both on the menu bar and relative to the other panes.

Table 16.1
View menu selections

Menu choice	Description
View as Pop-up	Controls whether the pop-up commands in the test menu are displayed on the menu bar or in a pop-up menu.
	By default this option is unchecked, and the pop-up commands in the test menu are displayed across the menu bar. Leave View as Pop-up unchecked if your menu resource contains the application's entire menu structure and you want it displayed as it would appear to the user.
	If you're working on a floating menu, check this option to display the test menu as it actually would appear. The command *Pop-up* appears on the menu bar, and you select Pop-up to display the menu itself.
First graphic	This graphic represents the default configuration of the panes, with the Test Menu pane over the Outline pane and to the right of the Attribute pane.
Second graphic	Check this graphic to put the Test Menu pane across the top of the edit window, like a normal menu bar.



Outline pane

The Outline pane shows you the pop-up commands, menu items, and separators of the new menu in pseudocode. The top line in the pane is the name of the menu, and the other lines are *statements* defining pop-up menus and menu items. You can right-click in this pane to view a SpeedMenu (or press *Alt+F10* from the pane).

The Outline pane's pseudocode is designed to make it easy for you to work with the structure of the menu. To see the complete code with all parameters for each statement, edit the menu's resource script (see "Editing menu resource scripts" on page 267).

The actual editing of the menu takes place in the Attribute pane. To move between the Outline pane and the Attribute pane, press *F6*.

To edit a statement, select it by doing any of the following:

- Press *Ctrl+↑* or *Ctrl+↓*.
- Choose a menu item from the Test Menu pane.
- Click a line in the Outline pane and press *F6*.
- Press ↑ or ↓ in the Outline pane and then press *F6*.

To move around inside the Attribute pane, you can use the mouse to position anywhere and make selections. Your selections take effect when you do any of the following:

- Press *Enter* to enter the change.
- Press *Ins* or choose Menu | New Menu Item to enter the change and insert a new item.
- Use *Ctrl+↑*, *Ctrl+↓*, or the mouse to move to another statement.
- Press *Ctrl+P* or choose Menu | New Pop-up to insert a new pop-up.
- Press *Ctrl+S* or choose Menu | New Separator to insert a new separator.

The Attribute pane options are described in Table 16.2.

Editing menus

Once you have a menu loaded into the Menu editor, you're ready to add new menu commands, pop-up menus, and separators, or to move, copy, and delete any part of the menu.

Adding new statements

For more information on adding menu items, see page 261.

To add a new statement to a menu (a pop-up menu, menu item, or separator), you must position the cursor in the Outline window on the line *preceding* where the statement is to go. To insert a statement at the beginning of the outline, highlight the top line (MENU_1 or the name of the menu resource).

When you've decided where the new statement is to go and you've highlighted the appropriate line, you can add a new statement by choosing one of the commands in the Menu menu.

Command	Action
New Pop-up	Inserts a new pop-up menu with a single item.
New Menu Item	Inserts a single item.
New Separator	Inserts a single separator.
New File Pop-up	Adds a complete generic File menu.
New Edit Pop-up	Adds an Edit menu with Cut, Copy, and Paste commands.
New Help Pop-up	Adds a predefined Help menu.
Check Duplicates	Described on page 266.

If you're adding a pop-up command that will appear first in the menu bar (the position typically occupied by the File menu), highlight the first line of the outline (MENU_1 or the name of the menu resource).

Adding menu items and separators

To add a new menu item or separator to the menu,

1. Decide where you want the new statement to appear in the menu and highlight the previous line in the Outline pane. The Menu editor inserts the new menu item or separator below the highlighted line.
2. Press *INS* or choose Menu | New Menu Item.
3. Type the name for the new menu item.
4. Press *Enter.*

You can also add three default menus to your project:

- Menu | New file pop-up adds a File menu as the first menu command. The new File menu contains the commands New, Open, Save, Save As, Print, Page Setup, Printer Setup, and Exit.
- Menu | New edit pop-up add an Edit menu as the second command from the left on the menu bar. The new Edit menu contains the commands Undo, Cut, Copy and Paste.
- Menu | New help pop-up add a Help menu as the last menu command (far right on the menu bar). The new Help menu contains the commands Index, Keyboard, Commands, Procedures, Using help, and About.

Editing menu items

A newly added menu item has the generic designation "Item". To make it useful, you must edit it.

When you first add a menu item, it is automatically selected, and you can edit it immediately. If you instead add other menu items, you must first select a menu item before you can edit it.

Using the Attribute pane

In the Attribute pane, use the mouse, *Tab*, or *Shift+Tab* to position on the field you want to edit.

The following table describes the selections you can make in the Attribute pane.

Table 16.2
Menu editor Attribute pane selections

Selection	Description
Item Text	The name of a pop-up menu or a menu item (command), and an optional description of its accelerator, if it has one.
Item help	A string that describes the menu item. This text is stored in the string table for your project (if you don't have a string table resource, it is added when you use this option). You can use this text as menu help on a status bar.
Item ID	A unique ID for a menu item. This text box is not available when a pop-up menu or menu separator is selected.
Item Type	Can be a pop-up menu, a menu item, or a menu separator (a line).
Break Before	Controls the format of menu commands in the menu bar and in pop-up menus. Choose one of the following options:
– No Break	There is no break before this command.
– Menu Bar Break	Starts a new line in the menu bar. In a pop-up menu, starts a new column and draws a vertical line to separate the columns.
– Menu Break	Starts a new line in the menu bar or a new column in a pop-up menu.
– Help Break	Moves the pop-up command or menu item to the far right of the menu bar. Use this option only with top-level statements that display in the menu bar.
Initial state	Controls the initial state of the menu command. Choose one of the following options:
– Enabled	In its initial state, the command is enabled. Use the *EnableMenuItem* function to change the state of the menu item.
– Disabled	In its initial state, the command is disabled. The user can't distinguish between Enabled and Disabled commands; they look the same on the menu. Use the *EnableMenuItem* function to change the state of the menu item.
– Grayed	In its initial state, the command is disabled and its text is grayed. The shading lets the user know the command is not currently available. Use the *EnableMenuItem* function to change the state of the menu item.
– Checked	Places a check next to the command. Choose this option if the item will function as a toggle and you want the command to initially appear

Table 16.2: Menu editor Attribute pane selections (continued)

	checked. Use the *CheckMenuItem* function to change the state of the command.
Key	Controls the keyboard shortcut for the menu item. You can either type a keyboard identifier (as defined in WINDOWS.H), or you can choose Menu|Accelerator key value (this is available on the SpeedMenu by right-clicking the MENUITEM) and press the key you want. For example, to assign the Home key to a menu item, select the menu item then right-click it. Choose Accelerator key value, then press the Home key. VK_HOME appears in the Key box and the Invert menu item modifier is checked. Press *Esc* or click the mouse to return to normal editing.
Modifiers	Define the accelerator keys for a menu item. Choose one of the following options:
– Alt	Uses Alt for the accelerator key combination (for example, Alt+W).
– Shift	Uses Shift for the accelerator combination (for example, Shift+F1).
– Control	Uses Control for the accelerator key combination (for example, Ctrl+W).
– Invert	Disables the flash feature—a built-in Windows function that flashes a menu-bar command when the user presses the accelerator key associated with that menu-bar command. This feature lets the user know which menu the accelerator key is on. Invert Menu Item is on by default when you create an accelerator.
Key type	Describes the type of key accelerator for a menu item. Choose one of the following options:
– ASCII	Defines the key accelerator as a standard ASCII key. All ASCII keys must be surrounded by quotation marks. A caret (^) indicates that the key is combined with the Ctrl key. Typically, you don't use single ASCII characters as accelerator keys; instead you combine them with Alt or Ctrl, such as Alt+R or Ctrl+L.
– Virtual key	A virtual key is typically a function key, an arrow key, or an editing key such as Home or End. Windows has predefined characters for virtual keys. These identifiers all start with VK_ and are defined in WINDOWS.H. You don't need to look up virtual key identifiers if you use Menu|Accelerator key value to insert the key. Virtual keys have no provision for Ctrl, Shift, and Alt combinations. If you use these keys, you need to check the appropriate Modifier check box in the Accelerator editor dialog box.

Entering item text

The item text is the menu name or command that appears in the menu bar or the menu. When the Item Text selection is highlighted, you can type a new text string directly into the box. You can also use the *Home, End,* and arrow keys to move the cursor in the text box.

If you want the user to be able to choose the menu or command by typing a letter in the menu or command name, put an ampersand (&) immediately before that letter. Windows will display the text with that letter underlined.

See Chapter 17 for more information on accelerators.

To link the *Ctrl+F4* accelerator to the new Stored Order command, add the accelerator text to your menus.

- Use the tab character (`\t`) to separate the menu title from the accelerator text with a tab (for example, `&Stored Order\tCtrl+F4`).
- Use the right-align character (`\a`) to right-align accelerator text (for example, `&Stored Order\aCtrl+F4`).

Windows applications generally use the plus sign to show key combinations, like *Shift+Del* or *Ctrl+Shift+F4.*

Entering item IDs

All menu items must be uniquely identified. When you add a new item, Resource Workshop automatically assigns an item ID that's different from all the other item IDs in this menu resource. You can accept this value, or you can replace it with another unique number or unique name.

See page 210 for a detailed discussion of identifiers.

If you type a name, Resource Workshop checks to see if an identifier by that name exists. If not, you see a dialog box asking if you want to create an identifier. Click OK. If you want to use the name as an item ID, you must create an identifier. From this point, create the new identifier as described in Chapter 14 (page 213).

Moving and copying statements

You can use Cut, Copy, and Paste on the Edit menu to move and copy the statements in the outline of the Menu editor.

To move a statement, highlight the statement and choose Edit | Cut. Note that you can't cut the last (or only) statement from the outline. There must always be at least one menu item, pop-up command, or menu separator in the outline. To insert the cut or copied statement into your menu, highlight the statement immediately before the point at which you want the statement to appear, and choose Edit | Paste.

To copy a statement, highlight it and choose Edit | Copy. The highlighted statement remains in the outline.

Deleting menu statements

Highlight the statement you want to delete, then press *Del*, Edit | Delete, or Edit | Cut.

Note the following about deleting menu statements:

- If you delete a `POPUP` statement, you delete the pop-up command it defines and all the items contained in the pop-up menu.
- You can't delete `_End Popup_` statements.
- You can't delete the last (or only) statement from the outline. There must always be at least one menu item, pop-up command, or menu separator in the outline.

Creating floating menus

A floating menu can be displayed anywhere in the application's window space; it is not tied to the menu bar.

➡ Each floating menu must be saved as a separate menu resource within the project file.

To create a floating menu,

1. Choose Resource | New and select Menu from the New Resource dialog box.
2. Choose View | View as Pop-up to see the floating menu as it will appear on the screen at run time.

   When you view the menu in the Test Menu pane, it will still appear tied to the menu bar, but as long as your code uses the *TrackPopupMenu* function correctly, the menu will float at run time.
3. Select the first line of the outline (MENU_1 or the name of the menu resource).
4. Press the *INS* key to add at least one menu item at the top of the outline.
5. Select the string `POPUP "Pop-up"` in the outline.
6. Press the *DEL* key to delete the `POPUP` statement, its menu item, and the `_End Popup_` statement.
7. Add any additional menu items you want.
8. Edit the menu items in the Attribute pane.
9. Save your project.

Testing menus

The Menu editor gives you immediate testing capability. The test menu is updated as you make changes to your menu, and you can display the menu at any time to check on its current appearance. As you choose menu items, the item is highlighted in the Outline pane and its properties appear in the Attributes pane. You can turn off this feature by unchecking Menu | Track test menu.

The Menu editor also has a built-in debugging tool that you can use to test for duplicate menu item IDs. If you choose Menu | Check Duplicates, the Menu editor searches for duplicates and, if it finds any, displays a dialog box with the message "Duplicate command value found."

When you close this message box, the Menu editor highlights the statement that contains the duplicate value. You must do one of the following:

To see the menu IDs in your resource, choose Resource | Edit as Text.

- If the item ID is a number, enter a new number that doesn't conflict with the other item IDs.
- If the item ID is an identifier, the Item ID box contains a text string, and the box to the right of the Item ID contains an integer. To change the identifier value,

  1. Choose Resource | Identifiers to display the Identifiers dialog box.
  2. Scroll down the list of identifiers until you find the one you want.
  3. Click the Change button and type a new value that doesn't conflict with the other item IDs.
  4. Click OK or press *Enter* to change the value.
  5. Click in the Menu editor window to continue editing your menu. (You can leave the Identifiers dialog box open for later use.)

For example, if you assign the value 101 to two identifiers, *wmnu_List* and *wmnu_Asc*, Menu | Check Duplicates would return the message "Duplicate command value found," and the Menu editor would highlight *wmnu_Asc* (the second of the two identifiers). As described above, you would then change the value of *wmnu_Asc* to something other than 101, 102, or 104 (the values of the other identifiers in the menu).

Editing menu resource scripts

To work with the resource script of a menu, select the menu name from the Project window by clicking it, then choose Resource | Edit As Text to display the resource script in the internal text editor.

See page 205 for a description of the internal text editor.

For example, to edit the resource script for the sample menu you'll create in the next section, you can open the project containing that menu, highlight the menu, and choose Resource | Edit As Text. Resource Workshop opens its internal text editor and displays the source code as follows:

See the online Help index for a description of the resource script options for menus.

```
MENU_1 MENU
BEGIN
  POPUP "&Widgets"
  BEGIN
    MENUITEM "&List\tCtrl+L", wmnu_List
    MENUITEM "&Add...\tCtrl+A", wmnu_Add
    MENUITEM SEPARATOR
    POPUP "A&rrange List"
    BEGIN
      MENUITEM "&Ascending\tCtrl+F2", wmnu_Asc
      MENUITEM "&Descending\tCtrl+F3", wmnu_Desc
    END
  END
END
```

Use the editor to make changes directly to the resource script. For example, to change two of the menu's memory options from LOADONCALL and MOVEABLE (the defaults) to PRELOAD and FIXED,

You can also make this change by choosing Resource| Memory Options.

1. In the text editor, alter the first line of the script to read:

   ```
   MENU_1 MENU PRELOAD FIXED
   ```

2. To compile what you just entered and see if it's correct, choose Compile | Compile Now.

   The Compile menu is available only when you are in the text editor. If you return to the Menu editor and then choose Resource | Edit as Text to switch to the text editor again, you'll see that Resource Workshop has inserted one of the default memory options into the script. The first line of the script now reads as follows:

   ```
   MENU_1 MENU PRELOAD FIXED DISCARDABLE
   ```

3. If you want to exit the Menu editor, choose the Close command from the text editor window's Control-menu box. Resource Workshop asks if you want to compile. When you click Yes, Resource Workshop compiles the menu and exits to the Project window.

Don't spend any time inserting comments in your resource script or formatting the text, because the Resource Workshop incremental compiler does its own formatting and discards all comments.

Sample menu

This section takes you through the creation of a simple pop-up menu called Widgets, first with the Menu editor, and then with a text editor.

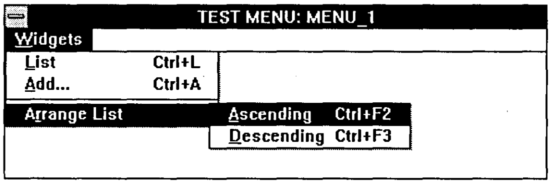


Figure 16.1
Sample menu

The first two commands in the menu (List and Add) let users look at a list of existing widgets or add new widgets. The third command, Arrange List, produces a pop-up menu with two additional commands (Ascending and Descending) that let users choose the sort order of the list of widgets.

Widgets and Arrange List are pop-up commands. Widgets displays the Widgets menu, and Arrange List displays the cascading menu that contains the Ascending and Descending commands.

List, Add, Ascending, and Descending are menu items (or commands). When the user clicks any of these commands, the application performs an action. The ellipsis (three dots) after the Add command indicates that the application displays a dialog box when the user chooses this command. Note the menu separator that groups the List and Add commands separately from the Arrange List pop-up command.

Creating the sample menu

The Menu editor has several features that make creating the sample menu much easier than creating it with a text editor. Among other things, the Menu editor handles menu IDs for you and stores identifiers in a separate identifier file (if you have created one). In addition, you can test the menu as you create it.

To create the sample menu,

Chapter 14 describes how to open a project.

1. Make sure you've already opened a project.
2. With a project open, choose Resource | New.
3. Resource Workshop displays the New Resource dialog box. Scroll the Resource Type list until you see MENU and then double-click.

   Resource Workshop adds a new menu resource to the Project window, then displays the new menu in the Menu editor with the first statement in the outline (`POPUP "Pop-up"`) highlighted.
4. To rename the initial menu statement from Pop-up to Widgets, type `&Widgets` in the Item Text text box in the Attribute pane and press *Enter.*

   The Menu editor updates both the test menu and the outline.

   ➡ In the test menu, note that the *W* in Widgets is underlined, indicating that you can press *Alt+W* to display the Widgets menu.

Adding commands to the menu

Next, add the commands to the Widgets menu.

1. To rename the first menu item and add text indicating the accelerator, press *Ctrl+↓* to highlight the second line of the outline (`MENUITEM "Item"`), and type `&List\tCtrl+L` in the Item Text text box.
2. Tab to the Item ID text box and type `wmnu_List` to enter an identifier for this command. Press *Enter.*

   Resource Workshop asks if you want to create a new identifier. Press *Enter* to display the New Identifier dialog box, and immediately press *Enter* to accept the value displayed.
3. With the List command highlighted, add a new menu command either by pressing *Ins* or by choosing Menu | New Menu Item.
4. Type `&Add...\tCtrl+A` in the Item Text box to change the text for the new menu item.
5. Create an identifier for the item by tabbing to the Item ID field, typing `wmnu_Add`, pressing *Enter,* and responding to the prompts as before.
6. Press *Ctrl+S* to put a separator after the Add command.
7. With the Add command highlighted, press *Ctrl+P* to add a new pop-up menu. Change the text to `A&rrange List`.

   Because you want additional commands to appear when the user chooses Arrange List, you define it as a pop-up command rather than as another menu item. A pop-up command in the middle of a menu creates a cascading menu.

Adding commands to the Arrange List menu

To define the two menu commands in the Arrange List pop-up menu,

1. Press *Ctrl+↓* until the first menu item in the Arrange List menu is highlighted.
2. Change the menu item `"Item"` to `"&Ascending\tCtrl+F2"`.
3. Create an identifier *wmnu_Asc* for this command.
4. Press *Ins* to add a new menu item after `"&Ascending"`, then rename it `"&Descending\tCtrl+F3"`.
5. Create an identifier *wmnu_Desc* for this command.
6. Save the project.

Testing the menu

Test the menu by clicking on the Widgets command in the Test Menu pane and dragging down to the Arrange List command. Your menu should look like Figure 16.1 on page 268.

You can also test for duplicate values in menu IDs by choosing Menu | Check Duplicates.

- If there are no duplicates, you get the message "No duplicates found."
- If there are duplicates, you get the message "Duplicate command value found." The "Testing menus" section on page 266 tells how to correct the duplicate values.

C H A P T E R 17

Creating accelerators

An *accelerator* is a hot key—a key combination the user presses to perform a task with your application. It substitutes for a menu command and, just like a menu command, creates a WM_COMMAND or WM_SYSCOMMAND message that tells the application what to do next.

Usually you create accelerators to duplicate commands on pop-up menus. For example, if you open the Edit menu in many Windows applications, you see these accelerators: *Alt+Backspace* (Undo), *Shift+Del* (Cut), *Ctrl+Ins* (Copy), and *Shift+Ins* (Paste).

You store accelerator definitions in an accelerator table (the accelerator resource). Each entry in the table is an accelerator that defines the key combination a user must press and the command it produces. If you like, you can create multiple accelerator tables (or resources) for different parts of your menu.

The Accelerator editor can create and edit accelerators for your application. Working with accelerators involves five basic steps:

See page 277 for an example that shows how to create an accelerator table.

1. Starting the Accelerator editor.
2. Starting the Menu editor. The Menu editor lets you define accelerator keys for menu items.
3. Creating or editing an accelerator table.
4. Testing the accelerator table for duplicate keys.
5. Saving the accelerator table.

Accelerator table key combinations

The key combinations in your accelerator table can use *ASCII keys* or *virtual keys*.

- An ASCII key is one that can be displayed—typically an alphanumeric character or punctuation mark.

- A virtual key is a function key, an arrow key, or an editing key like *Home* or *PgDn*. Although in some cases these keys might display characters on the screen, there's no standard that specifies which characters appear.

ASCII keys

All ASCII keys must be surrounded by quotation marks. A caret (^) indicates that the key is combined with the *Ctrl* key. The Alt check box in the Attribute pane indicates if the key is combined with the *Alt* key.

For example, both *Ctrl+W* and *Ctrl+Alt+W* are represented in the Outline pane as "^W", but there is also a check mark in the Alt check box in the Attribute pane for *Ctrl+Alt+W*.

➡ Typically, you wouldn't use a single ASCII character as an accelerator key; instead, you'd combine it with *Alt* or *Ctrl* (*Ctrl+L* or *Alt+L* instead of *L* alone).

Virtual keys

Windows has predefined identifiers for virtual keys—such as VK_BACK for *Backspace* and VK_F1 for *F1*. These identifiers, which all start with *VK_*, are defined in WINDOWS.H.

The Modifiers check boxes in the Attribute pane show if the key is combined with *Ctrl, Alt, Shift,* or any combination of the three.

For example, you could have two VK_F1 accelerators in your table. The first might be *Ctrl+F1* (the Control check box is checked) and the second *Shift+F1* (the Shift check box is checked).

➡ You needn't look up any of these virtual key identifiers if you use Key Value mode to insert the key (see page 276), because the Accelerator editor looks up the correct value and inserts it for you.

Starting the Accelerator editor

How you start the Accelerator editor depends on whether you're creating a new accelerator table or editing an existing one.

Creating a new accelerator table

You can create a new accelerator table in a new project or in an existing one.

To start the Accelerator editor and create a new accelerator table,

Chapter 14 describes how to open a project.

1. Choose File | New Project to create a new project or File | Open Project to open an existing project.
2. Choose Resource | New to display the New Resource dialog box.

3. In the New Resource dialog box, double-click ACCELERATOR in the Resource Type list.

Resource Workshop displays the Accelerator editor with an accelerator table template you can begin editing.

Editing an existing accelerator table

To start the Accelerator editor and edit an existing accelerator resource, open the project in which the accelerator resource is stored and do either of the following:

- Double-click the accelerator resource name in the Project window.
- Highlight the accelerator resource name in the Project window and choose Resource | Edit.

Resource Workshop displays the Accelerator editor with the accelerator table you have chosen loaded into it.

Running the Menu editor at the same time

When working with accelerators, it's a good idea to start the Menu editor and load in the menu containing the associated commands. That way you can see the command text and item IDs you'll need when you define the accelerators. As explained in the section "Setting the command value" (page 276), each accelerator must have an identifier that corresponds to a command on the menu.

For a description of how you add accelerators to your menus, see "Entering item text" on page 264 and "Adding commands to the Widgets menu" on page 269.

Using the Accelerator editor

The Accelerator editor screen is divided into two panes, the Outline pane and the Attribute pane. You can move between these two panes by using the mouse or the *F6* key.

Outline pane

The Outline pane shows you, in outline script form, all the accelerators defined in the table. The top line of this outline is the name of the accelerator table. The lines below it are accelerator entries, which have two parts:

See page 271 for an explanation of ASCII and virtual keys.

- The first part identifies the key that is used as an accelerator. It is either an ASCII key or a virtual key.
- The second part is the item ID of the command to which the accelerator is associated. This ID is either an integer or the name of an identifier.

To select an accelerator in the Outline pane, use the mouse or the arrow keys. You can also right-click in the Outline pane to view a SpeedMenu (or press *Alt+F10* from within the pane).

Attribute pane

Selecting an accelerator in the Outline pane shows its settings in the Attribute pane. With an accelerator selected, you can make changes to it in the Attribute pane, such as entering a new key combination or associating the accelerator with another command.

From the Attribute pane, you can use the mouse or press *Ctrl+↑* or *Ctrl+↓* to select an accelerator.

You can use the mouse to move around inside the Attribute pane and to make selections. In addition, you can use the following keys:

- *Tab* moves you forward through the Attribute pane, and *Shift+Tab* reverses direction. Note that each of the Modifiers check boxes is a tab stop.

  ➡ If you tab from the Command text box to the Key text box, the Accelerator editor changes to Key Value mode, in which you can press any key to enter it as an accelerator. To exit from this mode, click the mouse or press *Alt+Esc.* Key Value mode is described on page 276.
- The arrow keys select among the Key Type radio buttons, and the *Spacebar* toggles each of the Modifiers check boxes.

Your selections take effect when you press *Enter* (to change the accelerator) or *Ins* (to create a new accelerator), or when you move to another accelerator in the outline.

The following table describes the selections you can make in the Attribute pane.

Table 17.1 Attribute pane selections

Selection	Description
Command	The item ID (integer or identifier) for the command the accelerator is to execute. This value must match the value in the associated menu resource.
Key	The accelerator key. You can enter the key manually (entering quotation marks for ASCII keys and the proper syntax for virtual keys) or in Key Value mode (the Accelerator editor decides if it's an ASCII or virtual key and enters it for you in the appropriate format). If you tab into this text box from the Command text box, you're automatically in Key Value mode. See page 276 for a description of Key Value mode.
Key Type	Either ASCII or Virtual Key. In Key Value mode, the Accelerator editor sets these radio buttons automatically.
– ASCII	The accelerator uses an ASCII key (see page 271).

Table 17.1: Attribute pane selections (continued)

– Virtual Key	The accelerator uses a virtual key (see page 271).
Modifiers	The following descriptions of these check boxes tell what each option means if it's checked.
– Alt	The accelerator includes the *Alt* key (for example, *Alt+W*).
– Shift	The accelerator includes the *Shift* key (for example, *Shift+F1*).
– Ctrl	The accelerator includes the *Ctrl* key (for example, *Ctrl+F3*).
– Invert Menu Item	Using the accelerator causes the associated menu bar command to flash (to invert momentarily).

Editing an accelerator table

Once you have an accelerator table loaded into the Accelerator editor, you're ready to begin editing it. Using the Accelerator editor, you can define and change accelerators, and you can specify an accelerator key combination by pressing the desired key combination. You can also copy or delete any accelerators in the table, and you can test the table for duplicate identifier values.

Adding an accelerator key

To add a new accelerator to the accelerator table, press *Ins* or choose Accelerator | New Item. The new key appears in the outline below the currently selected line with the default values of 0 (zero) for the key value and a unique integer value for the command ID.

When you add a new accelerator, the editing focus automatically switches to the Attribute pane.

Selecting an accelerator key

To select an accelerator, you can do any of the following:

- Press *Ctrl+↑* or *Ctrl+↓* to highlight the accelerator in the Outline pane and automatically switch editing focus to the Attribute pane.
- Click the mouse on the accelerator in the Outline pane and then press *F6* to switch editing focus to the Attribute pane.
- If you're already in the Outline pane, use the arrow keys to select the accelerator and then press *F6* to switch editing focus to the Attribute pane.

Using the Attribute pane

The Attribute pane has text boxes, radio buttons, and check boxes that let you define the accelerator. The Attribute pane options are described in Table 17.1.

Setting the command value

In the Command text box, type the item ID (either an integer or an identifier) for the command the accelerator is to execute.

The Command string is automatically highlighted when you select an accelerator with *Ctrl+↑* or *Ctrl+↓* or you add an accelerator with *Ins*. Type the item ID directly into the text box.

If you're tying the accelerator to a command in an existing menu, start the Menu editor and load the menu resource. Note the command's item ID, and use that same ID in the Command text box for the accelerator.

Note the following about identifiers:

- If you enter an existing identifier name and see the "Create a new identifier:" dialog box, you've probably mistyped the name. Click No and check your spelling.
- If you deliberately enter a new identifier because you intend to add the associated menu item to the menu later, Resource Workshop asks if you want to create a new identifier. Click Yes or press *Enter*, and then enter a unique identifier value in the New Identifier dialog box.

  If you don't see the "Create a new identifier:" dialog box, the identifier already exists. Enter a unique identifier before you continue.

Specifying the accelerator key

To specify the accelerator key combination, enter the key combination in the Key text box.

Your accelerator should be consistent with the accelerators in other Windows applications, so don't use any key combinations required by Windows (such as *Ctrl+Esc*). For guidelines about choosing appropriate key combinations, see IBM's *Systems Application Architecture Common User Access Advanced Interface Design Guide.*

You can enter the key in either Key Value mode or manual mode.

Key Value mode. In Key Value mode, any key or key combination you press is automatically entered in the Key text box as the accelerator. The Accelerator editor determines if the key is an ASCII key or a virtual key and selects the correct Key Type radio button. The Accelerator editor also checks the appropriate Modifiers check boxes.

Manual mode. In Manual mode, you provide all the information that defines the accelerator. You must decide if the key is an ASCII key or a virtual key. If it's a virtual key, you have to know the correct Windows identifier and type it in uppercase letters. You must also select the appropriate Key Type radio button (ASCII or Virtual Key) and check the appropriate combination of the *Alt, Shift,* and *Ctrl* check boxes.

Flash feature

Windows has a built-in function that *flashes* a menu-bar command when the user presses an accelerator key for a command associated with the menu-bar command.

For example, if you've selected a block of data in many Windows applications, pressing *Shift+Del* (the equivalent of choosing Edit | Cut) causes Windows to temporarily invert (flash) the Edit command on the menu bar. This feature lets the user know which menu the accelerator is on.

The flash feature (also called *invert menu item*) is on by default when you create an accelerator. You can disable this feature by unchecking the Invert Menu Item option in the Accelerator editor Attribute pane.

Checking for duplicate key combinations

To ensure that you don't use the same key combination more than once, Resource Workshop lets you debug an accelerator table by searching for duplicate key combinations, as follows:

1. With an accelerator table open, choose Accelerator | Check Dup Keys.
2. If two accelerators use the same key combination, the Accelerator editor displays the message "Duplicate key value found" and highlights the second accelerator. Make your changes and continue debugging your accelerator table with Check Dup Keys until you see the message "No duplicate key values found."

Creating a sample accelerator table

In this section you'll create an accelerator table for the Widgets menu described on page 268. Without Resource Workshop, you'd have to use a text editor or word processor to create the resource script in the previous section.

➡ If you didn't save the sample menu from Chapter 16, you can still work through this section. If you did save the sample menu, several of the steps have additional or alternate instructions for you.

To create an accelerator table with Resource Workshop's Accelerator editor, do the following:

Chapter 14 describes how you open a project.

1. Open a project file or, if you saved the sample menu from Chapter 16, open the project that has your Widgets menu.
2. Choose Resource | New.
3. In the New Resource dialog box, double-click the ACCELERATOR resource type. You see the Accelerator editor with one new entry in it.

   If you saved the sample menu from Chapter 16, open the Menu editor by double-clicking on its name in the Project window.

   Resize the windows for the Menu editor and the Accelerator editor so you can see both at the same time.

   In the Menu editor Outline window, highlight the List menu item and note its item ID (*wmnu_List*) and the accelerator you intended for it (*Ctrl+L*).

   Click on the new accelerator in the Accelerator editor. If necessary, press *F6* to highlight the Command text box.
4. In the Command text box, enter the name of the identifier for the first command in the menu (*wmnu_List*).
5. Tab to the Key text box.

   Note that you are now in Key Value mode. Press *Ctrl+L*. The Accelerator editor enters the ASCII value "^L" and selects the ASCII radio button for you.
6. Press *Alt+Esc* to exit from Key Value mode, then press *Enter* to cause these settings to take effect on the highlighted accelerator key in the Outline pane.

   If you're not working with an existing project, Resource Workshop asks if you want to create an identifier for this accelerator. Click Yes, and then click OK in the New Identifier dialog box to accept the default value.
7. Press *Ins* to create a new accelerator.

   If you're working with the Widgets menu from Chapter 16, click on the Menu editor, select the next command with an accelerator key, and note its item ID and accelerator key.
8. Add the remaining accelerators—*wmnu_Add*, *wmnu_Asc*, and *wmnu_Desc*—substituting the appropriate identifier and key combination, until all the accelerators are defined.
9. Save the project.

The accelerator table is now finished. In the next sequence of steps, you'll deliberately create a duplicate key value so you can see how to debug your accelerator table with the Accelerator editor.

1. Highlight the second accelerator and change its key value from *Ctrl+A* to *Ctrl+L.*

   Press *Ctrl+↑* to highlight the accelerator, then press *Tab* to enter Key Value mode.
2. Type `Ctrl+L`, press *Alt+Esc,* and press *Enter* to make the change take effect.
3. Choose Accelerator | Check Dup Keys. Resource Workshop displays the "Duplicate key value found" dialog box. Press *Enter* to close it.
4. Now highlight *wmnu_Asc* or *wmnu_Desc* in the Outline pane and press *Del.*
5. Display the Edit menu. Note that there's a choice, Undo Delete Item, that refers to your deletion of the last accelerator.
6. Choose Undo Delete Item to restore the deleted accelerator.
7. Display the Edit menu again. The Undo command now says Undo Change Item. When you choose this command, the accelerator for the Add command changes from *Ctrl+L* back to *Ctrl+A.*
8. Choose Accelerator | Check Dup Keys again. You should get the message, "No duplicate key values found."

This example shows you how easy it is to create accelerators by using both the Menu editor and the Accelerator editor. You can switch back and forth between the two editors to see which accelerator keys are associated with which menu commands, and you can use Key Value mode to enter the keys, letting the Accelerator editor do most of your work for you. When you're done, you can check to ensure that you haven't created any duplicate keys. If you have, it's easy to change them, both in the menu and in the accelerator table.

C H A P T E R 18

Creating a string table

A *string table* holds error messages, prompts, or any other text strings your application needs to display. You can store multiple string tables in your project file. Typically, you'll define a separate string table for each logical grouping of your program, as described on page 283.

Defining strings of text as a separate resource makes it easy to edit the text without changing your source code. For example, if you're translating a Windows application into a foreign language, putting most of your text in string tables simplifies the process. (You would still have to translate text in other resources, such as dialog boxes.)

Working with string tables involves four basic tasks:

1. Starting the String editor.
2. Creating and editing string tables.
3. Saving the string table.
4. Compiling the resource into the executable file and testing the string table.

Beginning on page 285 you'll find a short tutorial that takes you through the steps of creating and editing a string table.

Starting the String editor

How you start the String editor depends on whether you're creating a new string table or editing an existing one.

To create a new string table

To start the String editor to create a new string table,

1. Open the project to which you want to add the string table.
2. Choose Resource | New. Resource Workshop displays the New Resource dialog box.
3. Scroll the Resource Type list box and double-click STRINGTABLE.

Resource Workshop opens the String editor and places a reference to the new string table in your Project window.

To edit an existing string table

To start the String editor to edit an existing string table,

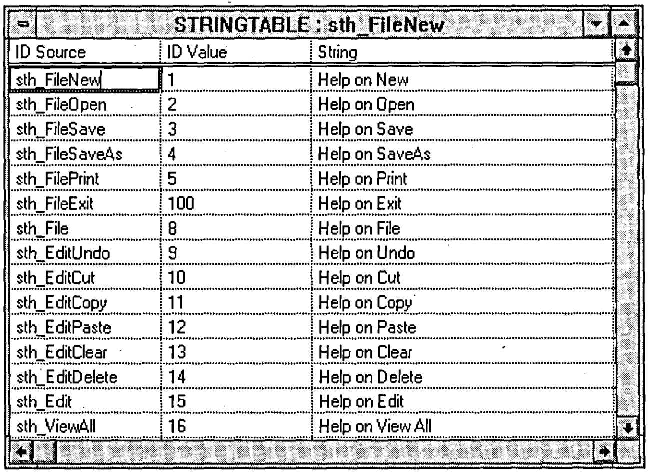
1. Open the project containing the string table you want to edit.
2. Find the string table in the Project window.
3. Double-click the string table entry or select it and choose Resource | Edit.

The string table you selected appears in the String editor.

Working with string tables

When you open the String editor, a string table appears. If you're creating a new string table, you'll see a single entry with the generic text "String." If you're editing an existing string table, you'll see string entries that look something like this:

Figure 18.1 String editor with string table entries



STRINGTABLE : sth_FileNew

ID Source	ID Value	String
sth_FileNew	1	Help on New
sth_FileOpen	2	Help on Open
sth_FileSave	3	Help on Save
sth_FileSaveAs	4	Help on SaveAs
sth_FilePrint	5	Help on Print
sth_FileExit	100	Help on Exit
sth_File	8	Help on File
sth_EditUndo	9	Help on Undo
sth_EditCut	10	Help on Cut
sth_EditCopy	11	Help on Copy
sth_EditPaste	12	Help on Paste
sth_EditClear	13	Help on Clear
sth_EditDelete	14	Help on Delete
sth_Edit	15	Help on Edit
sth_ViewAll	16	Help on View All

Each string table entry requires an ID Source, an ID Value, and the string itself.

- An ID Source contains an integer for the string. If you assign an identifier as the ID, it appears here. Otherwise, you'll see the integer ID.
- An ID Value always contains the integer ID for the string.
- A String is stored as a text string with a maximum length of 255 text characters.

You can right-click on an item in a string table to view a SpeedMenu, which lets you perform tasks quickly (you can also select the item and press *Alt+F10*). The next section describes how Windows handles string ID values and suggests a way to make the most efficient use of memory.

Windows and strings

Each string in a string table must have a unique integer ID. Windows groups strings into segments that contain 16 strings each. Strings with IDs from 0 to 15 make up the first segment, strings 16 to 31 are in the second segment, and so on. When you compile your resources, the strings are added to the executable file in segments that are loaded into memory at run time.

Windows loads an entire string segment into memory each time your application requires a particular string. If you plan how you assign string IDs carefully, you can reduce the amount of memory your application requires.

Suppose you define 32 strings for your application. If you assign IDs 0 through 31 to these strings, your executable file will have two 16-string segments. Each time your application needs a string and loads a segment, it is probably loading several strings that are not needed.

To make better use of memory, group your strings logically. For example, one part of your application might require five strings, and a second part might require nine strings. If you number the first group 0 to 4 and the second group 16 to 24, you create two segments, one with eleven unused IDs and the other with seven. Each unused ID only takes up one byte of memory, compared to the considerably greater amount of memory consumed by strings that aren't used. By organizing your strings this way, you allow Windows to load related strings without loading strings that aren't needed.

For more information about using identifiers and identifier files, see page 210.

When you specify a unique string ID, you can use an integer or an alphanumeric identifier (a **#define** in C or C++, or a constant declaration in Pascal) that stands for an integer. If you choose to use alphanumeric identifiers to make the string IDs easier to remember, store your identifiers in an identifier file (a header file for C and C++ or a unit or include file for Pascal). Be sure one of these files exists before you try to add identifiers to it from inside the String editor.

Entering a new string

To enter a new string in a string table,

- If the table is a new one, start entering information for the string as described in steps 3, 4, and 5, which follow.
- If you're adding a string to a table, start with step 1.

1. Select the string above the line where you want to add the new string.
2. Press *Ins* or choose Stringtable | New Item.
3. You can accept the number the String editor puts in this field, or you can type an integer or identifier ID Source.

   To restore the ID Source to its original setting, press *Esc* before pressing *Tab* or *Enter*.

   If you type an integer, the String editor automatically enters it in the ID Value field.

   If you type an identifier, Resource Workshop checks to see if it already exists. If it does, the String editor inserts the identifier's integer value in the ID Value field when you tab to the String text field. If the identifier doesn't exist, the String editor displays the "Create a new identifier?" dialog box.

   For more information about using identifiers and identifier files, see page 210.
4. Press *Tab* or click in the box under String and type the text string.

   To restore the String field to its original setting, press *Esc* before pressing *Tab* or *Enter*.

   Each string can be a maximum of 255 characters long and can contain any C-type escape sequences, including the following: \n (newline), \t (tab), \r (carriage return), \\ (backslash),\" (double quote).

   When the Resource Workshop compiler encounters a C-type escape sequence in a string entry, it produces the corresponding ASCII hexadecimal value in the object code, and it's up to your program to interpret the value correctly. For example, when the compiler parses *\b\040\x7F*, it produces the hex sequence *07207F*. Your code might interpret this sequence as the ASCII characters BEL, SPC, and DEL, or it could assign another meaning to these hex values.
5. Press *Enter* (to accept the new value) or *Ins* (to accept the value and insert a new one).

Editing existing strings

The String editor makes it easy to change individual strings. To select a string with your mouse, click the string you want to edit. Using the keyboard, press *Tab*, ↑, or ↓ to move through the table. Place your cursor on the string you want to edit.

Changing a string

You can erase the ID Source and String values for any string and then type new values. You can't directly change what's displayed in the ID Value field, but the String editor updates it depending on what you type in the ID Source field.

Editing the resource script of a string table

You can use the internal text editor to edit the resource script of a string table. To do so, select the string table in the Project window and choose Resource | Edit As Text.

The resource script text will appear, ready for you to edit.

Changing the string

To edit a string,

1. Find the string you want to edit and make the necessary changes to the string. Change only the text that appears between the quotation marks.
2. To compile what you just entered and stay in the String editor, choose Compile | Compile Now.

   Note that the Compile menu is available only when you are in the text editor.
3. If you want to exit the String editor, choose the Close command from the text editor window's Control-menu box.

   Resource Workshop asks if you want to compile. When you click Yes, Resource Workshop compiles the menu and exits to the Project window.

If there's a syntax error, Resource Workshop puts you back in the text editor so you can correct the error.

Creating a sample string table

The example that follows creates a few strings that Resource Workshop uses to describe menu options.

Without Resource Workshop, you'd use the following resource script to create these strings:

These strings would appear in the Resource Workshop Status line.

```
STRINGTABLE
BEGIN
  MENU_FILE,   "Create, open, or close files"
  MI_FILENEW,  "Create a new project, resource, or file"
  MI_FILEOPEN, "Open a resource file"
  MI_FILESAVE, "Save this resource file"
END
```

The uppercase alphanumeric string preceding each string is a unique identifier for the string. As with all Windows resources, each string requires an integer ID. Without Resource Workshop, you'd have to separately define the associated integer IDs for all these identifiers in a header file (for a C program) or in an include file or unit (for a Pascal program).

Here's how to create these sample strings with the Resource Workshop String editor:

1. Make sure you've already opened a project. If you've been doing the examples using MYPROJ.RC, you can open that project.

For more information on identifiers and identifier files, see page 210.

2. If you don't already have an identifier file (a header file for C **#define**s, or a unit or include file for Pascal constants) set up for the project, set one up now and call it MYPROJ.H or MYPROJ.PAS.
3. Choose Resource | New and double-click STRINGTABLE to start the String editor.
4. Backspace over the number in the text box under ID Source and type the identifier for the string. For the first string, it's `MENU_FILE`.
5. Press *Tab* to move to String.

   Before the String editor lets you move to the String field, it checks for an integer ID for the current string. First, it checks what you typed under ID Source. If you had entered an integer, the String editor would have put the same integer under ID Value and let you move to the String field.

For more information about identifiers, see page 211.

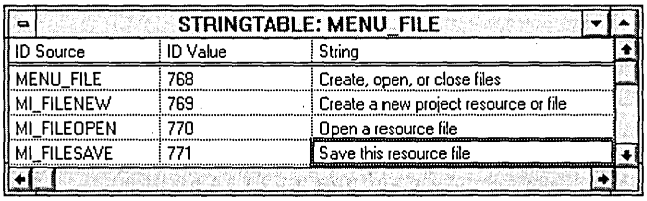
   Since you entered an alphanumeric identifier (MENU_FILE in this case), the String editor checks for a C **#define** or a Pascal constant declaration that points to an integer identifier. Because there isn't one, you're asked if you want to create a new identifier.
6. Click Yes to bring up the New Identifier dialog box.
7. Scroll down the File list until you find MYPROJ.H (or MYPROJ.PAS), then double-click it to select it as the identifier file to which the new identifier will be written.
8. Enter a unique integer ID in the Value text box. For the first identifier, type `768`.
9. Press *Return* or click OK to accept the new identifier and put it in MYPROJ.H or MYPROJ.PAS.
10. Now that you're done with the identifier, you're back in the String field. Type the text of the string. For the first string, it's `Create, open, or close files.`
11. To define the next string, press *Insert* or choose Stringtable | New Stringtable Item.

Repeat steps 4 through 11 to define the three other strings shown at the start of this section: MI_FILENEW, MI_FILEOPEN, and MI_FILESAVE (see Figure 18.2).

You'll notice that for each new string, the String editor increments the integer ID by 1 over the last integer ID. You needn't pick this number for the integer ID; the String editor simply puts it there for your convenience.

When you've finished creating the four strings, your string table should look like this:

Figure 18.2
String editor with four strings defined



STRINGTABLE: MENU_FILE

ID Source	ID Value	String
MENU_FILE	768	Create, open, or close files
MI_FILENEW	769	Create a new project resource or file
MI_FILEOPEN	770	Open a resource file
MI_FILESAVE	771	Save this resource file

As a last step, close the string table by choosing Close from the String editor window's Control menu. The String editor gives the new table the name of the first identifier in the table. If the first ID Source entry had been a number, that number would have become the name of the string table. A string table's name can be changed only by changing the first ID Source entry in the table.

This naming convention makes sense if you recall that strings are loaded in segments of 16 strings each, and the integer ID of a string indicates where the string occurs in a segment. The integer ID of the first string in the table indicates where the table starts in a segment.

C H A P T E R 19

Using the Bitmap editor

The Resource Workshop Bitmap editor is the tool you use to create or edit any bitmapped resource, including the following standard bitmapped resources:

- Icons
- Cursors
- Bitmaps
- Fonts

Features specific to the different resource types are described in Chapters 20 through 22.

This chapter describes the Bitmap editor tools and features whose functionality is generally the same for all types of bitmapped resources. The chapters on the individual resources describe the Bitmap editor tools and features that are unique to each resource type.

Starting the Bitmap editor

Resource Workshop automatically starts the Bitmap editor when you create a new bitmapped resource or edit an existing one. The specific steps for starting the Bitmap editor are given in the chapters for the individual resources.

If you create a new bitmap and there is a bitmap in the Windows clipboard, the dimensions of the clipboard bitmap appear in the New Bitmap attributes dialog box. If there is no clipboard bitmap, the default size is 64 pixels width and height.

Pixels, foreground and background colors

See page 299 for more information on foreground and background colors.

Bitmapped images drawn with the Bitmap editor are created on a grid of roughly square "dots" called *pixels*. You create the image by setting each pixel to what is referred to as a *foreground* or *background* color. The pixels assemble like a mosaic to form the bitmapped image.

Because the pixels exist side-by-side in a single plane, there is strictly speaking no distinction between foreground and background. In simplest terms, the foreground color is the color you select and draw with the *left* mouse button; the background color is the color you select and draw with the *right* mouse button.

You can use a variety of selected foreground or background colors for the features of your drawing (lines, boxes, shading, and so on), as well as for the image's "background" (which is, after all, an illusion, given the two-dimensional nature of the image). The ability to assign colors to both mouse buttons means you can have two colors at your disposal at any time.

➡ There is one important difference between the foreground and background color. When you delete or move a block of pixels in your image, the currently selected background color replaces the color in the pixels no longer occupied by the block.

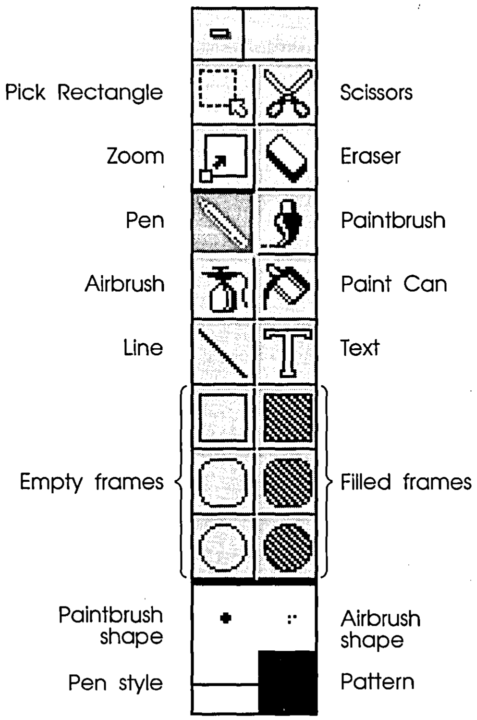
If you're using the Eraser tool, the buttons are reversed: the left button produces the background color and the right button produces the foreground color.

The current foreground color is indicated by *FG* on the Colors palette, and the current background color by *BG*. ("Current" is stressed here, because you can change the foreground or background color whenever you want.) If you select the same color as both foreground and background color, the Colors palette square reads *FB*.

Using the Tools palette

When you open a resource in the Bitmap editor, the Tools palette is in the upper right of the edit window. You use the Tools palette to choose the Bitmap editor tool you want to work with. At the bottom of the Tools palette are four style selections for brush shapes, fill patterns, and line widths.

Figure 19.1
Bitmap editor Tools palette



The Tools palette is similar to a window: you can move it, close it, and open it.

Most paint tools let you paint with either the current foreground color or the current background color.

If you see *FB* in the Colors palette, the same color is selected as the current foreground and background color.

- Use the left mouse button to paint using the current foreground color (*FG* on the Colors palette).
- Use the right mouse button to paint using the current background color (*BG* on the Colors palette).

To select a tool from the palette, click the tool you want. The sections that follow describe each tool.

Pick Rectangle tool

Choose Edit|Select All to select the entire image.

The Pick Rectangle tool selects a rectangular area of your image for copying, moving, or deleting. To select an area, put the tip of the pointer at one corner of the rectangle and drag to the diagonally opposite corner. When the flashing outline includes the area you want, release the mouse button. To deselect the area, click either mouse button outside the flashing outline or press *Enter* or *Esc.*

When you select an area, you can use the commands in the Edit menu to cut, copy, delete, duplicate, or paste the selected area, or you can use the mouse to move or duplicate the area.

Scissors tool

The Scissors tool performs basically the same function as the Pick Rectangle tool: selecting an area of an image. However, with the Scissors you can select and move areas of any shape, not just rectangles.

To select an area, drag with the Scissors until the outline includes the area you want, then release the mouse button. The area you've selected is indicated by a pulsating pattern. You can cut, copy, delete, duplicate, or move the selected area just as you can with the Pick Rectangle tool.

Zoom tool

The Bitmap editor uses the center of the image as a reference when zooming the entire image.

You can use the Zoom tool to zoom in or out on the entire image or to zoom in on a selected area.

To zoom in on the entire image, double-click on the Zoom icon in the Tools palette. Resource Workshop zooms to the next higher magnification: 400%, 800%, or 1600%. You can also choose View | Zoom In to perform the same function on the currently selected window.

When you zoom in on the image, use the Hand tool (page 297) or the scroll bars to move the zoomed image around.

To zoom out on the entire image, hold down the *Shift* key and then double-click the Zoom icon. To zoom a portion of the image, select the area you want to magnify by dragging a rectangle with the Zoom tool, then release the mouse button. Resource Workshop zooms out to the next lower magnification: 800%, 400%, or 100%. You can also choose View | Zoom Out to perform the same function.

Table 19.1 Zoom commands

View command	Accelerator key	Mouse action on Zoom icon
Zoom in	Ctrl+Z	Double-click
Zoom out	Ctrl+O	*Shift*+double-click
Actual size	Ctrl+A	None

To help you control images on a pixel-by-pixel basis, you can display a grid on any zoomed image by choosing Options | Editor Options and clicking the Grid On Zoomed Windows option. Each square of the grid highlights a single pixel.

➡ When you're working with two window panes (see page 297), zooming affects only the current (active) window.

Eraser tool

If you see *FB* in the Colors palette, the same color is selected as the current foreground and background color.

The Eraser tool can be used to erase the entire image, or it can be used as a drawing tool. Note that, for the Eraser, the color assignments to the mouse buttons are the reverse of the other drawing tools' assignments.

- If you double-click the Eraser in the Tools palette, the entire image is replaced with the current background color (*BG* on the Colors palette).
- If you drag with the left mouse button, the Eraser draws a line one pixel wide using the current background color.
- If you drag with the right mouse button, the Eraser draws a line one pixel wide using the current foreground color (*FG* on the Colors palette).

Before you use the Eraser, you can check the current colors in the Colors palette.

Pen tool

The Pen tool paints free-form lines using the current pen style shown in the Tools palette (see Figure 19.1). To sketch with the Pen tool, press a mouse button and drag the cursor across your image. When you've finished sketching, release the mouse button. (To paint absolutely straight lines, use the Line tool instead of the Pen.)

Paintbrush tool

The Paintbrush tool paints free-form patterns using the current brush style and the current pattern shown in the Tools palette (see Figure 19.1). To paint, drag the Paintbrush across your image. When you've finished painting, release the mouse button.

When you select the Paintbrush, the cursor takes the current brush shape. The area painted by the Paintbrush is always proportionally the same relative to the size of the image frame. In other words, if the brush is half the width of the image and you zoom in on the image, the brush is still half the width of the now zoomed image.

Before you use the Paintbrush, you can specify the brush shape, pattern, and colors.

Airbrush tool

The Airbrush tool paints free-form patterns using the current airbrush style and the current pattern shown in the Tools palette (see Figure 19.1). To use the Airbrush, you can use either of these techniques:

- You can drag it across the image. The Airbrush differs from the Paintbrush in that if you drag it slowly, it paints a thick pattern, but if you drag it quickly, it paints a scattered, thinner pattern.
- You can click it repeatedly, as if you were pressing the nozzle of a spray can.

➡ When you select the Airbrush, the cursor takes the current brush shape. The area painted by the Airbrush is always proportionally the same relative to the size of the image frame. In other words, if the Airbrush is half the width of the image and you zoom in on the image, the Airbrush is still half the width of the now zoomed image.

Before you use the Airbrush, you can specify the brush shape, pattern, and colors.

Paint Can tool

The Paint Can tool fills an area of your image with a selected color. To use the Paint Can, place its cross hair in the portion of the image you want to fill, then click a mouse button. The Paint Can replaces the color under the cursor with the selected color and fills out around that point until it meets a different color.

For example, if the selected color is red and you click on a blue square, all the blue around that point will be replaced by red. The Paint Can will not replace any other colors. If the blue square is part of a rectangular blue area that is entirely surrounded by green, only the blue rectangle will be changed to red.

➡ If you click on an area that's not entirely surrounded by other colors, the color will leak out into other parts of the image that are the same color as the original area.

➡ Because of problems inherent to display drivers, flood-filling a bitmapped image with the Paint Can doesn't always work properly. To solve this problem, Resource Workshop provides an alternative to the standard flood-fill algorithm that's more reliable, but slower. To enable this algorithm for any of the bitmapped resource editors, add the following line to the [RWS_Icon] section of WORKSHOP.INI:

```
RWS_OwnFloodFill=1
```

The following example shows a sample of the edited RWS_Icon section:

```
[RWS_Icon]
RWS_OwnFloodFill=1
PercentLeft=69
ZoomLeft=8
ZoomRight=1
bVert=1
```

Line tool

For free-form lines, use the Pen.

The Line tool paints straight lines. Press the mouse button and drag the Line tool across your image. When you've finished drawing the line, release the mouse button.

To constrain the lines to 45-degree increments (horizontal, vertical, or diagonal), hold down *Shift* as you paint.

Before you use the Line tool, you can specify the line style and current colors.

Text tool

To add text to your image, choose the Text tool and click where you want the text to begin. A flashing cursor indicates that you can begin typing text.

To specify how and where the text is displayed, you can

- Use Text | Font to specify the typeface, size, and style of the text.
- Use the Text | Align commands to specify how the text is aligned.

For more information about using the Text menu commands, see "Adding text to a bitmap" on page 303.

You don't need to choose any Text menu commands before you enter the text; you can also choose them immediately after you enter the text (and before you click the mouse again). For example, if you notice that the text you're typing is too large to fit in your image, you can stop typing and choose the Font command to decrease the size of the text.

Text is always displayed in the current foreground color. Before you type text, you can specify the foreground color by clicking the left mouse button on the color you want in the Colors palette. Just as with the typeface and size, you can change the current text color if you do so immediately after you enter text.

Painting empty frames

There are three tools you can use to paint empty frames using the current line color and style: the Rectangle, the Rounded Rectangle, and the Ellipse.

To paint an empty frame, select the tool you want and drag a frame in the image. Place the cursor crosshair at one corner of the frame and drag to the opposite corner. Release the mouse button when the frame is the way you want. Press *Shift* to paint a true square or circle.

Before you paint a frame, you can specify the frame width and the color.

Painting filled frames

There are three tools you can use to paint filled frames in your image: the Filled Rectangle, the Filled Rounded Rectangle, and the Filled Ellipse.

To paint a filled frame, select the tool you want and drag a frame in the image. Place the cursor crosshair at one corner of the frame and drag to the opposite corner. Release the mouse button when the frame is the way you want.

These tools use the current line style for the outline. Specify a null pen width if you don't want Resource Workshop to paint an outline around the fill pattern.

The current pattern is displayed in the lower right corner of the Tools Palette. If you drag with the left button, the outline and pattern take the current foreground color (FG), and the pattern fill takes the current background color (BG). If you drag with the right button, the colors are reversed.

The Set Pattern dialog box also includes two solid patterns, one black and one white.

- If you select the solid black pattern, the left button produces a solid fill of the foreground color and an outline of the background color. (As usual, the right button produces the reverse.)
- If you select the solid white pattern, the left button produces a solid fill of the background color and an outline of the foreground color. (Again, the right button produces the reverse.)

Before you paint a filled frame, you can specify the line style, color, and pattern.

Hand tool

You can also use the scroll bars to move the image around.

Sometimes when you display a zoomed image, not all of it fits in the display. You can use the Hand tool to move the image around so you can see other parts of it. Unlike other tools, the Hand tool isn't included in the Tools palette, but you can temporarily change any tool (except the Text tool) into a hand by holding down *Ctrl*. Using the hand, you can take hold of the image and drag it in the direction you want it to move.

Style selections

At the bottom of the Tools palette is a box that shows (clockwise from the top left) the brush shape, the airbrush shape, the current pattern, and the line style.

You can click any style you want to change, or you can use menu commands to choose styles. (For more information about choosing styles, see "Choosing brush shapes" on page 304, "Choosing paint patterns" on page 305, and "Choosing a line style" on page 306.)

Using the two window panes

In the Bitmap editor, you can look at two different views of the image you're creating or editing. You can split the window vertically or horizontally to show the two views side-by-side or one view above the other. You can also choose how to zoom each view.

To split the window, choose View | Split Horizontal or View | Split Vertical.

When the window is split, one of the panes is *active*. The active pane is the one in which you're working. To make a pane active, click the mouse in that pane.

To see more of one view than the other, move the cursor to the line that splits the images (the separator bar) and, when the cursor becomes a double arrow, drag the separator bar. For example, with the windows split vertically, you can drag the separator bar to the right to see more of a zoomed image.

To remove the split window entirely and return to a single view, drag the separator bar all the way to the left or right (for a vertical split) or to the top or bottom (for a horizontal split).

Reading the status line

The status line at the bottom of the Bitmap editor window is divided into two parts: the right side provides current paint tool information, and the left side displays information about the Bitmap editor's menu commands.

As you click a menu or use accelerator keys to choose a menu command, the left side of the Bitmap editor status line displays more information about the currently highlighted command.

The right side of the status line tells you which paint tool you're using and where it is on the screen. You might also see color information, depending on the tool you're using.

The tool status message you see depends on which tool you've selected and where the tool is on your screen. Here are two examples of messages with accompanying explanations.

Line x: 18 y: 32
This message indicates that you've selected the Line tool, which is located at the pixel coordinates 18,32. Pixel coordinates are counted from the upper left corner of the image. As you move towards the right side of the image, the X value increases; as you move towards the bottom of the image, the Y value increases.

You can look at palette index numbers and RGB values by double-clicking a color in the Colors palette.

Brush x: 20 y: 37 R: 128 G: 0 B: 0 Palette Index: 1
This message indicates that you've selected the Paintbrush, which is located at the pixel coordinates 20,37. The R, G, and B values indicate the red, green, and blue color values of the color at those coordinates. The color of the image at 20,37 is currently the color *1* in the Colors palette index, which has a value of 128 red, 0 (zero) green, and 0 blue.

See page 301 for more information on RGB values and palette indexes.

You can also see the palette index and RGB settings for the color at the cursor if you select the Pen, the Airbrush, or the Paint Can.

Working with colors

To choose the colors you want as you edit a resource in the Bitmap editor, use the Colors palette. You can work with the Colors palette even if your image is black and white, and you can hide or display the Colors palette at any time.

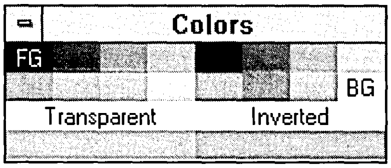


Figure 19.2 16-color Colors palette

You can use the Colors palette to

- Choose a foreground color
- Choose a background color
- Choose transparent and inverted areas (icon and cursor resources only)

Choosing the number of colors for a resource

When you create a new bitmap or icon, Resource Workshop displays the New Bitmap Attributes dialog box that lets you choose how many colors you want to include in your resource.

While you're editing a bitmap or icon, you can change the number of colors in the image using the Size and Attributes command. This command is located in the Bitmap menu or the Icon menu, depending on the type of resource you're currently editing.

Some of the higher resolution Windows 3.1 drivers require more than 512K of memory. For bitmap and icon resources, you can include up to 256 colors in your resource. The number of colors you can use (and see in the Colors palette) depends on the type of display driver you're using with Windows.

Using foreground and background colors

If you see *FB* in the Colors palette, the same color is selected as the current foreground and background color.

To use a foreground color,

1. Click the *left* mouse button on the color you want in the Colors palette. The letters *FG* appear on that color.
2. Select a tool that draws or paints, and click or drag with the left mouse button to draw or paint with the foreground color.

To use a background color,

1. Click the *right* mouse button on the color you want in the Colors palette. The letters *BG* appear on that color.
2. Select a tool that draws or paints, and click or drag with the right mouse button to draw or paint with the background color.

➡ The Eraser operates in the opposite fashion from the drawing tools. Dragging it with the left mouse button produces the background color, and dragging it with the right mouse button produces the foreground color.

Transparent and inverted color areas

The idea of a *transparent* or *inverted* color area is unique to icon and cursor resources.

- A transparent area "drops out" at run time, allowing the desktop color behind the icon or cursor to show through. This is especially useful in cursors, where you will typically not use the entire image area for the cursor itself.
- Using the inverted color in your icon or cursor causes that area to "reverse" the desktop color at run time.

  For example, if you create a cursor's cross hair from an inverted color, at run time the cross hair will appear in the reverse color from the desktop area underneath it. If you place the cursor over a black desktop area, the cross hair will be white; if you move it to a red area, it will be a teal blue.

Using transparent and inverted color areas is described in detail in Chapter 20, "Creating icons," and Chapter 21, "Creating cursors."

Setting transparent and inverted colors

The designated transparent and inverted colors do not appear in your icon or cursor at run time. Instead, they are replaced by the desktop color underneath or its inverse. The colors that you set as Transparent and Inverted should be colors that you won't use in your icon or cursor.

The default transparent color is the current desktop color set in the Windows Control Panel's color palette. If the desktop uses a dithered color, the default transparent color is the nearest solid color that Resource Workshop can provide. (If you have a 256-color device, this restriction does not apply; the default transparent color will always match the desktop color.)

You can change the transparent color to something other than the desktop color, but it will always revert to the desktop color each time you start Resource Workshop. Nevertheless, regions that you designate as transparent—using the default color or a color you assign—remain transparent and take on the current transparent color. Note, however, that you can change colors so that you have transparent and nontransparent regions that use the same color.

To change the colors the Bitmap editor displays for transparent and inverted areas, do either of the following:

- In the Colors palette, double-click the bar under either Transparent or Inverted.

- Select either Transparent or Inverted as the foreground or the background color. Then choose either Icon | Edit Foreground Color or Icon | Edit Background Color.

Resource Workshop displays the Set Transparent Color dialog box.

The dialog box is always Set Transparent Color, regardless of whether you started from the Transparent color bar or the Inverted color bar.

This dialog box works like the Edit Colors dialog box, except that it changes both the transparent and the inverted color at the same time.

Note the three color boxes at the top of the dialog box. As you adjust the RGB (Red, Green, Blue) values, the actual color created by those RGB percentages is displayed in the Requested box. The Granted box displays the closest available color (for devices that support 256 colors or more, it will be the same as the Requested color), and the Inverse box automatically shows its inverse.

The Granted color is assigned to the Transparent color bar, and the Inverse color is assigned to the Inverted color bar.

Hiding and showing the Colors palette

To hide the Colors palette, close it by double-clicking the system menu icon in the upper left corner of the palette.

You can also hide the Colors palette by choosing the Hide Palette command. The name of the menu where you'll find this command depends on the type of resource you're currently editing. For example, if you're editing an icon, the Hide Palette command is in the Icon menu. If you're editing a cursor, the menu is in the same position on the menu bar, but it is called the Cursor menu.

After you've hidden the palette, the name of the command changes to Show Palette so you can redisplay the palette.

Customizing colors

You can't edit colors in cursors or fonts.

If you're editing a color bitmap or icon, you can modify the Colors palette to include any colors supported by your display driver. It won't make sense to do this if the Colors palette already includes all the colors supported by your computer. But if your display driver is capable of displaying 256 colors and you're working with a 16-color image, you can include any of the 256 colors in the 16-Colors palette.

The following sections describe how to edit any of the colors in the Colors palette, including transparent and inverted.

Editing colors in the Colors palette

To select a color for editing, do either of the following:

- Double-click it in the Colors palette.
- Select it as the foreground or the background color, then choose Edit Foreground Color or Edit Background Color from the Icon or Bitmap menu.

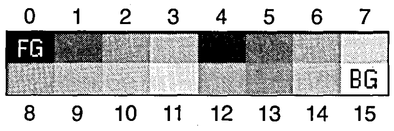
Resource Workshop brings up the Edit Color dialog box.

Palette index

To help you identify where the selected color belongs in the Colors palette, the Edit Color dialog box includes a palette index value. Each box in the Colors palette is assigned an index number, starting with zero at the top left and counting from left to right across each row. (Index values aren't assigned to the Transparent and Inverted bars in the Colors palette.)

For example, in the default 16-color palette, the colors are numbered as shown in the following figure:

Figure 19.3
16-color palette index



When the selected tool is the Pen, Paintbrush, Airbrush, or Paint Can, the right side of the status line displays the Palette index and RGB settings for the color under the cursor's position in the edit window.

Editing a color

On a device capable of displaying 256 colors or more, the Requested/Granted distinction doesn't apply. You see true colors, not dithered approximations.

To edit a color, you can change its RGB values either by typing new values in the left column or using the slide bars on the right side of the dialog box. Resource Workshop displays the closest matching color for the new RGB values in the Requested color box. (For a 16-color device, you might see a dithered color appear in this box.) In the Requested color box, Resource Workshop displays the closest available color as determined by the limits of the current device. You only see this right box change color if the closest match is different from the current color.

When you're finished changing colors, click OK or press *Enter* to put the new color in the Colors palette.

➡ Before you leave the Bitmap editor you must turn off the Save with Default Device Colors option to allow Resource Workshop to save your customized palette. Choose Options | Editor Options and make sure there isn't a check mark next to Save with Default Device Colors.

The Default button retrieves the color from the default palette (a Windows stock object) that has the same index as the Palette index (shown at the top of the dialog box).

The System button retrieves the color from the system palette that has the same index as the Palette index (shown at the top of the dialog box). This button is disabled for standard VGA displays (which do not support logical palettes) but is enabled for devices that support 256 colors or more.

Adding text to a bitmap

You can add text to any bitmap. For example, you can add text to an icon to represent the program name.

To add text to a resource, click the Text tool (shown at the left) and then click in the image where you want the text to start. Resource Workshop displays a flashing text cursor. Then type the text you want.

To set default alignment, font, size, and style (bold, italic, or underlined), choose Text | Align or Text | Font before you type. To change the alignment, font, size, or style of your text, choose Text | Align or Text | Font immediately, before you click the mouse again.

You can only enter text or change its color with the left mouse button.

In addition, immediately after you type the text (and before you click the mouse again), you can change its color by selecting a new color from the Colors palette.

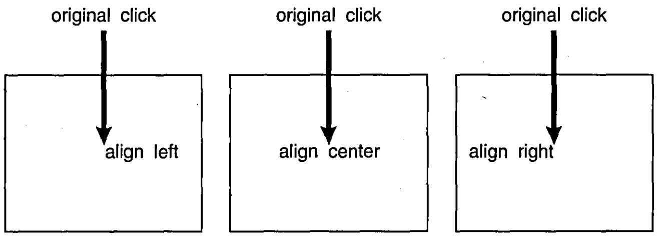
If you click another tool or click in another area of the image after entering text, you can't change anything in the text you've just entered. At that point, the text becomes just another part of the bitmap as though you had painted it there.

Aligning text

To align text, use the Text | Align commands.

The Align commands control where the text is displayed in relation to where you clicked the Text tool (the insertion point). You can use an Align command either before you type text or immediately after you finish typing.

Figure 19.4
Aligning text



You can't change the alignment of text you've typed once you click to make another selection.

Choosing fonts, size, and text style

To choose how text is displayed, use the Text | Font command either before you type text or immediately after you finish typing.

You can't change the font of text you've typed once you click to make another selection.

After you choose the Font command, Resource Workshop displays the Select Font dialog box.

You can choose the typeface, size, and style of text that you want. For example, you might want Arial as the typeface, 12 point as the size, and bold italic as the style.

The alphabet characters displayed at the bottom of the Select Font dialog box change to show you the current typeface, style, and size you've selected.

Choosing brush shapes

You can paint images in a resource using the Paintbrush or the Airbrush. Resource Workshop lets you choose the shape of the Paintbrush or Airbrush.

The current brush shape and airbrush shape are always displayed at the bottom of the Tools palette (see page 297).

Choose a brush shape or an airbrush shape in one of the following ways:

- For a brush shape, use Options | Brush Shape.
- For an airbrush shape, use Options | Airbrush shape.
- Click the brush or airbrush shape style selection.

You'll see a dialog box that shows you all the possible brush shapes you can choose. At the top of the dialog box, Resource Workshop shows you the shape that's currently selected.

The next time you use the Paintbrush or Airbrush, Resource Workshop uses the shape you specify. This shape stays the same until you specify a new one.

Choosing paint patterns

You can choose a pattern to paint on your image, using any of the following tools:

- The Paintbrush
- The Airbrush
- The filled rectangle
- The filled rounded rectangle
- The filled ellipse

Most of the patterns consist of black lines or dots on a white "field." If you draw with the left mouse button, the lines or dots take the current foreground color, and the field takes the current background color. If you draw with the right mouse button, the effect is reversed: the lines or dots take the current background color, and the field takes the current foreground color.

The current pattern is always displayed in the lower right corner of the Tools palette.

You can choose a pattern in one of the following ways:

- Choose Options | Pattern.
- Click the style selection for the pattern.

You'll see a dialog box that shows you all the possible patterns you can choose. At the top of the dialog box, Resource Workshop shows you the pattern that's currently selected. To select a new pattern, click one of the patterns in the dialog box and click OK.

The next time you use a tool capable of filling with a pattern, Resource Workshop uses the pattern you specify. This pattern stays the same until you specify a new one.

Choosing a line style

You can control the line style produced by any of the following tools:

- The Pen
- The Line tool
- The empty rectangle
- The empty rounded rectangle
- The empty ellipse

You can choose a line style in one of the following ways:

- Choose Options | Pen Style.
- Click the line style selection in the Tools palette.

You'll see a dialog box that shows you all the possible styles you can choose. At the top of the dialog box, Resource Workshop shows you the style that's currently selected.

Note the null choice for a pen width. You can use null when you want to paint a filled rectangle, rounded rectangle, or ellipse without a border.

The next time you use a tool capable of painting a line, Resource Workshop uses the width you specified. The pen width you choose stays the same until you specify a new one.

Aligning a selected area

You can align a selected area of an image with the top, bottom, sides, or center of the current edit window. Aligning the selected area moves it to the location you specify, just as if you had selected the area and moved it with the mouse. To align an area you've selected, choose Options | Align and choose the options you want from the Align Selection dialog box.

Choose the options under Vertical to align the selected area along the vertical axis of the edit window and the options under Horizontal to align it along the horizontal axis.

For example, if you choose Top and Left Side and click OK, the selected area moves to the top left corner of the window. If you choose Vertical | Center and Right Side and click OK, the selected area is centered between the top and bottom of the window and is moved as far to the right as possible.

The Bitmap editor uses the current background color to paint the area where the selected area was located, just as it would have done had you moved the selected area with the mouse.

Resizing a selected area

You can resize or move an area of an image you've selected with the Pick Rectangle tool by choosing Options | Size. If you resize the area, the Bitmap editor stretches or compresses the image inside the selection area accordingly.

When you choose Options | Size, the Bitmap editor displays the Stretch Selection dialog box.

The Left and Top values in this dialog box are the pixel coordinates of the top left corner of the currently selected area, and the other values are its current width and height in pixels. (When the dialog box is first displayed, the same values appear in both the Old Position/Size and New Position/Size columns.)

To move or resize the area (or both), pick pixel values that fall within the current frame and enter them in the boxes under New Position/Size. Press *Tab* to move from field to field.

For example, to move a rectangular area to the top left corner and make it 30 pixels wide and 5 pixels high, do the following:

1. Using the Pick Rectangle tool, select the area you want to move and resize.
2. Enter 0 as the Left value and press *Tab.*
3. Enter 0 as the Top value and press *Tab.*
4. Enter 30 as the width and press *Tab.*
5. Enter 5 as the height.
6. Click OK or press *Enter.*

The box is now positioned in the top left corner of the frame, with the new width and height. In addition, the Bitmap editor fills any open area left behind with the current background color, just as if you had moved the selection rectangle with the mouse.

Setting global Bitmap editor options

To set the global Bitmap editor options, choose Options | Editor Options to display the Set Bitmap editor Options dialog box.

Draw on both images

When you have two views of the same image displayed in the Bitmap editor, you can choose to have Resource Workshop update each image as you draw.

If you don't choose this option, Resource Workshop updates the other image only after you finish drawing an element. For example, as you drag a brush across the image, Resource Workshop shows the line only on the image on which you're actually drawing. However, once you release the mouse button, Resource Workshop then updates the other image.

On zoomed windows, you can display a grid that shows you how the image is painted on a pixel-by-pixel basis.

Save with default device colors

Uncheck this option if you want to save a custom Colors palette you've created. When this option is checked, any colors you've customized in the Colors palette will revert to the default color when you close the Bitmap editor.

You can save customized Colors palettes only if your display driver is capable of displaying 256 colors and supports logical palettes. See page 301 for information about customizing the Colors palette.

C H A P T E R 20

Creating icons

Icons are small bitmapped images, 64×64, 32 ×32, or 32×16 pixels in size. Windows programs typically use special icons to represent minimized windows.

To design your icons, use the Resource Workshop Bitmap editor.

The Bitmap editor includes various paint tools and an easy-to-use Colors palette for selecting colors. It also lets you zoom your image and shows you multiple views of the icon you're creating. Chapter 19 explains how to use the Bitmap editor.

Working with icons involves four basic steps:

1. Starting the Bitmap editor.

   If you are creating a new icon resource, the Bitmap editor presents you with an empty icon grid to work on. If you are modifying an existing icon, it appears in the Bitmap editor.
2. Creating or modifying the icon.
3. Testing the icon.
4. Saving the icon.

Creating a new icon

You can add a new icon to a project file, or you can create a standalone icon file with the extension .ICO.

Adding an icon to a project file

To add an icon to an .RC (or .DLG) file,

1. Open a new or existing project. (Chapter 14 describes how you open a project.)
2. Choose Resource | New. Resource Workshop displays the New Resource dialog box.
3. Scroll the Resource Type list box and select ICON.

The name of the current project file appears in the box under "Place resource in." You can scroll down this list to pick another file (if any is listed) or click OK to accept the current project file.

4. Resource Workshop next asks whether you want to create the icon in source form or in Microsoft-compatible binary format.

   Embedded and linked resources are discussed on page 203.

   Choose Source to create an icon that's embedded in the project file. Resource Workshop then displays the New Icon Image dialog box.

   If you choose Binary, you create a standalone .ICO file (see page 310).

5. The New Icon Image dialog box asks you to specify the size (in pixels) of the new icon and the number of colors you want supported. If there is a bitmap in the Windows clipboard, its size and color are used as the default.

   See page 299 for a description of color formats.

   Choose the image size and color format you want, then click OK. Resource Workshop puts the new icon name in the Project window and starts the Bitmap editor.

Creating a standalone icon file

You can create a standalone icon file by doing either of the following:

- Choosing Binary from the dialog box that asks if you want to create your icon in Source or Binary format.
- Choosing File | New Project and selecting .ICO from the New Project dialog box.

To link a standalone icon file to a project file, open the project file and then choose File | Add to Project. (See page 205.)

Binary format option

When you choose Binary format, Resource Workshop displays the New File Resource dialog box so you can name your icon file and specify the project in which it will be referenced. When the selected resource type is an icon, the File Name text box says *.ICO.

1. Enter a name for your .ICO file in the File Name text box. Make sure the name of your project appears in the list box labeled "A reference to the resource...". To change the directory the file is stored in, set the path by double-clicking the appropriate icons in the Directories list box. The path you have chosen is displayed in the Path text box.
2. When you've made your selections, click OK. Resource Workshop displays a dialog box saying the file you've named does not exist and asking if you want the file created. Click Yes.
3. When you exit the New File Resource dialog box, Resource Workshop displays the New Icon Image dialog box. When you select the icon's size and colors, Resource Workshop starts the Bitmap editor.

You can also pick a file name from the Files list box. In that case, however, you'll overwrite the existing file. As a precaution, Resource Workshop asks you to verify that you want the file overwritten.

Icon project file

If you choose File | New Project and select .ICO from the New Project dialog box, you automatically create a standalone icon file. Resource Workshop immediately displays the New Icon Image dialog box. After you specify size and colors, Resource Workshop starts the Bitmap editor.

Editing icons

The following sections describe several of the techniques you can use in editing your icons.

Viewing other resolutions

The standard EGA/VGA resolution is 32×32, but you can create icons in two other resolutions: 32×16 and 64×64. The View menu includes commands that let you view the icon in other display resolutions.

- If you're creating the icon in 32×32 resolution, the command View | CGA Resolution [32×16] lets you see how the icon would look on a CGA screen.
- If you're creating the icon in 32×16 (CGA) resolution, the command View | EGA/VGA Resolution [32×32] lets you see how the icon would look on an EGA or VGA screen.
- If you're creating the icon in 64×64 resolution, both commands are available. You can see how the icon would look on a CGA screen or on an EGA or VGA screen.

To get out of any of these view modes, choose View | Actual Size.

➡ All you change with these commands is the view of the icon, not the icon itself. It is still the same resolution.

Using transparent and inverted color areas

As mentioned on page 300, you can use transparent and inverted color areas in your icons. At run time, the transparent area "drops out" and lets the desktop color show through from underneath. Any parts of the icon painted with an inverted color take on the opposite color from the desktop underneath: white over black or teal over red, for example.

Inverted colors are generally more useful for cursors than for icons.

When you start a new icon, it is initially a grid made up entirely of the current transparent color. If you draw a single line of another color across the grid and then test the icon, you see only that line against the desktop color; you don't see the grid outline or the area that is filled with the transparent color.

Adding an image to an icon resource

In general, you create a new icon resource for each icon design (called simply an *icon*), and you don't put different icons in the same icon resource. However, it's likely you'll want to put different color formats of the same icon in one icon resource. These color variations on the same icon are called *images*. For example, if you want a 2-color and a 16-color version of the same design, you can store both versions in the same icon resource.

The reason the icon resource supports different color formats is that Windows picks a color format based on the ability of the display hardware to support the format. Windows picks a 2-color format for a monochrome display driver and a 16-color format for the standard Windows VGA driver.

Windows 3.*x* doesn't fully support the 256-color version of an icon, even if your display hardware supports it. Your program must supply its own support for 256 colors.

To add a new image to an existing icon resource,

1. Open a project.
2. Double-click the ICON entry in the Project window, or select the ICON entry and choose Resource | Edit. Resource Workshop displays the Icon window.

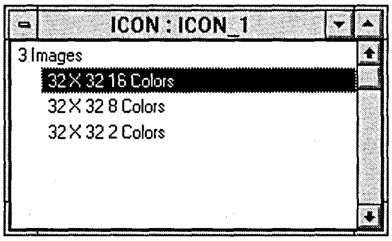


Figure 20.1
Icon window

3. To create a new version of an existing icon, choose Images | New Image. Resource Workshop displays the New Icon Image dialog box, from which you can choose the image size and color format.
4. Choose the same size as the existing image and a new color format, then click OK. Resource Workshop displays the new image entry in the Icon window.

5. Double-click the new icon, or select it and choose Images | Edit Image. You'll see the Bitmap editor.
6. Typically, what you do next is open one of the existing icon images and copy it into the new (still blank) image. You might also have to edit the image if the colors are translated in a way that changes the form of the icon.

Changing an icon's attributes

To change the attributes—resolution or color format—of the current icon image,

1. Choose Icon | Size and Attributes to display the Icon Image Attributes dialog box.

   Except for its having one additional push button (described next), the Icon Image Attributes dialog box is identical to the New Icon Image dialog box.
2. Select a resolution or color format and click OK.

➡ Changing the icon's attributes actually changes the icon, as contrasted with viewing the icon at other resolutions.

Displaying device information

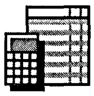
The additional push button in the Icon Image Attributes dialog box is called Device Info. When you click this button, Resource Workshop displays the Display Device Information dialog box, which give the following information about your display device:

- Number of bits per pixel
- Number of color planes
- Number of colors supported
- Whether it is a palette device

If the device supports logical color palettes, the Display Device Information dialog box lists the following information:

- The number of entries in the system palette
- The number of reserved entries in the system palette
- The color resolution of the device in bits per pixel

Creating a sample icon



This section describes how to create a sample icon, shown at left.

To create the new icon, you must first open a project, and then go through a series of steps to bring up the Bitmap editor with the new, blank image in it.

Chapter 14 describes how you open a project.

1. Choose File | New Project to create a new project or File | Open Project to open an existing project.
2. Choose Resource | New and tell Resource Workshop to create a new resource. When Resource Workshop asks what type of resource you want, choose ICON.
3. Choose Source to store the icon as a resource script in the .RC file.
4. Resource Workshop opens the New Icon Image dialog box. Check 32×32 and 16 Colors, then click OK.
5. Resource Workshop opens the Bitmap editor.

Drawing the calculator

To draw the calculator icon,

1. Choose Options | Editor Options, check Grid On Zoomed Windows to help you line up the calculator buttons, and click OK.

   The grid shows you the individual pixels when you zoom your icon in the Bitmap editor. It's easier to see what you're doing if you draw on a zoomed-in view of the icon.
2. Leave the icon displayed at its actual size in the right window. To zoom the image in the left window, double-click the Zoom icon in the Tools Palette until the image is zoomed as large as you can get it with all of it still visible onscreen.

   Each square of the grid on the zoomed view represents a single pixel.

   Now you're ready to start drawing.
3. Choose the color dark red for the calculator. When you click in the Colors palette, you see the letters *FG* (foreground) on the color dark red.
4. Click the filled rectangle in the Tools Palette. It's the square tool on the right under the Text tool (the large "T").
5. Draw the calculator in the bottom left part of the icon.

   Be sure to leave two pixels below the calculator to put in the shading.

   You can use the filled rectangle to draw all parts of the calculator—the face, the display area, and the buttons. Choose different colors for the calculator display and buttons by clicking in the Colors palette before drawing (try yellow for the keys and dark cyan for the screen). When you've finished, your calculator should look something like this:

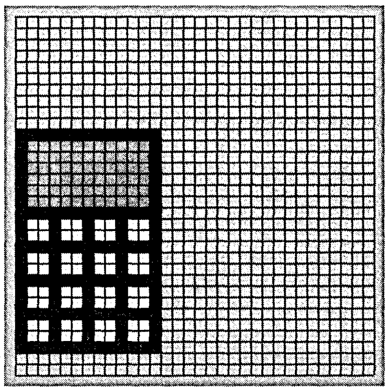


Figure 20.2
Calculator before adding drop shading

If you make a mistake, you can use the Undo feature (Edit | Undo or *Alt+Backspace*) to rectify it. You can also use the Pick Rectangle tool or the Scissors tool to outline an area and delete it (use the *Del* key or Edit | Delete) or the Eraser to erase it.

If you want to erase everything and start over again, just double-click the Eraser tool in the Tools Palette.

Adding a three-dimensional effect

To add a three-dimensional effect to the calculator,

1. Choose the color black for the shadow.
2. Use the Line tool to fill in a shaded line down the right side and across the bottom of the calculator. The line should be two pixels wide and should start two pixels below the top and end two pixels short of the left side.
3. You can also make the calculator window look three-dimensional by drawing a white line along the window's left edge and across its top.

Here's what the calculator should look like now:

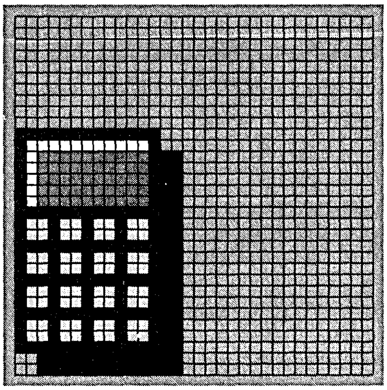


Figure 20.3 Calculator with shading

When you finish drawing the calculator, choose File | Save Project to save your new icon image. It's always a good idea to save early and often.

Drawing the ledger page

To draw the ledger page,

1. Choose the color black and then choose the Line tool.
2. Use the Line tool to draw a vertical black line starting on the fifth pixel to the left of the top, right corner of the calculator. Carry the line up to the top of the edit window, across to its upper right corner, down to within two pixels of the bottom of the Bitmap editor, and across to meet the black shading to the right of the calculator. You've just drawn the outline of the ledger page.
3. Choose the color white and select the Paint Can tool. Click the Paint Can on the ledger page to fill it with white.
4. Choose cyan (the lightest blue color) and the Line tool.
5. Click the Line style in the bottom right corner of the Tools Palette and, in the Current Pen Style dialog box, pick the two-pixel line width (the one to the right of Null).

Use the Edit|Undo command to correct any errors.

6. Starting two pixels up from the bottom of the ledger page and one pixel in from where you want the line to start (because it's two pixels wide), draw a series of two-pixel-high lines across the width of the visible part of the ledger page. Each line should be two pixels from the previous line. (The bottom three lines will be bounded on the left by the calculator shading.)
7. Choose light red and the Line tool.

8. Click on the Line style in the bottom right corner of the Tools Palette and, in the Current Pen Style dialog box, pick the single-pixel line width (the one directly above Null).
9. Starting three pixels from the right edge of the ledger, draw a vertical line the length of the ledger page. Move four pixels to the left and draw a second red vertical line.
10. Choose black and draw a horizontal line that starts immediately above the topmost cyan area and goes the width of the ledger page.
11. When you've finished your icon, the Bitmap editor window should look like the following:

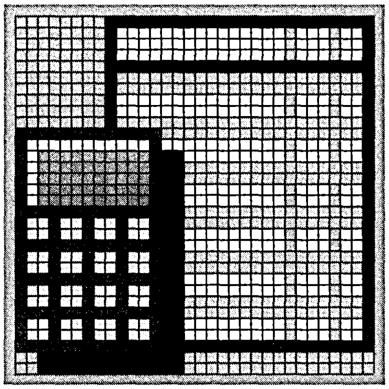


Figure 20.4
Finished Home
Budget icon

C H A P T E R 21

Creating cursors

Cursors are bitmapped images 32×32 pixels in size that represent the mouse pointer's current location on the screen. A Windows application often has a number of different cursors that represent different program functions.

Windows provides a set of standard cursors you can use in your programs. In addition, you can create your own special cursors to represent different functions of the program. One of these special cursors is the Resource Workshop Paint Can cursor that displays when you choose the Paint Can tool in the Bitmap editor.

To design your cursors, use the Resource Workshop Bitmap editor. See Chapter 19.

Working with cursors involves four basic steps:

1. Starting the Bitmap editor.

   If you are creating a new cursor resource, the Bitmap editor presents you with an empty cursor grid to work on. If you are modifying an existing cursor, it appears in the Bitmap editor.
2. Creating or modifying the cursor.
3. Testing the cursor.
4. Saving the cursor.

Creating a new cursor

You can add a new cursor to a project file, or you can create a standalone cursor file with the extension .CUR.

Adding a cursor to a project file

To add a cursor to an .RC (or .DLG) file,

1. Open a new or existing project. (Chapter 14 describes how you open a project.)

2. Choose Resource | New. Resource Workshop displays the New Resource dialog box.
3. Select CURSOR from the Resource Type list box.

   The name of the current project file appears in the box under "Place resource in." You can scroll down this list to pick another file (if any is listed) or click OK to accept the current project file.
4. Resource Workshop next asks whether you want to create the cursor in source form or in Microsoft-compatible binary format.
5. Choose Source to create a cursor that's embedded in the project file. Resource Workshop then starts the Bitmap editor.

   If you choose Binary, you create a standalone .CUR file (see page 320).

Embedded and linked resources are discussed on page 203.

Creating a standalone cursor file

You can create a standalone cursor file by doing either of the following:

- Choosing Binary from the dialog box that asks if you want to create your cursor in Source or Binary format.
- Choosing File | New Project and selecting .CUR from the New Project dialog box.

To link a standalone cursor file to a project file, open the project file and then choose File | Add to Project. (See page 205.)

Binary format option

When you choose Binary format, Resource Workshop displays the New File Resource dialog box so you can name your cursor file and specify the project in which it will be referenced. When the selected resource type is a cursor, the File Name text box says *.CUR.

1. Enter a name for your .CUR file in the File Name text box. Make sure the name of your project appears in the list box labeled "A reference to the resource...". To change the directory the file is stored in, set the path by double-clicking on the appropriate icons in the Directories list box. The path you have chosen is displayed in the Path text box.
2. When you've made your selections, click OK. Resource Workshop displays a dialog box saying the file you've named does not exist and asking if you want the file created. Click Yes.
3. When you exit the New File Resource dialog box, Resource Workshop starts the Bitmap editor.

You can also pick a file name from the Files list box. In that case, however, you'll overwrite the existing file. As a precaution, Resource Workshop asks you to verify that you want the file overwritten.

Cursor project file

If you choose File | New Project and select .CUR from the New Project dialog box, you automatically create a standalone cursor file. Resource Workshop immediately starts the Bitmap editor.

Editing cursors

The following sections describe several of the techniques you can use in editing your cursors.

Colors palette for cursors

The Colors palette for cursors is different from the Colors palette for the other bitmapped resources in one important respect: There are only two available foreground and background colors—black and white. A typical cursor is a white shape (an arrow or a question mark, for example) with a black outline.

Working with transparent and inverted areas

As mentioned on page 300, you can use transparent and inverted color areas in your cursors. At run time, the transparent area "drops out" and lets the desktop color show through from underneath. Any parts of the cursor painted with an inverted color take on the opposite color from the desktop underneath: white over black or teal over red, for example. Inverted colors are commonly used for the cursor's "hot spot."

When you start a new cursor, it is initially a grid made up entirely of the current transparent color. If you draw a single line of another color across the grid and then test the cursor, you see only that line against the desktop color; you don't see the grid outline or the area that is filled with the transparent color.

Don't set your transparent or inverted colors to black or white, or your cursor will either not be visible at run time or will be constantly changing color as the user moves the mouse over the application surface.

Setting the cursor's hot spot

An important consideration when you edit a cursor is where to put the hot spot, the cursor's active area. The hot spot is the single pixel in the cursor that fixes the location when the user places the cursor and clicks to make a selection.

To set a hot spot, decide on the exact pixel coordinates for the hot spot. Pixel coordinates are in horizontal (x) and vertical (y) units. The upper left pixel of the cursor image is x=0 and y=0. The lower right pixel for a 32×32 cursor is x=31 and y=31, and for a 32×16 cursor is x=31 and y=15.

To set the hot spot, do the following:

1. With the cursor in the Bitmap editor, zoom in on the cursor image until it's big enough to let you precisely choose the pixel coordinates for the hot spot.
2. Make sure the grid is displayed on the zoomed image. If necessary, choose Options | Editor Options and check Grid On Zoomed Windows.
3. Select a paint tool that lets you precisely point to a pixel. The Line tool is a good choice because it includes a cross hair to show exactly where its hot spot is.
4. With the paint tool you've chosen, point to the location on the zoomed image where you want the hot spot and look at the coordinates displayed on the status line. Make a note of these coordinates.
5. Choose Cursor | Set Hot Spot and type the coordinates.

C H A P T E R 22

Creating fonts

This manual uses "font" and "font resource" interchangeably.

According to the standard definition, a *font* is a set of characters of a given size and style. A *font resource* is a collection of data used by a computer to draw individual bitmapped images (which might be letters or other characters) on an output device such as a display monitor or a printer.

The font contains data that describes the overall collection of images, such as the typeface name, the suggested size, the character set, the letters in the font, and so on. The font also contains the information the computer needs to draw each bitmapped image.

Working with font resources involves five basic steps:

1. Starting the Bitmap editor by loading either a new font or an existing one.
2. Creating or editing a font image with the Bitmap editor.
3. Saving the font resource as part of a project file or in a separate file.
4. Exiting Resource Workshop and adding the font resource to a special resource-only DLL with an .FON extension.
5. Inserting a call to the .FON file in your program, compiling it, and testing the font resource.

Font types

Windows supports two basic types of fonts: raster fonts and outline fonts. Resource Workshop creates and edits Windows raster fonts only.

- Raster fonts contain a bitmapped image of each character.
- Outline fonts contain drawing commands for each character. Usually, they also contain "hints" the computer uses to produce better quality images at various sizes. The outline fonts that Windows supports lack these hints. These limited outline fonts are called vector fonts, of which Roman, Script, and Modern are examples. As a Windows user, more sophisticated font technology is available to you through third-party font rasterizers.

Why put bitmaps in a font?

Although you can use Resource Workshop to create customized letters, you'll probably want to use a specialized font development package to do that kind of work. You're more likely to use Resource Workshop to create *bitmapped image fonts*: small bitmaps you want to group together.

In some cases, font resources and bitmap resources aren't interchangeable. If you're creating a brush to paint an area onscreen, you must create it as a bitmap resource. But for creating bitmapped images to simply display onscreen, such as a bomb or a stop-sign bitmap image, you could use either bitmap or font resources. For example, the bomb shown at left was created as a font resource.

There are several reasons why you might want to define images as part of a font resource instead of as separate bitmap resources:

- It's simpler to write Windows code to load a font into memory and paint it than it is to load and paint the same image stored as a bitmap.
- A font can store up to 256 bitmaps. If your program typically uses a certain set of bitmapped images at the same time, you can put these images in a font resource, and then the program can load just this one resource when it needs this set of images.
- A font can store multiple images more efficiently than the same images stored as individual bitmap resources.

Note, however, that a font resource has a certain amount of memory overhead; each time you load a font into memory, Windows also loads the font header (see page 329). If you're creating a single image and are trying to decide whether to create it as a font resource or as a bitmap resource, you should define it as a bitmap if efficient memory use is important.

Creating a new font resource

You can add a new font resource to a project file, or you can create a standalone font file with the extension .FNT.

Adding a font to a project file

To add a font to an .RC (or .DLG) file,

1. Open a new or existing project. (Chapter 14 describes how you open a project.)
2. Choose Resource | New. Resource Workshop displays the New Resource dialog box.
3. Select FONT from the Resource Type list box.

The name of the current project file appears in the box under "Place resource in." You can scroll down this list to pick another file (if any is listed) or click OK to accept the current project file.

4. Resource Workshop next asks whether you want to create the font in source form or in Microsoft-compatible binary format.

   Embedded and linked resources are discussed on page 203.

   Choose Source to create a font that's embedded in the project file. Resource Workshop then starts the Bitmap editor.

   If you choose Binary, you create a standalone .FNT file (see the next section).

Creating a standalone font file

You can create a standalone font file by doing either of the following:

- Choosing Binary from the dialog box that asks if you want to create your font in Source or Binary format.
- Choosing File | New Project and selecting .FNT from the New Project dialog box.

To link a standalone font file to a project file, open the project file and then choose File | Add to Project. (See page 205.)

Binary format option

When you choose Binary format, Resource Workshop displays the New File Resource dialog box so you can name your font file and specify the project in which it will be referenced. When the selected resource type is a font, the File Name text box says *.FNT.

1. Enter a name for your .FNT file in the File Name text box. Make sure the name of your project appears in the list box labeled "A reference to the resource...". To change the directory the file is stored in, set the path by double-clicking on the appropriate fonts in the Directories list box. The path you have chosen is displayed in the Path text box.
2. When you've made your selections, click OK. Resource Workshop displays a dialog box saying the file you've named does not exist and asking if you want the file created. Click Yes.
3. When you exit the New File Resource dialog box, Resource Workshop starts the Bitmap editor.

You can also pick a file name from the Files list box. In that case, however, you'll overwrite the existing file. As a precaution, Resource Workshop asks you to verify that you want the file overwritten.

Font project file

If you choose File | New Project and select .FNT from the New Project dialog box, you automatically create a standalone font file. Resource Workshop immediately starts the Bitmap editor.

Editing a font resource

When you open a new or existing font resource in the Bitmap editor, the characters or images in the font you're editing are displayed in a "list" in the right border of the window.

When you start the Bitmap editor, it automatically loads the first font image in the list. To load another image for editing, click on it in the right border of the Bitmap editor. Then use the Bitmap editor tools to make changes.

Defining and adding characters for a font

When you create a new font resource, it includes only one 8×8 pixel image. Usually you'll want more than one image in your font resource. You also might want to specify a different size for your font images. To specify more than one image in a font resource and to change the size of font images, use the Font | Font Size option.

Defining the font size

To define the font size,

1. Choose Font | Font Size to display the Font Size Information dialog box.
2. Make your size choices among the Size options described in Table 22.1. The images in a font resource can be *variable-width* or *fixed-width.*

See page 328 for more information on variable-width fonts.

   - Fonts containing letters or images that can vary in width are called variable-width or *proportional* fonts. For example, in a variable-width font the letter "m" is considerably wider than the letter "i." Books (including this one) are almost always printed in a variable-width font.
   - Fonts in which the characters or images are all the same width are called fixed-width or *monospaced* fonts. Most typewriters use fixed-width fonts.

Variable-width characters usually take less space and are more pleasing to the eye than fixed-width font characters. Compare the following two lines, especially the spacing around the letter "i."

Minimum (variable-width font)
`Minimum (fixed-width font)`

Table 22.1
Font size options

Option	Description
Width	If you want all images to be the same width (fixed-width), type the width in pixels. If you want the widths to vary (variable-width), type a 0 (zero) here and specify a maximum width.
Height	Type the height, in pixels, of the font images.
Average Width	Resource Workshop calculates an average width for your font images if you have specified 0 for Width (font is variable-width). Otherwise (font is fixed-width), Average Width is the same as Width. The Average Width is calculated when you open this dialog box. (You won't see this value change if you type other changes into this dialog box.)
Maximum Width	For variable-width fonts, specify the maximum width in pixels. This option is available only if you have typed 0 next to Width.
Stretch current chars	Check this option if you want the height or width of existing images to increase or decrease, based on height and width changes you type into this dialog box. Checking this option allows you to distort (stretch) characters from their original shape.

Setting the number of characters

The Font Size Information dialog box also lets you choose how many images to include in your font resource. The First and Last values of the Character options (see Table 22.2) determine how many characters the font resource will hold.

If you set the number too low, you can change the values to increase the number of available characters.

Mapping the character set

Choose a range of decimal codes to use in mapping your font images to the ANSI character set. For example, to map a font image to the character *a*, specify the decimal code 97. The image itself needn't be the character *a*, unless you want it to be. In the sample font resource discussed at the end of this chapter, a bomb image is mapped to *a*. The second font image would then be mapped to the character *b*, decimal code 98.

➡ The ANSI values to which you map a set of bitmapped images are arbitrary, but must be in the range 0 to 255. These ANSI values become important when you load the font in your program and display the bitmaps, because you use the ANSI value that corresponds to a bitmap to display it, the same as you would use an ANSI value to display a character.

Use the following Character options to map the character set or images in your font resource:

Table 22.2
Character options

Option	Description
First	Type an ANSI decimal code to define the first image in your font. For example, if you want the first image to correspond to *a*, type `97`.
Last	Type an ANSI decimal code to define the last image in your font. For example, if you want the last image to correspond to *z*, type `122`.
Default	Type an ANSI decimal code to define the default font image that will be displayed when you edit this font resource. The Default value must be within the character range defined by the First and Last values. For example, if you've typed 97 for First and 122 for Last, you can't type 88 for Default.
Break	Type an ANSI decimal code to define a break character for your font resource. (A break character, typically the space character, is used to pad justified lines.) The Break value must be within the character range defined by the First and Last values.

Creating variable-width fonts

To create variable-width images or characters in your font resource,

1. Choose Font | Font Size.
2. Next to Width, type `0` (zero).
3. Next to Maximum Width, type the maximum width (in pixels) for all the images or characters in the font resource.

Setting the width of a character or image

After you've chosen a width of zero in the Font Size Information dialog box, you can choose the width for an individual character or image using the Font | Character Width command.

For example, suppose you've used the Font | Font Size option to define the following sizes for your font resource, which contains bitmapped images:

Width	0
Height	32
Maximum Width	32

Based on these sizes, all images will be 32 pixels high and a maximum of 32 pixels wide. As you're editing a particular image, you can use the Font | Character Width option to define the width of that image.

Choosing Font | Character Width displays the Character Width dialog box.

Use Stretch Current Chars to distort the image from its normal proportions.

Next to Width, type a value that's less than or equal to Maximum Width. In addition, you can check the Stretch Current Chars option if you want the existing image to stretch or shrink based on the width change you type into this dialog box.

Defining a header for a font resource

Every font resource includes a header that describes general information about the font, such as typeface name and copyright information. If you're defining a font resource that consists of alphanumeric characters, the header defines typestyle and size for all characters in the font.

To define the header for a font,

1. Display the font in the Bitmap editor by double-clicking the font name in the project window, or by using Resource | New to create a new font resource.
2. Choose Font | Header to specify header information. You see the Font Header Information dialog box.
3. Define the header for your customized fonts. The following table describes your choices:

Table 22.3
Font header options

Option	Description
Face Name	Type a name that you want to assign to your font.
Device	Type a device name for your font if you want to inform your programs that this font can be used only on a particular device.
Copyright	Type copyright information for your custom font.
Font Version	Font version 2.00 is supported in all cases. You can use version 3.00 if you're creating a Windows 3.*x* application that will run in a protected mode environment (Standard mode or 386 Enhanced mode) on an 80386 (or later) processor.
Attributes	
– Italic	The font contains italicized characters.
– Underline	The font contains underlined characters.
– Strikeout	The font contains characters that are struck out.
– Variable Pitch	The font is a variable-width font.
– Weight	The font is of normal weight (400) or boldfaced (700).
– Family	Describes the font family. The acceptable values are 0 Don't care 3 Modern 1 Roman 4 Script 2 Swiss 5 Decorative
– Char set	Defines the character set. The value can be 0 through 255. 0, 2, and 255 have the following predefined meanings:

Table 22.3: Font header options (continued)

	0	ANSI, the default Windows character set
	2	Symbol, used for math and scientific formulas
	255	OEM, a machine-specific character set
Sizes		
– Horz Res	Horizontal number of pixels per logical inch on your video display.	
– Vert Res	Vertical number of pixels per logical inch on your video display.	
– Points	Type size. A point is 1/72 of an inch. A character is measured from the top of the ascender to the bottom of the descender. The value you enter here should not include space for diacritical marks.	
– Internal Leading	The space in pixels reserved for diacritical marks.	
– External Leading	The additional space in pixels between lines of characters.	
– Ascent	The height in pixels of the character above the baseline.	

Changing size and attributes

When you're editing a font resource in the Bitmap editor, the Font menu has three commands that let you make changes to the font images.

- Header defines the header information for your font resource, including the font version, the font name, copyright information, and so on. For more information, see page 329.
- Font Size defines the character set in this font, and the width and height of each character. For more information, see page 326.
- Character Width specifies the width for a particular image in a variable width font resource. (This command won't be available unless you've already defined a variable width font using Font | Font Size.) For more information, see page 328.

Using your fonts in your applications

With other resources, you use Resource Workshop or Borland Resource Compiler (BRC) to bind your resources to your executable files. However, fonts can't be bound to an executable file. What you must do instead is create a special, resource-only DLL that has an .FON extension, and then load it into your program by using the Windows function **AddFontResource**.

You can create an .FON file using either C++ or Turbo Pascal and Turbo Assembler.

C H A P T E R 23

Creating user-defined resources

In addition to the resource types discussed in previous chapters, you can also define your own resources. After you create a new resource type, you can add any number of *user-defined resources* of this type to your project.

You might want to define your own resource types to contain data that doesn't fit into one of the standard resource types. For example, if you want to create a character string resource that's longer than the STRINGTABLE limit of 255 characters, you can define your own resource type and store your character strings there.

See Charles Petzold's *Programming Windows* for more information on metafiles.

You can also include *metafiles* in your project as user-defined resources. A metafile is a type of bitmap (in source form, it's a collection of Graphics Device Interface (GDI) calls) that's not only easier to scale and more device-independent than the standard Bitmap resource, but also often takes up less storage space than a Bitmap resource.

When you define a new resource, you can store the data either as part of the resource definition in a project file or as a separate file. As with any resource, Resource Workshop can compile the data and bind it to your executable file to make the data available to your application at run time.

➡ You can also use the RCDATA resource type to add data to your application. See page 335 for more information.

Working with user-defined resources involves five basic steps:

1. Creating a user-defined resource type.
2. Adding the user-defined resource to your project.
3. Editing the user-defined resource.
4. Testing the user-defined resource.
5. Saving the user-defined resource.

Creating a resource type

Before you can add a user-defined resource to your project, you must first create a type for it, as follows:

Chapter 14 describes how you open a project.

1. Open a project.
2. Choose Resource | New.
3. In the New Resource dialog box, click the New Type button. Resource Workshop displays the New Resource Type dialog box.
4. In the New Resource Type dialog box, type a name for the resource type you're creating. For example, if you're creating a resource to contain a large block of text, you could name your new type TEXT. Press *Enter*.
5. When asked if you want to create an identifier for the new resource type, click Yes, then type a value for the identifier in the Value box of the New Identifier dialog box. This value is the ID that Windows and your program will associate with this identifier type.

If you use a numeric ID, it must be greater than 255, because Windows reserves the values 1 through 255 for standard resources.

Once you've defined a new resource type, whenever you create a new resource, you'll see that resource type listed in the New Resource dialog box with the standard resource types (BITMAP, DIALOG, and so on).

Adding user-defined resources

After you've created a resource type, you can add a resource of that type to your project, as follows:

Chapter 14 describes how you open a project.

1. Open a project.
2. Choose Resource | New. The New Resource dialog box appears.
3. In the New Resource dialog box, select your user-defined resource type. Under "Place resource in", select the project file (probably the current one) where you want to store the resource.
4. After you click OK in the New Resource dialog box, Resource Workshop opens the text editor with a blank definition for your user-defined resource. For example, if you add a user-defined resource called TEXT, you see the following code in the text editor:

```
TEXT_1 TEXT
{

}
```

The next section describes how you edit a user-defined resource.

Editing user-defined resources

Chapter 14 describes how you open a project.

To edit a user-defined resource, you must have a project open. You can either create the resource (see the previous section) or you can open an already existing one in the Project window.

The resource name is listed in the Project window by its user-defined resource type, just as any resource of a predefined type is listed. For example, if you create a TEXT resource type and a Text resource, you see a TEXT resource entry in the Project window.

To open the editor,

- Double-click the name of the resource you want to edit, or
- Select the resource name and then choose Resource | Edit or Resource | Edit As Text (both start the text editor).

Once you've brought the resource up in the text editor, you can add data to it or edit the data in it. Here's an example of what you'll see when you add a new user-defined resource of type RESTYPE to your project:

```
RESTYPE_1 RESTYPE
BEGIN
  [data definitions]
END
```

The first line shows the resource name and type. Resource Workshop constructs a default name for a new resource by appending to the resource type an underscore and an integer. For example, the first RESTYPE resource you add to your project becomes RESTYPE_1, the second becomes RESTYPE_2, and so on.

To add data to your resource, do one of the following:

- Use the text editor to type data between the curly braces.
- Store the data in a separate file and add the file name to the end of the first line of the resource script, as follows:

  ```
  RESTYPE_1 RESTYPE MYRES.FIL
  ```

  You must also delete the lines containing the BEGIN and END statements.
- Use the Add to Project command, as described in the next section.

➡ After you make any changes to the resource script, you must recompile the resource to save your changes. If you exit without recompiling, the changes will be lost.

Embedding resource data in the project file

As noted in the previous section, you can add data to your resource by storing the data in a separate file. The disadvantage to this approach is that, if something happens to the file, the data is lost. Another option is to embed the external data into the project file script.

If, for example, you have a user-defined resource type called TEXT and a resource data file called MY_RES.TXT, you can embed the data in the project by doing the following:

1. Choose File | Add to Project. Resource Workshop displays the Add File to Project dialog box.
2. Select "user Resource data" from the File Type list box.
3. Change the File Name text box to `*.TXT`.
4. Change the directory (if necessary) until MY_RES.TXT is visible in the File listing. Double-click MY_RES.TXT.
5. Resource Workshop displays the Custom Resource Type dialog box. Double-click the entry for the TEXT resource type. A new TEXT resource appears in the Project window.

You can also enter the full path and file name into the File Name text box.

➥ If you select the new resource and choose Resource | Edit (or Edit as Text—the effect is the same), you'll see that the resource data has been converted to hexadecimal format. For that reason, you should keep the external data file available in case you want to edit the resource script later.

Entering data in the resource script

When you use the text editor, *Tab, Del, Home, End, PgUp, PgDn,* and *Backspace* function as usual. However, don't use this editor to do much formatting because Resource Workshop is likely to rearrange the text when it compiles or decompiles this resource.

Here are some guidelines for specifying data between the BEGIN and END parameters:

- The data can include any combination of numeric values and strings.
- You can use hexadecimal, octal, or decimal notation to represent numeric values.
  - Use either 0x (a zero followed by the letter *x*) or $ (a dollar sign) as the leading characters for hexadecimal notation. This notation supports only 16-bit values. If you want to use an odd number of hexadecimal values, use the *hexstring* data type (described in the next bullet).
  - Use 0o (a zero followed by the letter *o*) or just 0 (zero) as the leading characters for octal notation.

- You can also represent hexadecimal values by using a *hexstring* of hexadecimal values enclosed in single quotation marks. The compiler ignores any spaces you insert to make the hex codes more readable. For example, you could represent the ASCII values of the characters in the word *string* as `'73 74 72 69 6E 67'` and the hex number 06E07A as `'06E07A'`.
- If you include text strings in your resource, enclose the strings in quotation marks, like this: `"string"`. Strings aren't automatically null terminated. To terminate a string with a null character, type `\0` (backslash zero) at the end of the string.

For example, your resource script with data added could look like:

```
RESTYPE_1 RESTYPE
BEGIN
  "This is a string."
  0xFFAA 0o7076 01077
  '54 68 69 73 20 69 73 0D 0A 73 6F 6D 65 20 73 61'
  '6D 70 6C 65 0D 64 61 74 61 2E'
END
```

Handling data stored in a separate file

If the data is stored in a separate file, you must use an external editor to edit the file. To add a reference to the data file to the resource script,

- On the first line of the script, following the name of the resource type, enter the complete path name of the file.
- Delete the BEGIN and END statements.

The following resource script for the resource RESTYPE_1 indicates that the data is stored in the file C:\RW\MYDATA.TXT: `RESTYPE_1 RESTYPE c:\rw\mydata.txt`

Using the RCDATA resource type

RCDATA resources have been included in Resource Workshop for compatibility purposes.

You can use the predefined RCDATA resource type to add a data resource to your application. It works the same as a user-defined resource type. The main difference between the two is addressability: you might prefer to have many different types of user-defined resources rather than just one.

If you do use an RCDATA resource, add it to the project by choosing File | New and adding a new resource whose type is RCDATA. You'll see a blank RCDATA definition in the text editor, and you can type the data between the BEGIN and END parameters. Use the same rules for typing data as described for user-defined resources.

APPENDIX A

Error messages

This appendix describes the error messages that can be generated by Borland C++. The error messages in this appendix include messages that can be generated by the compiler, the MAKE utility, the librarian (TLIB), the linker (TLINK), and the Windows Help compiler. This appendix also lists the errors that you can receive when you run your program (run-time errors).

Messages are listed in ASCII alphabetic order. Messages beginning with symbols come first, then messages beginning with numbers, and then messages beginning with letters of the alphabet. Messages that begin with symbols are alphabetized by the first word in the message that follows the symbols. For example, you might receive the following error message if you incorrectly declared your function *my_func*:

```
my_func must be declared with no parameters
```

To find this error message, look under the alphabetized listing of "must."

Message classes

Messages fall into three categories: fatal errors, errors, and warnings.

Fatal errors

Fatal errors can be generated by the compiler, the linker, and the MAKE utility. Fatal errors cause the compilation to stop immediately; you must take appropriate action to fix the error before you can resume compiling.

If the compiler or MAKE utility issues a fatal error, no .EXE file is created. If the linker issues a fatal error, any .EXE file that might have been created by the linker is deleted before the linker returns.

Errors

Errors can be generated by the compiler, the linker, the MAKE utility, and the librarian. In addition, errors can be generated by your program at run time.

Errors generated by the compiler indicate program syntax errors, command-line errors, and disk or memory access errors. Compiler errors don't cause the compilation to stop—the compiler completes the current phase of the compilation and then stops and reports the errors encountered. The compiler attempts to find as many real errors in the source program as possible during each phase (preprocessing, parsing, optimizing, and code-generating).

Errors generated by the linker don't cause the linker to delete the .EXE or .MAP files. However, you shouldn't execute any .EXE file that was linked with errors. Linker errors are treated like fatal errors if you're compiling from the Integrated Development Environment (IDE).

The MAKE utility generates errors when there is a syntax or semantic error in the source makefile. You must edit the makefile to fix these types of errors.

Run-time errors are usually caused by logic errors in your program code. If you receive a run-time error, you must fix the error in your source code and recompile the program for the fix to take effect.

Warnings

Warnings can be issued by the compiler, the linker, and the librarian. Warnings do not prevent the compilation from finishing. However, they do indicate conditions that are suspicious, even if the condition that caused the warning is legitimate within the language. The compiler also produces warnings if you use machine-dependent constructs in your source files.

Help compiler messages

The Help compiler displays messages when it encounters errors or warnings while building the Help resource file. Messages generated during the processing of the project file begin with the letter **P** and are followed by an error number. Messages that occur during the processing of the RTF topic file(s) begin with the letter **R** and are followed by an error number.

Whenever possible, the Help compiler displays the topic number and/or the file name that contains the error. If you've numbered your topics, the topic number is given with an error message that refers to that topic's sequential position in your RTF file (first, second, and so on). These numbers might be identical to the page number shown by your word processor. In your Help source files, topics are separated by hard page breaks even though there are no "pages" in the Help system.

Messages beginning with the word "Error" are fatal errors. Errors are always reported, and no usable Help resource file will result from the build. Messages beginning with the word "Warning" are less serious in nature. A build with warnings produces a valid Help resource file that loads under Windows, but the file might contain operational errors.

Message listings

Messages are written with the message class first, followed by the source file name and line number where the error was detected, and finally with the text of the message itself.

Some messages include a symbol (such as a variable, file name, or module) that is taken from your program. Symbols in the message explanations are shown in *italics* to indicate that they're variable in nature.

Be aware that the compiler generates messages as they are detected. Because C and C++ don't force any restrictions on placing statements on a line of text, the true cause of the error might be one or more lines before or after the line number mentioned in the error message.

Message explanations

')' missing in macro invocation *MAKE error*
A left parenthesis is required to invoke a macro.

(expected *Compiler error*
A left parenthesis was expected before a parameter list.

) expected *Compiler error*
A right parenthesis was expected at the end of a parameter list.

, expected *Compiler error*
A comma was expected in a list of declarations, initializations, or parameters.

: expected after private/protected/public *Compiler error*
When used to begin a **private/protected/public** section of a C++ class, these reserved words must be followed by a colon.

< expected *Compiler error*
The keyword **template** was not followed by a left angle bracket (<). Every template declaration must include the template formal parameters enclosed within angle brackets (< >), immediately following the **template** keyword.

> expected *Compiler error*
A new-style cast (for example, dynamic_cast) is missing a closing ">".

@ seen, expected a response-files name *Librarian error*
The response file is not given immediately after @.

{ expected *Compiler error*
A left brace ({) was expected at the start of a block or initialization.

} expected *Compiler error*
A right brace (}) was expected at the end of a block or initialization.

16-bit segments not supported in module *module* *Linker error*
16-bit segments aren't supported in 32-bit applications. Check to make sure that you haven't inadvertently compiled your 32-bit application using the 16-bit compiler.

286/287 instructions not enabled *Compiler error*
Use the **–2** command-line compiler option or the 80286 options from the Options|Compiler|Code Generation|Advanced Code Generation dialog box to enable 286/287 opcodes. The resulting code cannot be run on 8086- and 8088-based machines.

32-bit record encountered *Linker error*
An object file that contains 80386 32-bit records was encountered, and the **/3** option had not been used.

Abnormal program termination *Run-time error*
The program called *abort* because there wasn't enough memory to execute. This can happen as a result of memory overwrites.

Access can only be changed to public or protected *Compiler error*
A C++ derived class can modify the access rights of a base class member, but only to **public** or **protected**. A base class member cannot be made **private**.

Added file *filename* does not begin correctly, ignored *Librarian warning*
The librarian has decided that the file being added is not an object module, so it will not try to add it to the library. The library is created anyway.

Address of overloaded function *function* doesn't match *type* *Compiler error*
A variable or parameter is assigned/initialized with the address of an overloaded function, and the type of the variable/parameter doesn't match any of the overloaded functions with the specified name.

***module* already in LIB, not changed!** *Librarian warning*
An attempt to use the + action on the library has been made, but there is already an object with the same name in the library. If an update of the module is desired, the action should be +-. The library has not been modified.

Ambiguity between *function1* and *function2* *Compiler error*
Both of the named overloaded functions could be used with the supplied parameters. This ambiguity is not allowed.

Ambiguous member name *name* *Compiler error*
A structure member name used in inline assembly must be unique. If it is defined in more than one structure all of the definitions must agree in type and offset within the structures. The member name in this case is ambiguous. Use the syntax `(struct` *`xxx`*`).`*`yyy`* instead.

Ambiguous Override of Virtual Base Member *func1*: *func2* *Compiler error*
A virtual function in a virtual base class was overridden with two or more different functions along different paths in the inheritance hierarchy.

Ambiguous operators need parentheses *Compiler warning*

This warning is displayed whenever two shift, relational, or bitwise-Boolean operators are used together without parentheses. Also, an addition or subtraction operator that appears unparenthesized with a shift operator will produce this warning. Programmers frequently confuse the precedence of these operators.

Ambiguous override of virtual base member *base_function*: *derived_function* *Compiler error*

This error message is issued when a virtual function that is defined in a virtual base class is overridden with different functions having two derived classes in the same inheritance hierarchy. For example,

```
struct VB
{
   virtual f();
};

struct A:virtual VB
{
   virtual f();
};

struct B:virtual VB
   virtual f();
};

struct D:A,B
{
}      //errors here
```

The above code will be flagged with the following errors:

```
Error: Ambiguous override of virtual base member VB::f():A::f()
Error: Ambiguous override of virtual base member VB::f():B::f()
```

Application load & execute error 0001 *Compiler error*
Application load & execute error FFE0 *Compiler error*

There was insufficient extended memory available for the protected-mode command-line tool to load.

Array allocated using new may not have an initializer *Compiler error*

When initializing a vector (array) of classes, you must use the *default constructor* (the constructor that has no arguments).

Array bounds missing] *Compiler error*

Your source file declared an array in which the array bounds were not terminated by a right bracket.

Array must have at least one element *Compiler error*

ANSI C and C++ require that an array be defined to have at least one element (objects of zero size are not allowed). An old programming trick declares an array element of a structure to have zero size, then allocates the space actually needed with *malloc*. You can still use this trick, but you must declare the array element to have (at least) one element if you are compiling in strict ANSI mode. Declarations (as opposed to definitions) of arrays of unknown size are still allowed.

For example,

```
char ray[];         /* definition of unknown size -- illegal */
char ray[0];        /* definition of 0 size -- illegal */
extern char ray[];  /* declaration of unknown size -- ok */
```

Array of references is not allowed *Compiler error*

It is illegal to have an array of references because pointers to references are not allowed and array names are coerced into pointers.

Array size for 'delete' ignored *Compiler warning*

With the latest specification of C++, it is no longer necessary to specify the array size when deleting an array; to allow older code to compile, Borland C++ ignores this construct and issues this warning.

Array size too large *Compiler error*

The declared array is larger than 64K.

Array variable *identifier* is near *Compiler warning*

Whenever you use either the **-Ff** or **-Fm** command-line options to set threshold limit, global variables larger than the threshold size are automatically made far by the compiler. However, when the variable is an initialized array with an unspecified size, its total size is not known when the decision whether to make it near or far has to be made by the compiler, and so it is made near. If the number of initializers given for the array causes the total variable size to exceed the data size threshold, the compiler issues this warning. If problems are caused by having the variable made near (for example, if the linker reports a group overflow due to too much global data), you must make the offending variable explicitly far by inserting the keyword **far** immediately to the left of the variable name in its definition.

Assembler statement too long *Compiler error*

Inline assembly statements cannot be longer than 480 bytes.

Assigning *type* to *enumeration* *Compiler warning*

Assigning an integer value to an **enum** type. This is an error in C++, but is reduced to a warning to give existing programs a chance to work.

Assignment to this not allowed, use X::operator new instead *Compiler error*

In early versions of C++, the only way to control allocation of a class of objects was to use the **this** parameter inside a constructor. This practice is no longer allowed because a better, safer, and more general technique is to instead define a member function **operator new.**

Attempt to export non-public symbol *symbol* *Linker error*

This error usually occurs when a .DEF file specifies an EXPORT for a symbol that you either forgot to define or misspelled.

Attempt to grant or reduce access to *identifier* *Compiler error*

A C++ derived class can modify the access rights of a base class member, but only by restoring it to the rights in the base class. It cannot add or reduce access rights.

Attempting to return a reference to a local object *Compiler error*

In a function returning a reference type, you attempted to return a reference to a temporary object (perhaps the result of a constructor or a function call). Because this object will disappear when the function returns, the reference will then be illegal.

Attempting to return a reference to local variable *identifier* *Compiler error*

This C++ function returns a reference type, and you are trying to return a reference to a local (auto) variable. This is illegal because the variable referred to disappears when the function exits. You can return a reference to any static or global variable, or you can change the function to return a value instead.

Bad call of intrinsic function *Compiler error*

You have used an intrinsic function without supplying a prototype, or you supplied a prototype for an intrinsic function that was not what the compiler expected.

Bad character in parameters -> *char* *Linker error*

One of the following characters (or any control character other than horizontal tab, linefeed, carriage return, or *Ctrl+Z*) was encountered in the command line or in a response file:

```
" * < = > ? [ ] |
```

Bad define directive syntax *Compiler error*

A macro definition starts or ends with the ## operator, or contains the # operator that is not followed by a macro argument name.

Bad field list in debug information in module *module* *Linker error*

This is typically caused by bad debug information in the OBJ file. Borland Technical Support should be informed.

Bad file name *filename* *Linker error*

An invalid file name was passed to the linker.

Bad file name format in include directive *Compiler error*

Include file names must be surrounded by quotes ("FILENAME.H") or angle brackets (<FILENAME.H>). The file name was missing the opening quote or angle bracket. If a macro was used, the resulting expansion text is incorrect; that is, it is not surrounded by < > or " ".

Bad filename format in include statement *MAKE error*

Include file names must be surrounded by quotes or angle brackets. The file name was missing the opening quote or angle bracket.

Bad file name format in line directive *Compiler error*

Line directive file names must be surrounded by quotes ("FILENAME.H") or angle brackets (<FILENAME.H>). The file name was missing the opening quote or angle bracket. If a macro was used, the resulting expansion text is incorrect; that is, it is not surrounded by quote marks.

Bad GCD type in GRPDEF, extended dictionary aborted *Librarian warning*
Bad GRPDEF type encountered, extended dictionary aborted *Librarian warning*

The librarian has encountered an invalid entry in a group definition (GRPDEF) record in an object module while creating an extended dictionary. The only type of GRPDEF record that the librarian (and linker) supports is segment index type. If any other type of GRPDEF is encountered, the librarian won't be able to create an extended dictionary. It's possible that an object module created by products other than Borland tools can create GRPDEF records of other types. It's also possible for a corrupt object module to generate this warning.

Bad header in input LIB *Librarian error*

When adding object modules to an existing library, the librarian has determined that it has a bad library header. Rebuild the library.

Bad ifdef directive syntax *Compiler error*

An **#ifdef** directive must contain a single identifier (and nothing else) as the body of the directive.

Bad LF_POINTER in module *module* *Linker error*

This is typically caused by bad debug information in the OBJ file. Borland Technical Support should be informed.

Bad macro output translator *MAKE error*

Invalid syntax for substitution within macros.

Bad object file *filename* near file offset *offset* *Linker error*

The linker has found a bad OBJ file. This is usually caused by a translator error.

Bad object file record *Linker error*
Bad object file *file* near file offset *offset* *Linker error*

An ill-formed object file was encountered. This is most commonly caused by naming a source file or by naming an object file that was not completely built. This can occur if the machine was rebooted during a compile, or if a compiler did not delete its output object file when a *Ctrl+Break* was pressed.

Bad OMF record type *type* encountered in module *module* *Librarian error*

The librarian encountered a bad Object Module Format (OMF) record while reading through the object module. The librarian has already read and verified the header records on the *module*, so this usually indicates that the object module has become corrupt in some way and should be re-created.

Bad syntax for pure function definition *Compiler error*

Pure virtual functions are specified by appending "= 0" to the declaration. You wrote something similar, but not quite the same.

Bad undef directive syntax *Compiler error*

An #**undef** directive must contain a single identifier (and nothing else) as the body of the directive.

Bad undef statement syntax *MAKE error*

An **!undef** statement must contain a single identifier and nothing else as the body of the statement.

Bad version number in parameter block *Linker error*

This error indicates an internal inconsistency in the IDE. If it occurs, exit and restart the IDE. This error does not occur in the standalone version.

Base class *class* contains dynamically dispatchable functions *Compiler error*

Currently, dynamically dispatched virtual tables do not support the use of multiple inheritance. This error occurs because a class that contains DDVT functions attempted to inherit DDVT functions from multiple parent classes.

Base class *class* is inaccessible because also in *class* *Compiler warning*

It is not legal to use a class as both a direct and indirect base class because the members are automatically ambiguous. Try making the base class virtual in both locations.

Base class *class* is included more than once *Compiler error*

A C++ class can be derived from any number of base classes, but can be directly derived from a given class only once.

Base class *class* is initialized more than once *Compiler error*

In a C++ class constructor, the list of initializations following the constructor header includes base class *class* more than once.

Base initialization without a class name is now obsolete *Compiler error*

Early versions of C++ provided for initialization of a base class by following the constructor header with just the base class constructor parameter list. It is now recommended that you include the base class name: this makes the code much clearer, and is required when there are multiple base classes.

Old way:

```
derived::derived(int i) : (i, 10) { ... }
```

New way:

```
derived::derived(int i) : base(i, 10) { ... }
```

Bit field cannot be static *Compiler error*

Only ordinary C++ class data members can be declared **static,** not bit fields.

Bit field too large *Compiler error*
This error occurs when you supply a bit field with more than 32 bits.

Bit fields must be signed or unsigned int *Compiler error*
In ANSI C, bit fields can only be **signed** or **unsigned int** (not **char** or **long**, for example).

Bit fields must be signed or unsigned int *Compiler warning*
In ANSI C, bit fields cannot be of type **signed char** or **unsigned char**. When not compiling in strict ANSI mode, the compiler allows such constructs, but flags them with this warning.

Bit fields must contain at least one bit *Compiler error*
You cannot declare a named bit field to have 0 (or less than 0) bits. You can declare an unnamed bit field to have 0 bits, a convention used to force alignment of the following bit field to a byte boundary (or word boundary, if you select Word Alignment). In C++, bit fields must have an integral type; this includes enumerations.

Bit fields must have integral type *Compiler error*
In C++, bit fields must have an integral type; this includes enumerations.

Body has already been defined for function *function* *Compiler error*
A function with this name and type was previously supplied a function body. A function body can be supplied only once.

Both return and return with a value used *Compiler warning*
The current function has **return** statements with and without values. This is legal in C, but is almost always an error. Possibly a **return** statement was omitted from the end of the function.

Call of nonfunction *Compiler error*
The name being called is not declared as a function. This is commonly caused by incorrectly declaring the function or by misspelling the function name.

Call to function *function* with no prototype *Compiler warning*
The "Prototypes required" warning was enabled and you called function *function* without first giving a prototype for it.

Call to undefined function *function* *Compiler error*
Your source file declared the current function to return some type other than **void** in C++ (or **int** in C), but the compiler encountered a return with no value. This is usually some sort of error. **int** functions are exempt in C because in old versions of C there was no **void** type to indicate functions that return nothing.

***virtual* can only be used with member functions**
A data member has been declared with the virtual specifier. Only member functions can be declared virtual.

Cannot access an inactive scope *Compiler error*
You have tried to evaluate or inspect a variable local to a function that is currently not active. (This is an integrated debugger expression evaluation message.)

Cannot add or subtract relocatable symbols *Compiler error*
The only arithmetic operation that can be performed on a relocatable symbol in an assembler operand is addition or subtraction of a constant. Variables, procedures, functions, and labels are relocatable symbols. Assuming that *Var* is a variable and *Const* is a constant, then the instructions

```
MOV AX,Const+Const
```

and

```
MOV AX,Var+Const
```

are valid, but `MOV AX,Var+Var` is not.

Cannot allocate a reference *Compiler error*

An attempt to create a reference using the **new** operator has been made; this is illegal because references are not objects and cannot be created through **new**.

***identifier* cannot be declared in an anonymous union** *Compiler error*

The compiler found a declaration for a member function or static member in an anonymous union. Such unions can contain data members only.

function1* cannot be distinguished from *function2 *Compiler error*

The parameter type lists in the declarations of these two functions do not differ enough to tell them apart. Try changing the order of parameters or the type of a parameter in one declaration.

Cannot call *main* from within the program *Compiler error*

C++ does not allow recursive calls of the function *main*.

Cannot call near class member function with a pointer of type *type* *Compiler error*

Member functions of near classes (classes are near by default in the tiny, small, and medium memory models) cannot be called using far or huge member pointers. (Note that this also applies to calls using pointers to members.) Either change the pointer to be near, or declare the class as far.

Cannot cast from *type1* to *type2* *Compiler error*

A cast from type *type1* to type *type2* is not allowed. In C, a pointer can be cast to an integral type or to another pointer. An integral type can be cast to any integral, floating, or pointer type. A floating type can be cast to an integral or floating type. Structures and arrays cannot be cast to or from. You cannot cast from a **void** type.

C++ checks for user-defined conversions and constructors, and if one cannot be found, then the preceding rules apply (except for pointers to class members). Among integral types, only a constant zero can be cast to a member pointer. A member pointer can be cast to an integral type or to a similar member pointer. A similar member pointer points to a data member if the original does, or to a function member if the original does; the qualifying class of the type being cast to must be the same as or a base class of the original.

Cannot convert *type1* to *type2* *Compiler error*

An assignment, initialization, or expression requires the specified type conversion to be performed, but the conversion is not legal.

Cannot create instance of abstract class *class* *Compiler error*

Abstract classes—those with pure virtual functions—cannot be used directly, only derived from.

Cannot define a pointer or reference to a reference *Compiler error*

It is illegal to have a pointer to a reference or a reference to a reference.

Cannot find *class*::*class*(*class* &) to copy a vector *Compiler error*

When a C++ class *class1* contains a vector (array) of class *class2*, and you want to construct an object of type *class1* from another object of type *class1*, there must be a constructor `class2::class2(class2&)` so that the elements of the vector can be constructed. This constructor, called a *copy constructor*, takes just one parameter (a reference to its class).

Usually the compiler supplies a copy constructor automatically. However, if you have defined a constructor for class *class2* that has a parameter of type *class2&* and has additional parameters with default values, the copy constructor cannot be created by the compiler. (This is because `class2::class2(class2&)` and `class2::class2(class2&, int = 1)` cannot be distinguished.) You must redefine this constructor so that not all parameters have default values. You can then define a copy constructor or let the compiler create one.

Cannot find *class*::operator=(*class*&) to copy a vector *Compiler error*

When a C++ class *class1* contains a vector (array) of class *class2*, and you want to copy a class of type *class1*, there must be an assignment operator `class2::operator=(class2&)` so that the elements of the vector can be copied. Usually the compiler supplies such an operator automatically. However, if you have defined an **operator=** for class *class2*, but not one that takes a parameter of type *class2&*, the compiler will not supply it automatically—you must supply one.

Cannot find default constructor to initialize array element of type *class* *Compiler error*

When declaring an array of a class that has constructors, you must either explicitly initialize every element of the array, or the class must have a default constructor (it will be used to initialize the array elements that don't have explicit initializers). The compiler will define a default constructor for a class unless you have defined any constructors for the class.

Cannot find default constructor to initialize base class *class* *Compiler error*

Whenever a C++ derived class *class2* is constructed, each base class *class1* must first be constructed. If the constructor for *class2* does not specify a constructor for *class1* (as part of *class2*s header), there must be a constructor `class1::class1()` for the base class. This constructor without parameters is called the default constructor. The compiler will supply a default constructor automatically unless you have defined any constructor for class *class1*; in that case, the compiler will not supply the default constructor automatically—you must supply one.

Cannot find default *constructor* to initialize member *identifier* *Compiler error*

When a C++ class *class1* contains a member of class *class2*, and you want to construct an object of type *class1* but not from another object of type *class1*, there must be a constructor `class2::class2()` so that the member can be constructed. This constructor without parameters is called the *default constructor*. The compiler supplies a default constructor automatically unless you've defined a constructor for class *class2*. If you have, the compiler won't supply the default constructor automatically—you must supply one.

Cannot find MAKE.EXE *MAKE error*

The MAKE command-line tool cannot be found. Be sure that MAKE.EXE is in either the current directory or in a directory contained in your directory path.

Cannot generate COM file: data below initial CS:IP defined *Linker error*

This error results from trying to generate data or code below the starting address (usually 100) of a .COM file. Be sure that the starting address is set to 100 by using the (ORG 100H) instruction. This error message should not occur for programs written in a high-level language. If it does, ensure that the correct startup (C0*x*) object module is being linked in.

Cannot generate COM file: invalid initial entry point address *Linker error*

You used the **/Tdc** or **/t** option, but the program starting address is not equal to 100H, which is required with .COM files.

Cannot generate COM file: program exceeds 64K *Linker error*

You used the **/Tdc** or **/t** option, but the total program size exceeds the .COM file limit.

Cannot generate COM file: segment-relocatable items present *Linker error*

You used the **/Tdc** or **/t** option, but the program contains segment-relative fixups, which are not allowed with .COM files.

Cannot generate COM file: stack segment present *Linker error*

You used the **/Tdc** or **/t** option, but the program declares a stack segment, which is not allowed with .COM files.

Cannot generate *function* from template function *template* *Compiler error*

A call to a template function was found, but a matching template function cannot be generated from the function template.

Cannot have a non-inline function in a local class *Compiler error*
Cannot have a static data member in a local class *Compiler error*

All members of classes declared local to a function must be entirely defined in the class definition. This means that such local classes cannot contain any static data members, and all of their member functions must have bodies defined within the class definition.

Cannot have multiple paths for implicit rule *MAKE error*

You can have only one path for each of the extensions in an implicit rule; for example, `{path}.c.obj`. Multiple path lists are allowed only for dependents in an explicit rule.

Cannot have path list for target *MAKE error*

You can only specify a path list for dependents of an explicit rule. For example:

```
{path1;path2}prog.exe: prog.obj   # Invalid
 prog.exe: {path1;path2}prog.obj   # Valid
```

Cannot initialize a class member here *Compiler error*

Individual members of **structs**, **unions**, and C++ **class**es cannot have initializers. A **struct** or **union** can be initialized as a whole using initializers inside braces. A C++ **class** can be initialized only by the use of a constructor.

Cannot initialize *type1* with *type2* *Compiler error*

You are attempting to initialize an object of type *type1* with a value of type *type2*, which is not allowed. The rules for initialization are essentially the same as for assignment.

Cannot modify a const object *Compiler error*

This indicates an illegal operation on an object declared to be **const**, such as an assignment to the object.

Cannot overload 'main' *Compiler error*

main is the only function that cannot be overloaded.

***function* cannot return a value** *Compiler error*

A function with a return type **void** contains a **return** statement that returns a value; for example, an **int**.

***identifier* cannot start a parameter declaration** *Compiler error*

An undefined 'identifier' was found at the start of an argument in a function declarator. This error usually occurs because the wrong header file was used. If that isn't the cause, check to see if the type name is misspelled or if the type declaration is missing.

***identifier* cannot start an argument declaration** *Compiler error*

Undefined *identifier* found at the start of an argument in a function declarator. This error usually occurs because the wrong header file was used. If that isn't the cause, check to see if the type name is misspelled or if the type declaration is missing.

Cannot take address of *main* *Compiler error*

In C++ it is illegal to take the address of the main function.

Cannot throw *type* — ambiguous base class *base* *Compiler error*

It is not legal to throw a class that contains more than one copy of a (nonvirtual) base class.

Cannot write a string option *MAKE error*

the **–W** MAKE option writes a character option to MAKE.EXE. If there's any string option (for example, **–D**xxxx="My_foo" or **–U**xxxxx), this error message is generated.

Cannot write GRPDEF list, extended dictionary aborted *Librarian error*

The librarian cannot write the extended dictionary to the tail end of the library file. This usually indicates lack of space on the disk.

Can't grow LE/LIDATA record buffer *Librarian error*

Command-line error. See the **out of memory reading LE/LIDATA record from object module** message.

Case bypasses initialization of a local variable *Compiler error*

In C++ it is illegal to bypass the initialization of a local variable in any way. In this instance, there is a **case** label that can transfer control past this local variable.

Case outside of switch *Compiler error*

The compiler encountered a **case** statement outside a **switch** statement. This is often caused by mismatched braces.

Case statement missing : *Compiler error*

A **case** statement must have a constant expression followed by a colon. The expression in the **case** statement either is missing a colon or has an extra symbol before the colon.

***catch* expected** *Compiler error*

In a C++ program, a *try* block must be followed by at least one *catch* block.

Character constant must be one or two characters long *Compiler error*

Character constants can be only one or two characters long.

Character constant too long *MAKE error*

A char constant in an expression is too long.

Circular dependency exists in makefile *MAKE error*

The makefile indicates that a file needs to be up-to-date *before* it can be built. Take, for example, the explicit rules:

```
filea:  fileb
fileb:  filec
filec:  filea
```

This implies that file*a* depends on file*b*, which depends on file*c*, and file*c* depends on file*a*. This is illegal because a file cannot depend on itself, indirectly or directly.

Class *class* may not contain pure functions *Compiler error*

The class being declared cannot be abstract; it therefore cannot contain any pure functions.

Class member *member* declared outside its class *Compiler error*

C++ class member functions can be declared inside the class declaration only. Unlike nonmember functions, they cannot be declared multiple times or at other locations.

Code has no effect *Compiler warning*

The compiler encountered a statement with operators that have no effect. For example, the statement

```
a + b;
```

has no effect on either variable. The operation is unnecessary and probably indicates a bug in your file.

Colon expected *MAKE error*

Your implicit rule is missing a colon at the end.

```
.c.obj:     # Correct
.c.obj      # Incorrect
```

Command arguments too long *MAKE error*

The arguments to a command were more than the 511-character limit imposed by DOS.

Command syntax error *MAKE error*

This message occurs if

- The first rule line of the makefile contained any leading whitespace.
- An implicit rule did not consist of *.ext.ext:*.
- An explicit rule did not contain a name before the : character.
- A macro definition did not contain a name before the = character.

Command too long *MAKE error*

The length of a command has exceeded 512 characters. You might want to use a response file.

Common segment exceeds 64K *Linker error*

The program had more than 64K of near uninitialized data. Try declaring some uninitialized data as **far**.

Compiler could not generate copy constructor for class *class* *Compiler error*

The compiler cannot generate a needed copy constructor due to language rules.

Compiler could not generate default constructor for class *class* *Compiler error*

The compiler cannot generate a needed default constructor due to language rules.

Compiler could not generate operator= for class *class* *Compiler error*

The compiler cannot generate a needed assignment operator due to language rules.

Compiler table limit exceeded *Compiler error*

One of the compiler's internal tables overflowed. This usually means that the module being compiled contains too many function bodies. Making more memory available to the compiler will not help with such a limitation; simplifying the file being compiled is usually the only remedy.

Compound statement missing } *Compiler error*

The compiler reached the end of the source file and found no closing brace. This is often caused by mismatched braces.

Condition is always false *Compiler warning*
Condition is always true *Compiler warning*

The compiler encountered a comparison of values where the result is always true or false. For example:

```
void proc(unsigned x) {
   if (x >= 0) {       /* always 'true' */
   }
}
```

Conflicting type modifiers *Compiler error*

This occurs when a declaration is given that includes, for example, both **near** and **far** keywords on the same pointer. Only one addressing modifier can be given for a single pointer, and only one language modifier (**cdecl**, **pascal**, or **interrupt**) can be given for a function.

symbol* conflicts with module *module *Linker warning*

This indicates an inconsistency in the definition of *symbol*; TLINK found one virtual function and one common definition with the same name.

Constant expression required *Compiler error*

Arrays must be declared with constant size. This error is commonly caused by misspelling a **#define** constant.

Constant is long *Compiler warning*

The compiler encountered either a decimal constant greater than 32767 or an octal (or hexadecimal) constant greater than 65535 without a letter *l* or *L* following it. The constant is treated as a **long**.

Constant member *member* in class without constructors *Compiler error*

A class that contains constant members must have at least one user-defined constructor; otherwise, there would be no way to initialize such members.

Constant member *member* is not initialized *Compiler warning*

This C++ class contains a constant member *member*, which does not have an initialization. Constant members can only be initialized; they cannot be assigned to.

Constant out of range in comparison *Compiler warning*

Your source file includes a comparison involving a constant subexpression that was outside the range allowed by the other subexpression's type. For example, comparing an **unsigned** quantity to –1 makes no sense. To get an **unsigned** constant greater than 32767 (in decimal), you should either cast the constant to **unsigned** or append a letter *u* or *U* to the constant.

When this message is issued, the compiler still generates code to do the comparison. If this code ends up always giving the same result, such as comparing a **char** expression to 4000, the code still performs the test.

Constant variable *variable* must be initialized *Compiler error*

This C++ object is declared **const**, but is not initialized. Because no value can be assigned to it, it must be initialized at the point of declaration.

Constructor cannot be declared const or volatile *Compiler error*

A constructor has been declared as **const** and/or **volatile**, and this is not allowed.

Constructor cannot have a return type specification *Compiler error*

C++ constructors have an implicit return type used by the compiler, but you cannot declare a return type or return a value.

Conversion may lose significant digits *Compiler warning*

For an assignment operator or some other circumstance, your source file requires a conversion from **long** or **unsigned long** to **int**, or **unsigned int** type. Because **int** type and **long** type variables don't have the same size, this kind of conversion can alter the behavior of a program.

Conversion of near pointer not allowed *Compiler error*

A near pointer cannot be converted to a far pointer in the expression evaluation box when a program is not currently running. This is because the conversion needs the current value of DS in the user program, which doesn't exist.

Conversion operator cannot have a return type specification *Compiler error*

This C++ type conversion member function specifies a return type different from the type itself. A declaration for conversion function **operator** cannot specify any return type.

Conversion to *type* will fail for members of virtual base *class* *Compiler error*

This warning can occur when a member pointer (whose class contains virtual bases) is cast to another member-pointer type and you use the **–Vv** option. This error indicates that if the member pointer being cast happens to point (at the time of the cast) to a member of ***class***, the conversion cannot be completed, and the result of the cast will be a NULL member pointer.

Could not allocate memory for per module data *Librarian error*

The librarian has run out of memory.

Could not create list file *filename* *Librarian error*

The librarian could not create a list file for the library. This could be due to lack of disk space.

Could not find a match for *argument(s)* *Compiler error*

No C++ function could be found with parameters matching the supplied arguments.

Could not find file *filename* *Compiler error*

The compiler is unable to find the file supplied on the command line.

Could not get procedure address from DLL *filename* *Linker error*

The linker was not able to get a procedure from the specified DLL. Check to make sure that you have the correct DLL version.

Could not load DLL *filename* *Linker error*

The linker was not able to load the specified DLL. Check to make sure the DLL is on your path.

Could not write output *Librarian error*
The librarian could not write the output file.

Couldn't alloc memory for per module data *Librarian error*
The librarian has run out of memory.

***filename* couldn't be created, original won't be changed** *Librarian warning*
An attempt has been made to extract an object ('*' action) but the librarian cannot create the object file to extract the module into. Either the object already exists and is read only, or the disk is full.

Couldn't get LE/LIDATA record buffer *Librarian error*
Command-line error. See the **Out of memory reading LE/LIDATA record from object module** message.

Couldn't get procedure address from dll *dll* *Linker error*
The linker wasn't able to get a procedure from the specified DLL. Check to make sure you have the correct version of the DLL.

Couldn't load dll *dll* *Linker error*
The linker wasn't able to load the specified DLL. Check to make sure the DLL is on your path.

Cycle in include files: *filename* *MAKE error*
This error message is issued if a makefile includes itself in the make script.

Debug info switch ignored for .COM files *Linker warning*
Borland C++ does not include debug information for .COM files.

Debug information enabled, but no debug information found in OBJs *Linker warning*
No part of the application was compiled with debug information, but you requested that debug information be turned on in the link.

Debug information in module *module* will be ignored *Linker warning*
Object files compiled with debug information now have a version record. The major version of this record is higher than what TLINK currently supports and TLINK did not generate debug information for the module in question.

Debugging information overflow; try fewer modules with debug info *Linker error*
Too many modules containing debugging information are included in the link. Recompile your program with fewer modules marked for debug information.

Declaration does not specify a tag or an identifier *Compiler error*
This declaration doesn't declare anything. This might be a **struct** or **union** without a tag or a variable in the declaration. C++ requires that something be declared.

Declaration is not allowed here *Compiler error*
Declarations cannot be used as the control statement for **while**, **for**, **do**, **if**, or **switch** statements.

Declaration missing ; *Compiler error*
Your source file contained a declaration that was not followed by a semicolon.

Declaration syntax error *Compiler error*
Your source file contained a declaration that was missing some symbol or had some extra symbol added to it.

Declaration terminated incorrectly *Compiler error*
A declaration has an extra or incorrect termination symbol, such as a semicolon placed after a function body. A C++ member function declared in a class with a semicolon between the header and the opening left brace also generates this error.

Declaration was expected *Compiler error*

A declaration was expected here but not found. This is usually caused by a missing delimiter such as a comma, semicolon, right parenthesis, or right brace.

Declare operator delete (void*) or (void*, size_t) *Compiler error*

Declare the operator **delete** with a single **void*** parameter or with a second parameter of type **size_t**. If you use the second version, it will be used in preference to the first version. The global operator **delete** can be declared using the single-parameter form only.

Declare operator delete[] (void*) or (void*, size_t) *Compiler error*

Declare the operator delete with one of the following:

- A single void* parameter
- A second parameter of type size_t

If you use the second version, it will be used in preference to the first version. The global operator delete can be declared using the single-parameter form only.

Declare type *type* prior to use in prototype *Compiler warning*

When a function prototype refers to a structure type that has not previously been declared, the declaration inside the prototype is not the same as a declaration outside the prototype. For example,

```
int func(struct s *ps);
struct s { /* ... */ };
```

Because there is no **struct** *s* in scope at the prototype for *func*, the type of parameter *ps* is a pointer to undefined **struct** *s*, and is not the same as the **struct** *s* that is later declared. This results in warning and error messages about incompatible types, which would be mysterious without this warning message. To fix the problem, you can move the declaration for **struct** *s* ahead of any prototype that references it, or add the incomplete type declaration `struct s;` ahead of any prototype that references struct *s*. If the function parameter is a **struct**, rather than a pointer to **struct**, the incomplete declaration is not sufficient; you must then place the struct declaration ahead of the prototype.

Default argument value redeclared *Compiler error*

When a parameter of a C++ function is declared to have a default value, this value can't be changed, redeclared, or omitted in any other declaration for the same function.

Default argument value redeclared for parameter *parameter* *Compiler error*

When a parameter of a C++ function is declared to have a default value, this value cannot be changed, redeclared, or omitted in any other declaration for the same function.

Default expression may not use local variables *Compiler error*

A default argument expression is not allowed to use any local variables or other parameters.

Default outside of switch *Compiler error*

The compiler encountered a **default** statement outside a **switch** statement. This is most commonly caused by mismatched braces.

Default value missing *Compiler error*

When a C++ function declares a parameter with a default value, all of the following parameters must also have default values. In this declaration, a parameter with a default value was followed by a parameter without a default value.

Default value missing following parameter *parameter* *Compiler error*

All parameters following the first parameter with a default value must also have defaults specified.

Define directive needs an identifier *Compiler error*

The first non-whitespace character after a **#define** must be an identifier. The compiler found some other character.

***symbol* defined in module *module* is duplicated** *Linker error*

There is a conflict between two symbols (either public or communal). This usually means that a symbol is defined in two modules. An error occurs if both are encountered in the .OBJ file(s), because TLINK doesn't know which is valid. A warning results if TLINK finds one of the duplicated symbols in a library and finds the other in an .OBJ file; in this case, TLINK uses the one in the .OBJ file.

Delete array size missing] *Compiler error*

The array specifier in an operator is missing a right bracket.

Destructor cannot be declared const or volatile *Compiler error*

A destructor has been declared as **const** and/or **volatile**, and this is not allowed.

Destructor cannot have a return type specification *Compiler error*

It is illegal to specify the return type for a destructor.

Destructor for *class* is not accessible *Compiler error*

The destructor for this C++ class is **protected** or **private**, and cannot be accessed here to destroy the class. If a class destructor is **private**, the class cannot be destroyed, and thus can never be used. This is probably an error. A **protected** destructor can be accessed only from derived classes. This is a useful way to ensure that no instance of a base class is ever created, but only classes derived from it.

Destructor for *class* required in conditional expression *Compiler error*

If the compiler must create a temporary local variable in a conditional expression, it has no good place to call the destructor, because the variable might or might not have been initialized. The temporary variable can be explicitly created, as with `classname (val, val)`, or implicitly created by some other code. Recast your code to eliminate this temporary value.

Destructor name must match the class name *Compiler error*

In a C++ class, the tilde (~) introduces a declaration for the class destructor. The name of the destructor must be the same as the class name. In your source file, the tilde (~) preceded some other name.

Divide error *Run-time error*

You've tried to divide an integer by zero. You can trap this error with the **signal** function. Otherwise, Borland C++ calls **abort** and your program terminates.

Division by zero *Compiler error*

Your source file contained a division or remainder operator in a constant expression with a zero divisor.

Division by zero *Compiler warning*

A division or remainder operator expression had a literal zero as a divisor.

Division by zero *MAKE error*

A division or remainder operator in an **!if** statement has a zero divisor.

do statement must have while *Compiler error*

Your source file contained a **do** statement that was missing the closing **while** keyword.

***filename* does not exist – don't know how to make it** *MAKE error*

There is a nonexistent file name in the build sequence, and no rule exists that would allow the file name to be built.

DOS error, ax = *number* *Linker error*

This error occurs if a DOS call returned an unexpected error. The *ax* value printed is the resulting error code. This could indicate a TLINK internal error or a DOS error. The only DOS calls TLINK makes in which this error could occur are read, write, seek, and close.

do-while statement missing (*Compiler error*

In a **do** statement, the compiler found no left parenthesis after the **while** keyword.

do-while statement missing) *Compiler error*

In a **do** statement, the compiler found no right parenthesis after the test expression.

do-while statement missing ; *Compiler error*

In a **do** statement test expression, the compiler found no semicolon after the right parenthesis.

Duplicate case *Compiler error*

Each **case** of a **switch** statement must have a unique constant expression value.

Duplicate Handler for *type1*, already had *type2* *Compiler error*

It's illegal to specify two handlers for the same type.

Duplicate ordinal for exports: *string* (*ordval1*) and *string* (*ordval2*) *Linker error*

Two exports have been found for the same symbol, but with differing ordinal values. You must use the same ordinal value or remove one of the exports.

Empty LEDATA record in module *module* *Linker warning*

This warning can happen if the translator emits a data record containing data. If this should happen, report the occurrence to the translator vendor; there should be no bad side effects from the record.

Enum syntax error *Compiler error*

An **enum** declaration did not contain a properly formed list of identifiers.

Error changing file buffer size *Librarian error*

The librarian is attempting to adjust the size of a buffer used while reading or writing a file, but there is not enough memory. It is likely that quite a bit of system memory will have to be freed up to resolve this error.

Error directive: *message* *Compiler error*

The text of the #**error** directive being processed in the source file is displayed.

Error directive: *message* *MAKE error*

MAKE has processed an #**error** directive in the source file, and the text of the directive is displayed in the message.

Error opening *filename* *Librarian error*

librarian cannot open the specified file for some reason.

Error opening *filename* for output *Librarian error*

librarian cannot open the specified file for output. This is usually due to lack of disk space for the target library, or a listing file. This error occurs when the target file exists but is marked as a read-only file.

Error renaming *filename* to *filename* *Librarian error*

The librarian builds a library into a temporary file and then renames the temporary file to the target library file name. If there is an error, usually due to lack of disk space, this message is posted.

Error writing output file *Compiler error*

A DOS error that prevents Borland C++ from writing an .OBJ, .EXE, or temporary file. Check the output directory and make sure that this is a valid directory. Also check that there is enough free disk space.

_ _*except* or _ _*finally* expected following _ _*try* *Compiler error*
In C, a _ _*try* block must be followed by an _ _*except* or _ _*finally* handler block.

Exception handling variable may not be used here *Compiler error*
An attempt has been made to use one of the exception handling values that are restricted to particular exception handling constructs, such as GetExceptionCode().

Exception specification not allowed here *Compiler error*
Function pointer type declarations are not allowed to contain exception specifications.

Explicit stacks are ignored for PE images *Linker warning*
Windows 32-bit applications are PE format applications, which do not have explicit stacks. The stack segment will be linked into the image, but it will not be used as the application stack. Instead, the stack size parameter will be used to set the stack size, and the operating system will allocate a stack for the application.

Export *symbol* is duplicated *Linker warning*
This warning occurs if two different symbols with the same name are exported by the use of **_export**. The linker cannot determine which definition it should export, and therefore uses the first symbol.

Expression expected *Compiler error*
An expression was expected here, but the current symbol cannot begin an expression. This message can occur where the controlling expression of an **if** or **while** clause is expected or where a variable is being initialized. It is often due to an accidentally inserted or deleted symbol in the source code.

Expression of scalar type expected *Compiler error*
The not (**!**), increment (**++**), and decrement (**- -**) operators require an expression of scalar type. Only types **char**, **short**, **int**, **long**, **enum**, **float**, **double**, **long double**, and pointer types are allowed.

Expression syntax *Compiler error*
This is a catchall error message when the compiler parses an expression and encounters a serious error. This is most commonly caused by two consecutive operators, mismatched or missing parentheses, or a missing semicolon on the previous statement.

Expression syntax error in !if statement *MAKE error*
The expression in an **!if** statement is badly formed—it contains a mismatched parenthesis, an extra or missing operator, or a missing or extra constant.

***reason* – extended dictionary not created** *Librarian warning*
The librarian could not produce the extended dictionary because of the *reason* given in the warning message.

Extended dictionary not found in library *library*, extended dictionaries ignored *Linker warning*

The **/E** option for TLINK requires that all libraries in the link have extended dictionaries. When a library without an extended dictionary is encountered during a link operation in which the **/E** option is specified, the linker abandons extended dictionary processing and proceeds to link with a default link.

Extern variable cannot be initialized *Compiler error*
The storage class **extern** applied to a variable means that the variable is being declared but not defined here—no storage is being allocated for it. Therefore, you can't initialize the variable as part of the declaration.

Extern *symbol* was not qualified with _ _import in module *module* *Linker warning*
Windows 32-bit applications which reference imported symbols need to make indirections to get to the data. For calls, this is handled automatically by the linker. For references to imported DATA, the compiler must generate an indirection, or the application will function incorrectly. The compiler knows to generate the indirection when the symbol is qualified with

_ _import. If the linker sees a segment relative reference to a symbol that is imported, and if the symbol was not qualified with _ _import, you will get this message.

Extra argument in template class name *template* *Compiler error*
A template class name specified too many actual values for its formal parameters.

Extra parameter in call *Compiler error*
A call to a function, via a pointer defined with a prototype, had too many arguments given.

Extra parameter in call to *function* *Compiler error*
A call to the named function (which was defined with a prototype) had too many arguments given in the call.

Failed to locate DPMI server (DPMI16BI.OVL) *Compiler error*
Failed to locate protected mode loader (DPMILOAD.EXE) *Compiler error*
Make sure that DPMI16BI.OVL and DPMILOAD.EXE are somewhere on your path or in the same directory as the protected mode command-line tool you were attempting to use.

Failed read from *filename* *Linker error*
The linker was unable to read from the file.

Failed write to *filename* *Linker error*
The Linker was unable to write to the file.

_ _far16 may only be used with _ _pascal or _ _cdecl *Compiler error*
When you use **_ _far16** to make calls to functions or reference data in a 16-bit DLL, such functions and data can be modified only by **_ _pascal** or **_ _cdecl**.}]

File must contain at least one external declaration *Compiler error*
This compilation unit was logically empty, containing no external declarations. ANSI C and C++ require that something be declared in the compilation unit.

Filename too long *Compiler error*
The file name given in an **#include** directive was too long for the compiler to process. Path names must be no longer than 260 characters.

File name too long *MAKE error*
The path name in an **!include** directive overflowed MAKE's internal buffer (512 bytes).

***filename* file not found** *Librarian warning*
The command-line librarian attempted to add a nonexisting object but created the library anyway.

***filename* file not found** *Librarian error*
The IDE creates the library by first removing the existing library and then rebuilding. If any objects do not exist, the library is considered incomplete and TLIB generates this error. If the IDE reports that an object does not exist, either the source module has not been compiled or there were errors during compilation. Rebuilding your project should resolve the problem or indicate where the errors have occurred.

***filename* (*linenum*): File read error** *Linker error*
A DOS error occurred while TLINK read the module definition file. This usually means that a premature end of file occurred.

Fixup to zero length segment in module *module* *Linker error*
A reference has been made past the end of an image segment. This reference would end up accessing an invalid address, and has been flagged as an error.

Fixup overflow at *address*, target = *address* *Linker warning*
These messages indicate an incorrect data or code reference in an object file that TLINK must fix up at link time.

The cause is often a mismatch of memory models. A **near** call to a function in a different code segment is the most likely cause. These errors can also result if you generate a **near** call to a data variable or a data reference to a function. In either case the symbol named as the *target* in the error message is the referenced variable or function. The reference is in the named module, so look in the source file of that module for the offending reference.

In an assembly language program, a fixup overflow frequently occurs if you have declared an external variable within a segment definition, but this variable actually exists in a different segment.

If this technique does not identify the cause of the failure, or if you are programming in assembly language or in a high-level language other than Borland C++, there might be other possible causes for this message. Even in Borland C++, this message could be generated if you are using different segment or group names than the default values for a given memory model.

Fixup to zero length segment in module *module* *Linker error*

This error usually occurs if you make a reference to a segment that doesn't contain any data. If the segment isn't grouped with other segments, the result is a zero-length physical segment, which cannot exist. The linker therefore cannot make a reference to it.

Floating point error: Divide by 0. *Run-time error*
Floating point error: Domain. *Run-time error*
Floating point error: Overflow. *Run-time error*

These fatal errors result from a floating-point operation for which the result is not finite.

- "Divide by 0" means the result is +INF or –INF exactly, such as 1.0/0.0.
- "Domain" means the result is NAN (not a number).
- "Overflow" means the result is +INF (infinity) or –INF with complete loss of precision, such as assigning 1e200*1e200 to a **double**.

Floating point error: Partial loss of precision. *Run-time error*
Floating point error: Underflow. *Run-time error*

These exceptions are masked by default, and the error messages do not occur. Underflows are converted to zero and losses of precision are ignored. They can be unmasked by calling *_control87*.

Floating point error: Stack fault. *Run-time error*

The floating-point stack has been overrun. This error does not normally occur and might be due to assembly code using too many registers or to a misdeclaration of a floating-point function.

These floating-point errors can be avoided by masking the exception so that it doesn't occur, or by catching the exception with *signal*. See the functions *_control87* and *signal* for details.

for statement missing (*Compiler error*

In a **for** statement, the compiler found no left parenthesis after the **for** keyword.

for statement missing) *Compiler error*

In a **for** statement, the compiler found no right parenthesis after the control expressions.

for statement missing ; *Compiler error*

In a **for** statement, the compiler found no semicolon after one of the expressions.

Friends must be functions or classes *Compiler error*

A **friend** of a C++ class must be a function or another class.

Function call missing) *Compiler error*

The function call argument list had some sort of syntax error, such as a missing or mismatched right parenthesis.

Function calls not supported *Compiler error*

In integrated debugger expression evaluation, calls to functions (including implicit conversion functions, constructors, destructors, overloaded operators, and inline functions) are not supported.

Function defined inline after use as extern *Compiler error*

Functions cannot become inline after they have already been used. Either move the inline definition forward in the file or delete it entirely.

Function definition cannot be a Typedef'ed declaration *Compiler error*

In ANSI C a function body cannot be defined using a typedef with a function Type.

Function *function* cannot be static *Compiler error*

Only ordinary member functions and the operators **new** and **delete** can be declared static. Constructors, destructors, and other operators must not be static.

Function *function* should have a prototype *Compiler error*

A function was called with no prototype in scope.

In C, `int foo();` is not a prototype, but `int foo(int);` is, and so is `int foo(void);`. In C++, `int foo();` is a prototype, and is the same as `int foo(void);`. In C, prototypes are *recommended* for all functions. In C++, prototypes are *required* for all functions. In C and C++, a function definition (a function header with its body) serves as a prototype if it appears before any other mention of the function.

Function should return a value *Compiler warning*

This function was declared (perhaps implicitly) to return a value. A **return** statement was found without a return value or the end of the function was reached without a **return** statement being found. Either return a value or declare the function as **void**.

Functions *function1* and *function2* both use the same dispatch number *Compiler error*

This error is the result of a dynamically dispatched virtual table (DDVT) problem. When you override a dynamically dispatchable function in a derived class, use the same dispatch index. Each function within the same class hierarchy must use a different dispatch index.

Functions cannot return arrays or functions *Compiler error*

A function cannot return an array or a function. Only pointers or references to arrays or functions can be returned.

Functions containing local destructors are not expanded inline in function *function* *Compiler warning*

You've created an inline function for which Borland C++ turns off inlining. You can ignore this warning if you like; the function will be generated out of line.

Functions containing *reserved word* are not expanded inline *Compiler warning*

Functions containing any of the reserved words **do, for, while, goto, switch, break, continue**, and **case** cannot be expanded inline, even when specified as **inline**. The function is still perfectly legal, but will be treated as an ordinary static (not global) function.

Functions may not be part of a struct or union *Compiler error*

This C **struct** or **union** field was declared to be of type function rather than pointer to function. Functions as fields are allowed only in C++.

General error *Linker error*
General error in library file *filename* in module *module* near module file offset 0x*yyyyyyyy*. *Linker error*
General error in module *module* near module file offset 0x*yyyyyyyy* *Linker error*

The linker gives as much information as possible about what processing was happening at the time of the unknown fatal error. Call Technical Support with information about .OBJ or .LIB files.

Global anonymous union not static *Compiler error*
In C++, a global anonymous union at the file level must be static.

Goto bypasses initialization of a local variable *Compiler error*
In C++ it is illegal to bypass the initialization of a local variable in any way. You'll get this error when there is a **goto** that tries to transfer control past this local variable.

Goto into an exception handler is not allowed *Compiler error*
It's illegal to jump into a try block or an exception handler that's attached to a try block.

Goto statement missing label *Compiler error*
The **goto** keyword must be followed by an identifier.

Group *group* exceeds 64K *Linker error*
A group exceeded 64K bytes when the segments of the group were combined.

Group overflowed maximum size: *group* *Compiler error*
The total size of the segments in a group (for example, DGROUP) exceeded 64K.

Group *group1* overlaps group *group2* *Linker warning*
This means that TLINK has encountered nested groups. This warning occurs only when overlays are used.

Handler for *type1* hidden by previous handler for *type2* *Compiler warning*
This warning is issued when a handler for a type *D* that is derived from type *B* is specified after a handler for *B*, since the handler for *D* will never be invoked.

***specifier* has already been included** *Compiler error*
This type specifier occurs more than once in this declaration. Delete or change one of the occurrences.

Hexadecimal value contains more than 3 digits *Compiler warning*
Under older versions of C, a hexadecimal escape sequence could contain no more than three digits. The ANSI standard allows any number of digits to appear as long as the value fits in a byte. This warning results when you have a long hexadecimal escape sequence with many leading zero digits (such as "\x00045"). Older versions of C would interpret such a string differently.

function1* hides virtual function *function2 *Compiler warning*
A virtual function in a base class is usually overridden by a declaration in a derived class. In this case, a declaration with the same name but different argument types makes the virtual functions inaccessible to further derived classes.

Identifier expected *Compiler error*
An identifier was expected here, but not found. In C, this error occurs in a list of parameters in an old-style function header, after the reserved words **struct** or **union** when the braces are not present, and as the name of a member in a structure or union (except for bit fields of width 0). In C++, an identifier is also expected in a list of base classes from which another class is derived, following a double colon (**::**), and after the reserved word **operator** when no operator symbol is present.

Identifier *identifier* cannot have a type qualifier *Compiler error*
A C++ qualifier *class::identifier* cannot be applied here. A qualifier is not allowed on **typedef** names, on function declarations (except definitions at the file level), on local variables or parameters of functions, or on a class member except to use its own class as a qualifier (which is redundant but legal).

If statement missing (*Compiler error*
In an **if** statement, the compiler found no left parenthesis after the **if** keyword.

If statement missing) *Compiler error*
In an **if** statement, the compiler found no right parenthesis after the test expression.

If statement too long *MAKE error*
Ifdef statement too long *MAKE error*
Ifndef statement too long *MAKE error*

An **If**, **Ifdef**, or **Ifndef** statement has exceeded 4,096 characters.

Ignored *module*, path is too long *Librarian warning*

The path to a specified .OBJ or .LIB file is greater than 64 characters. The max path to a file for librarian is 64 characters.

Illegal ACBP byte in SEGDEF *Linker error*

This is usually a translator error.

Illegal character *character* (0x*value*) *Compiler error*

The compiler encountered some invalid character in the input file. The hexadecimal value of the offending character is printed. This can also be caused by extra parameters passed to a function macro.

Illegal character in constant expression *expression* *MAKE error*

MAKE encountered a character not allowed in a constant expression. If the character is a letter, this probably indicates a misspelled identifier.

Illegal component to GRPDEF *Linker error*

This is usually a translator error.

Illegal group definition: *group* *Linker error*

This error is caused by a malformed GRPDEF record in an .OBJ file. This could result from custom-built .OBJ files or a bug in the translator used to generate the .OBJ file. If this occurs in a file created by Borland C++, recompile the file. If the error persists, contact Borland Technical Support.

Illegal initialization *Compiler error*

In C, initializations must be either a constant expression, or else the address of a global **extern** or **static** variable plus or minus a constant.

Illegal octal digit *Compiler error*

An octal constant containing a digit of 8 or 9 was found.

Illegal parameter to _ _emit_ _ *Compiler error*

You supplied an argument to _ _*emit*_ _ that is not a constant or an address.

Illegal pointer subtraction *Compiler error*

This is caused by attempting to subtract a pointer from a nonpointer.

Illegal structure operation *Compiler error*

In C or C++, structures can be used with dot (.), address-of (&), or assignment (=) operators, or can be passed to or from functions as parameters. In C or C++, structures can also be used with overloaded operators. The compiler encountered a structure being used with some other operator.

Illegal to take address of bit field *Compiler error*

It is not legal to take the address of a bit field, although you can take the address of other kinds of fields.

Illegal use of floating point *Compiler error*

Floating-point operands are not allowed in shift, bitwise Boolean, indirection (*), or certain other operators. The compiler found a floating-point operand with one of these prohibited operators.

Illegal use of member pointer *Compiler error*

Pointers to class members can be used only with assignment, comparison, the **.***, **->***, **?:**, **&&**, and **||** operators, or passed as arguments to functions. The compiler has encountered a member pointer being used with a different operator.

Illegal use of pointer *Compiler error*

Pointers can be used only with addition, subtraction, assignment, comparison, indirection (*) or arrow (->) operators. Your source file used a pointer with some other operator.

Ill-formed pragma *Compiler warning*

A pragma does not match one of the pragmas expected by the Borland C++ compiler.

Image base address must be a multiple of 0x10000 *Linker error*

Based images must be aligned on 64k boundaries.

Images fixed at specific addresses typically will not run under Win32s *Linker warning*

Windows 32s loads all applications in a single address space. It's impossible to predict where you application is going to be loaded, because other 32-bit applications might have been loaded before yours. Fixed images must be loaded at their requested base address or the loader will fail to run them.

Implicit conversion of *type1* to *type2* not allowed *Compiler error*

When a member function of a class is called using a pointer to a derived class, the pointer value must be implicitly converted to point to the appropriate base class. In this case, such an implicit conversion is illegal.

Improper use of typedef *identifier* *Compiler error*

Your source file used a **typedef** symbol where a variable should appear in an expression. Check for the declaration of the symbol and possible misspellings.

Include files nested too deep *Compiler error*

When the compiler detects that header files are nested more than 1,000 levels deep, it assumes that the header file is recursive, and stops compilation with this (fatal) error.

***filename* (*linenum*): Incompatible attribute** *Linker error*

The linker encountered incompatible segment attributes in a CODE or DATA statement. For instance, both PRELOAD and LOADONCALL can't be attributes for the same segment.

Incompatible type conversion *Compiler error*

The cast requested can't be done. Check the types.

Incorrect command-line argument: *argument* *MAKE error*

You've used incorrect command-line arguments.

Incorrect command-line option: *option* *Compiler error*

The compiler did not recognize the command-line parameter as legal.

Incorrect configuration file option: *option* *Compiler error*

The compiler did not recognize the configuration file parameter as legal; check for a preceding hyphen (–).

Incorrect number format *Compiler error*

The compiler encountered a decimal point in a hexadecimal number.

Incorrect use of default *Compiler error*

The compiler found no colon after the **default** keyword in a **case** statement.

Incorrect version of RLINK32.DLL *Linker error*

The RLINK32.DLL used was not the correct version. Check to make sure you have the correct version of the DLL.

Initializing enumeration with *type* *Compiler warning*

You're trying to initialize an **enum** variable to a different type. For example,

```
enum count { zero, one, two } x = 2;
```

results in this warning, because 2 is of type **int**, not type **enum count**. It is better programming practice to use an **enum** identifier instead of a literal integer when assigning to or initializing **enum** types.

This is an error, but is reduced to a warning to give existing programs a chance to work.

Inline assembly not allowed *Compiler error*
Your source file contains inline assembly-language statements and you're trying to compile it from within the integrated environment. You must use BCC to compile source files that contain inline assembly.

Inline assembly not allowed in inline and template functions *Compiler error*
The compiler cannot handle inline assembly statements in a C++ inline or template function. You could eliminate the inline assembly code or, in case of an inline function, make this a macro or remove the **inline** storage class.

int and string types compared *MAKE error*
You have tried to compare an integer operand with a string operand in an **!if** or **!elif** expression.

Internal linker error *errorcode* *Linker error*
An error occurred in the internal logic of TLINK. This error shouldn't occur in practice, but is listed here for completeness in the event that a more specific error isn't generated. If this error persists, write down the *errorcode* number and contact Borland Technical Support.

Invalid combination of opcode and operands *Compiler error*
The built-in assembler does not accept this combination of operands. Possible causes are the following:

- There are too many or too few operands for this assembler opcode.
- The number of operands is correct, but their types or order do not match the opcode; for example **DEC 1, MOV AX,** or **MOV 1,AX**. Try prefacing the operands with type overrides; for example **MOV AX, WORD PTR foo**.

Invalid entry point offset *Linker error*
This message occurs only when modules with 32-bit records are linked. It means that the initial program entry point offset exceeds the DOS limit of 64K.

Invalid indirection *Compiler error*
The indirection operator (*) requires a non-**void** pointer as the operand.

Invalid initial stack offset *Linker error*
This message occurs only when modules with 32-bit records are linked. It means that the initial stack pointer value exceeds the DOS limit of 64K.

Invalid macro argument separator *Compiler error*
In a macro definition, arguments must be separated by commas. The compiler encountered some other character after an argument name.

Invalid page size value ignored *Librarian warning*
Invalid page size is given. The page size must be a power of 2, and it cannot be smaller than 16 or larger than 32,768.

Invalid pointer addition *Compiler error*
Your source file attempted to add two pointers together.

Invalid register combination (e.g. [BP+BX]) *Compiler error*
The built-in assembler detected an illegal combination of registers in an instruction. Valid index register combinations are **[BX]**, **[BP]**, **[SI]**, **[DI]**, **[BX+SI]**, **[BX+DI]**, **[BP+SI]**, and **[BP+DI]**. Other index register combinations (such as **[AX]**, **[BP+BX]**, and **[SI+DX]**) are not allowed.

Local variables (variables declared in procedures and functions) are usually allocated on the stack and accessed via the BP register. The assembler automatically adds **[BP]** in references to such variables, so even though a construct like **Local[EBX]** (where **Local** is a local variable) appears valid, it is not, because the final operand would become **Local[BP+EBX]**.

Invalid segment definition *Linker error*
The compiler produced a flawed object file. If this occurs in a file created by Borland C++, recompile the file. If the problem persists, contact Borland Technical Support.

Invalid template argument list *Compiler error*
In a template declaration, the keyword **template** must be followed by a list of formal arguments enclosed within the **<** and **>** delimiters; an illegal template argument list was found.

Invalid template qualified name *template::name* *Compiler error*
When defining a template class member, the actual arguments in the template class name that is used as the left operand for the **::** operator must match the formal arguments of the template class. For example:

```
template <class T> class X
{
   void  f();
};

template <class T> void X<T>::f(){}
```

The following would be illegal:

```
template <class T> void X<int>::f(){}
```

Invalid use of dot *Compiler error*
An identifier must immediately follow a period operator (**.**).

Invalid use of template *template* *Compiler error*
Outside of a template definition, it is illegal to use a template class name without specifying its actual arguments. For example, you can use **vector<int>** but not **vector**.

Irreducible expression tree *Compiler error*
This is a sign of some form of compiler error. An expression on the indicated line of the source file has caused the code generator to be unable to generate code. The offending expression should be avoided. Notify Borland Technical Support if the compiler encounters this error.

base* is an indirect virtual base class of *class *Compiler error*
A pointer to a C++ member of the given virtual base class cannot be created; an attempt has been made to create such a pointer (either directly or through a cast).

***identifier* is assigned a value that is never used** *Compiler warning*
The variable appears in an assignment, but is never used anywhere else in the function just ending. The warning is indicated only when the compiler encounters the closing brace.

***identifier* is declared as both external and static** *Compiler warning*
This identifier appeared in a declaration that implicitly or explicitly marked it as global or external, and also in a static declaration. The identifier is taken as static. You should review all declarations for this identifier.

***identifier* is declared but never used** *Compiler warning*
Your source file declared the named variable as part of the block just ending, but the variable was never used. The warning is indicated when the compiler encounters the closing brace of the compound statement or function. The declaration of the variable occurs at the beginning of the compound statement or function.

constructor* is not a base class of *class *Compiler error*

A C++ class constructor **class** is trying to call a base class constructor **constructor**, or you are trying to change the access rights of `class::constructor`. **constructor** is not a base class of **class**. Check your declarations.

identifier* is not a member of *struct *Compiler error*

You are trying to reference *identifier* as a member of ***struct***, but it is not a member. Check your declarations.

***identifier* is not a non-static data member and can't be initialized here** *Compiler error*

Only data members can be initialized in the initializers of a constructor. This message means that the list includes a static member or function member.

***identifier* is not a parameter** *Compiler error*

In the parameter declaration section of an old-style function definition, *identifier* is declared but is not listed as a parameter. Either remove the declaration or add *identifier* as a parameter.

***type* is not a polymorphic class type** *Compiler error*

A dynamic_cast was used with a pointer to a class that was compiled with the –RT compiler option disabled.

identifier* is not a public base class of *classtype *Compiler error*

The right operand of a **.***, **–>***, or **::operator** was not a pointer to a member of a class that is either identical to or an unambiguous accessible base class of the left operand's class type.

***filename* is not a valid library** *Linker warning*

This error occurs if a file that wasn't a valid library module was passed to the linker in the library section.

***member* is not accessible** *Compiler error*

You are trying to reference C++ class member *member*, but it is **private** or **protected** and cannot be referenced from this function. This sometimes happens when attempting to call one accessible overloaded member function (or constructor), but the arguments match an inaccessible function. The check for overload resolution is always made before checking for accessibility. If this is the problem, try an explicit cast of one or more parameters to select the desired accessible function.

Last parameter of *operator* must have type int *Compiler error*

When a postfix **operator++** or **operator– –** is declared, the last parameter must be declared with the type **int**.

Library contains COMDEF records – extended dictionary not created *Librarian warning*

An object record being added to a library contains a COMDEF record. This is not compatible with the extended dictionary option.

Library too large, restart with library page size *size* *Librarian error*

The library being created could not be built with the current library page size.

Linkage specification not allowed *Compiler error*

Linkage specifications such as **extern** "C" are allowed only at the file level. Move this function declaration out to the file level.

Linker name conflict for *function* *Compiler error*

When the mangled name of a C++ inline function or a virtual table is too long and has to be truncated (this happens most often with templates), and the truncated name matches a previously generated function or virtual table, this error is issued by the compiler. The problem can be fixed by changing the name of the class or function, or by compiling with the **–Vs** option.

Linker stack overflow *Linker error*

The linker uses a recursive procedure for marking modules to be included in an executable image from libraries. This procedure can cause stack overflows in extreme circumstances. If you get this error message, remove some modules from libraries, include them with the object files in the link, and try again.

Lvalue required *Compiler error*

The left hand side of an assignment operator must be an addressable expression. These include numeric or pointer variables, structure field references or indirection through a pointer, or a subscripted array element.

Macro argument syntax error *Compiler error*

An argument in a macro definition must be an identifier. The compiler encountered some non-identifier character where an argument was expected.

Macro expansion too long *Compiler error*

A macro cannot expand to more than 4,096 characters.

Macro expansion too long *MAKE error*

A macro cannot expand to more than 4,096 characters. This error often occurs if a macro recursively expands itself. A macro cannot legally expand to itself.

Macro substitute text *string* is too long *MAKE error*
Macro replace text *string* is too long *MAKE error*

The macro substitution or replacement text *string* overflowed MAKE's internal buffer of 512 bytes.

main must have a return type of int *Compiler error*

In C++, function **main** has special requirements, one of which is that it cannot be declared with any return type other than **int**.

Malformed command-line *Compiler error*

An invalid entry in the command line was found.

Matching base class function for *function* has different dispatch number. *Compiler error*

If a DDVT function is declared in a derived class, the matching base class function must have the same dispatch number as the derived function.

Matching base class function for *function* is not dynamic *Compiler error*

If a DDVT function is declared in a derived class, the matching base class function must also be dynamic.

Maximum precision used for member pointer type *type* *Compiler warning*

When you use the **–Vmd** option, the compiler has to use the most general (and the least efficient) representation for that member pointer type when it is declared and its class hasn't been fully defined. This can cause less efficient code to be generated (and make the member pointer type unnecessarily large), and can also cause problems with separate compilation.

Member function must be called or its address taken *Compiler error*

When a member function is used in an expression, either it must be called or its address must be taken using the **&** operator. In this case, a member function has been used in an illegal context.

Member identifier expected *Compiler error*

The name of a structure or C++ class member was expected here, but not found. The right side of a dot (**.**) or arrow (**->**) operator must be the name of a member in the structure or class on the left of the operator.

Member is ambiguous: *member1* and *member2* *Compiler error*

You must qualify the member reference with the appropriate base class name. In C++ class *class*, member *member* can be found in more than one base class, and was not qualified to indicate which was meant. This happens only in multiple inheritance, where the member name in each base class is not hidden by the same member name in a derived class on the same path. The C++ language rules require that this test for ambiguity be made before checking for access rights (**private**,

protected, **public**). It is therefore possible to get this message even though only one (or none) of the members can be accessed.

Member *member* cannot be used without an object *Compiler error*
This means that the user has written `class::member` where *member* is an ordinary (nonstatic) member, and there is no class to associate with that member. For example, it is legal to write `obj.class::member`, but not to write `class::member`.

Member *member* has the same name as its class *Compiler error*
A static data member, enumerator, member of an anonymous union, or nested type cannot have the same name as its class. Only a member function or a nonstatic member can have a name that is identical to its class.

Member *member* is initialized more than once *Compiler error*
In a C++ class constructor, the list of initializations following the constructor header includes the same member name more than once.

Member pointer required on right side of .* or –>* *Compiler error*
The right side of a C++ dot-star (**.***) or an arrow-star (**->***) operator must be declared as a pointer to a member of the class specified by the left side of the operator. In this case, the right side is not a member pointer.

Memory full listing truncated! *Librarian warning*
The librarian has run out of memory creating a library listing file. A list file will be created but is not complete.

Memory reference expected *Compiler error*
The built-in assembler requires a memory reference. Most likely you have forgotten to put square brackets around an index register operand; for example, **MOV AX,BX+SI** instead of **MOV AX,[BX+SI]**.

Misplaced break *Compiler error*
The compiler encountered a **break** statement outside a **switch** or looping construct.

Misplaced continue *Compiler error*
The compiler encountered a **continue** statement outside a looping construct.

Misplaced decimal point *Compiler error*
The compiler encountered a decimal point in a floating-point constant as part of the exponent.

Misplaced elif directive *Compiler error*
The compiler encountered an **#elif** directive without any matching **#if**, **#ifdef**, or **#ifndef** directive.

Misplaced elif statement *MAKE error*
An **!elif** directive is missing a matching **!if** directive.

Misplaced else *Compiler error*
The compiler encountered an **else** statement without a matching **if** statement. An extra **else** statement could cause this message, but it could also be caused by an extra semicolon, missing braces, or some syntax error in a previous **if** statement.

Misplaced else directive *Compiler error*
The compiler encountered an **#else** directive without any matching **#if**, **#ifdef**, or **#ifndef** directive.

Misplaced else statement *MAKE error*
An **!else** directive does not have a matching **!if** directive.

Misplaced endif directive *Compiler error*
The compiler encountered an **#endif** directive without a matching **#if**, **#ifdef**, or **#ifndef** directive.

Misplaced endif statement *MAKE error*
An **!endif** directive does have a matching **!if** directive.

***filename* (*linenum*): Missing internal name** *Linker error*
In the IMPORTS section of the module definition file there was a reference to an entry specified via module name and ordinal number. When an entry is specified by ordinal number an internal name must be assigned to this import definition. Your program uses this internal name to refer to the imported definition. The syntax in the module definition file should be:

```
<internalname>=<modulename>.<ordinal>
```

Mixed common types in module *module*. Cannot mix COMDEFs and VIRDEFs. *Linker error*
You cannot mix both COMDEFs and VIRDEFs. Turn off the **–Fc** switch to stop generating COMDEFs, or turn on the **–Vs** switch to stop generating VIRDEFs.

Mixing pointers to different 'char' types *Compiler warning*
You converted a **signed char** pointer to an **unsigned char** pointer, or vice versa, without using an explicit cast. (Strictly speaking, this is incorrect, but it is often harmless.)

Multiple base classes require explicit class names *Compiler error*
In a C++ class constructor, each base class constructor call in the constructor header must include the base class name when there is more than one immediate base class.

Multiple declaration for *identifier* *Compiler error*
This identifier was improperly declared more than once. This might be caused by conflicting declarations such as `int a; double a;`, by a function declared two different ways, by a label repeated in the same function, or by some declaration repeated other than an **extern** function or a simple variable (in C).

Multiple entry points defined *Linker error*
More than one entry point was defined for the application. You can only have one entry point.

***identifier* must be a member function** *Compiler error*
Most C++ operator functions can be members of classes or ordinary nonmember functions, but certain ones are required to be members of classes. These are **operator =**, **operator –>**, **operator ()**, and type conversions. This operator function is not a member function but should be.

***identifier* must be a member function or have a parameter of class type** *Compiler error*
Most C++ operator functions must have an implicit or explicit parameter of class type. This operator function was declared outside a class and does not have an explicit parameter of class type.

***identifier* must be a previously defined class or struct** *Compiler error*
You are attempting to declare *identifier* to be a base class, but either it is not a class or it has not yet been fully defined. Correct the name or rearrange the declarations.

***identifier* must be a previously defined enumeration tag** *Compiler error*
This declaration is attempting to reference *identifier* as the tag of an **enum** type, but it has not been so declared. Correct the name, or rearrange the declarations.

***identifier* must be declared with no parameters** *Compiler error*
This C++ operator function was incorrectly declared with parameters.

***identifier* must be declared with one parameter** *Compiler error*
This C++ operator function was incorrectly declared with more than one parameter.

***operator* must be declared with one or no parameters** *Compiler error*

When **operator++** or **operator – –** is declared as a member function, it must be declared to take either no parameters (for the prefix version of the operator) or one parameter of type **int** (for the postfix version).

***operator* must be declared with one or two parameters** *Compiler error*

When **operator++** or **operator – –** is declared as a nonmember function, it must be declared to take either one parameter (for the prefix version of the operator) or two parameters (for the postfix version).

***identifier* must be declared with two parameters** *Compiler error*

This C++ operator function was incorrectly declared with other than two parameters.

Must take address of a memory location *Compiler error*

Your source file used the address-of operator (**&**) with an expression that cannot be used that way; for example, a register variable (in C).

Need an identifier to declare *Compiler error*

In this context, an identifier was expected to complete the declaration. This might be a **typedef** with no name, or an extra semicolon at file level. In C++, it might be a class name improperly used as another kind of identifier.

No : following the ? *Compiler error*

The question mark (**?**) and colon (**:**) operators do not match in this expression. The colon might have been omitted, or parentheses might be improperly nested or missing.

No base class to initialize *Compiler error*

This C++ class constructor is trying to implicitly call a base class constructor, but this class was declared with no base classes. Check your declarations.

No closing quote *MAKE error*

There is no closing quote for a string expression in a **!if** or **!elif** expression.

No declaration for function *function* *Compiler warning*

You called a function without first declaring that function. In C, you can declare a function without presenting a prototype, as in `int func();`. In C++, every function declaration is also a prototype; this example is equivalent to `int func(void);`. The declaration can be either classic or modern (prototype) style.

No module definition file specified; using defaults *Linker warning*

This warning occurs when you do not specify a .DEF file for the link.

No file name ending *Compiler error*

The file name in an #**include** statement was missing the correct closing quote or angle bracket.

No filename ending *MAKE error*

The file name in an **!include** statement is missing the correct closing quote or angle bracket.

No file names given *Compiler error*

The command line of the Borland C++ command-line compiler (BCC) contained no file names. You must specify a source file name.

No internal name for IMPORT in .DEF file *Linker error*

The .DEF file has a semantic error. You probably forgot to put the internal name for an import before the module name. For example:

```
IMPORTS
   _foo.1
```

The proper syntax is:

```
IMPORTS
  _foo=mydll.1
```

No macro before = *MAKE error*
You must give a macro a name before you can assign it a value.

No match found for wildcard *expression* *MAKE error*
There are no files matching the wildcard *expression* for MAKE to expand. For example, if you write

```
prog.exe: *.obj
```

MAKE sends this error message if there are no files with the extension .OBJ in the current directory.

No output file specified *Linker error*
No EXE or DLL file was specified. Because the linker defaults to the first .OBJ name, this error is usually caused because no object object files were included.

No program starting address defined *Linker warning*
This warning means that no module defined the initial starting address of the program. This is probably caused by forgetting to link in the initialization module C0*x*.OBJ.

No stack *Linker warning*
This warning is issued if no stack segment is defined in any of the object files or in any of the libraries included in the link. This is a normal message for the tiny memory model in Borland C++, or for any application program that will be converted to a .COM file. Except for DLLs, this indicates an error.

If a Borland C++ program produces this message for any but the tiny memory model, make sure you are using the correct C0*x* startup object files.

No stub for fixup at *address* *Linker warning*
This error occurs when the target for a fixup is in an overlay segment, but no stub is found for a target external. This is usually the result of not making public a symbol in an overlay that is referenced from the same module.

No terminator specified for in-line file operator *MAKE error*
The makefile contains either the **&&** or **<<** command-line operators to start an inline file, but the file is not terminated.

Non-const function *function* called for const object *Compiler warning*
A non-**const** member function was called for a **const** object. This is an error, but was reduced to a warning to give existing programs a chance to work.

Nonportable pointer comparison *Compiler warning*
Your source file compared a pointer to a nonpointer other than the constant zero. You should use a cast to suppress this warning if the comparison is proper.

Nonportable pointer conversion *Compiler error*
An implicit conversion between a pointer and an integral type is required, but the types are not the same size. This cannot be done without an explicit cast. This conversion might not make any sense, so be sure this is what you want to do.

Nonportable pointer conversion *Compiler warning*
A nonzero integral value is used in a context where a pointer is needed or where an integral value is needed; the sizes of the integral type and pointer are the same. Use an explicit cast if this is what you really meant to do.

Nonresident Name Table is greater than 64K *Linker warning*

The maximum size of the Nonresident name table is 64K (in accordance with the industry-wide executable specification standard). The linker continues with the link but ignores any subsequent Nonresident names encountered during linking.

Nontype template argument must be of scalar type *Compiler error*

A nontype formal template argument must have scalar type; it can have an integral, enumeration, or pointer type.

Non-ANSI Keyword Used: *keyword* *Compiler error*

A non-ANSI keyword (such as **__fastcall**) was used when strict ANSI conformance was requested via the **–A** option.

Non-virtual function *function* declared pure *Compiler error*

Only virtual functions can be declared pure, because derived classes must be able to override them.

Non-volatile function *function* called for volatile object *Compiler warning*

In C++, a class member function was called for a volatile object of the class type, but the function was not declared with **volatile** following the function header. Only a volatile member function can be called for a volatile object.

***filename* not a MAKE** *MAKE error*

The file you specified with the **–f** option is not a makefile.

Not an allowed type *Compiler error*

Your source file declared some sort of forbidden type; for example, a function returning a function or array.

Not enough memory *MAKE error*

All your working storage has been exhausted.

Not enough memory to run application *Linker error*

There is not enough memory to run TLINK. Try reducing the size of any RAM disk or disk cache currently active. If you're running real mode, try using the MAKE **–S** option, or removing TSRs and network drivers.

Not enough memory for command-line buffer *Librarian error*

This error occurs when the librarian runs out of memory.

***module* not found in library** *Librarian warning*

An attempt to perform either a '_' or '*' on a library has been made and the indicated object does not exist in the library.

Null pointer assignment *Run-time error*

When a small or medium memory model program exits, a check is made to determine if the contents of the first few bytes within the program's data segment have changed. These bytes would never be altered by a working program. If they have been changed, the message "Null pointer assignment" is displayed to inform you that (most likely) a value was stored to an uninitialized pointer. The program might appear to work properly in all other respects; however, this is a serious bug which should be attended to immediately. Failure to correct an uninitialized pointer can lead to unpredictable behavior (including "locking" the computer up in the large, compact, and huge memory models). You can use the integrated debugger to track down null pointers.

Numeric constant too large *Compiler error*

String and character escape sequences larger than hexadecimal \xFF or octal \377 cannot be generated. Two-byte character constants can be specified by using a second backslash. For example, \x0D\x0A represents a two-byte constant. A numeric literal following an escape sequence should be broken up like this:

```
printf("\x0D" "12345");
```

This prints a carriage return followed by `12345`.

Object module *filename* is invalid *Librarian error*
The librarian could not understand the header record of the object module being added to the library and has assumed that it is an invalid module.

Objects of type *type* cannot be initialized with {} *Compiler error*
Ordinary C structures can be initialized with a set of values inside braces. C++ classes can be initialized with constructors only if the class has constructors, private members, functions, or base classes that are virtual.

Old debug information in module *module* will be ignored *Linker warning*
Debug information in the .OBJ file is incompatible with this linker, and it will be ignored.

Only <<KEEP or <<NOKEEP *MAKE error*
You have specified something besides KEEP or NOKEEP when closing a temporary inline file.

Only member functions may be 'const' or 'volatile' *Compiler error*
Something other than a class member function has been declared **const** and/or **volatile.**

Only one of a set of overloaded functions can be "C" *Compiler error*
C++ functions are by default overloaded, and the compiler assigns a new name to each function. If you want to override the compiler's assigning a new name by declaring the function extern "C", you can do this for only one of a set of functions with the same name. (Otherwise the linker would find more than one global function with the same name.)

Operand of delete must be non-const pointer *Compiler error*
It is illegal to delete a constant pointer value using operator **delete.**

Operator [ ] missing] *Compiler error*
The C++ **operator[ ]** was declared as **operator [**. You must add the missing] or otherwise fix the declaration.

Operator –> must return a pointer or a class *Compiler error*
The C++ **operator->** function must be declared to either return a class or a pointer to a **class** (or **struct** or **union**). In either case, it must be something to which the –> operator can be applied.

Operator delete must return void *Compiler error*
This C++ overloaded operator **delete** was declared in some other way.

Operator delete[] must return void *Compiler error*
This C++ overloaded operator **delete** was declared in some other way. Declare the **delete** with one of the following:

- A single void* parameter
- A second parameter of type size_t

If you use the second version, it will be used in preference to the first version. The global operator **delete** can be declared using the single-parameter form only.

Operator must be declared as function *Compiler error*
An overloaded operator was declared with something other than function type.

Operator new must have an initial parameter of type size_t *Compiler error*
Operator **new** can be declared with an arbitrary number of parameters, but it must always have at least one parameter that specifies the amount of space to allocate.

Operator new[] must have an initial parameter of type size_t *Compiler error*
Operator **new** can be declared with an arbitrary number of parameters. It must always have at least one parameter that specifies the amount of space to allocate.

Operator new must return an object of type void * *Compiler error*
The C++ overloaded operator **new** was declared another way.

Operator new[] must return an object of type void * *Compiler error*
This C++ overloaded operator **new** was declared another way.

Operators may not have default argument values *Compiler error*
It is illegal for overloaded operators to have default argument values.

Out of memory *Compiler error*
The total working storage is exhausted. Compile the file on a machine with more memory.

Out of memory *Librarian error*
For any number of reasons, the librarian or Borland C++ ran out of memory while building the library. For many specific cases a more detailed message is reported, leaving "Out of memory" to be the basic catchall for general low-memory situations.

If you get this message because the public symbol tables have grown too large, you must free up memory. For the command line this could involve removing TSR's or device drivers using real mode memory or close windows. In the IDE, some additional memory can be gained by closing editors.

Out of memory *Linker error*
The linker has run out of dynamically allocated memory needed during the link process. This error is a catchall for running into a TLINK limit on memory usage. This usually means that too many modules, externals, groups, or segments have been defined by the object files being linked together. You can try reducing the size of RAM disks and/or disk caches that might be active.

Out of memory creating extended dictionary *Librarian error*
The librarian has run out of memory creating an extended dictionary for a library. The library is created but will not have an extended dictionary.

Out of memory for block *block* *Linker error*
This error should not occur. If it does, call Borland Technical Support and give them the text of the message, including the block name.

Out of memory reading LE/LIDATA record from object module *Librarian error*
The librarian is attempting to read a record of data from the object module, but it cannot get a large enough block of memory. If the module that is being added has a large data segment or segments, it is possible that adding the module before any other modules might resolve the problem. By adding the module first, there will be memory available for holding public symbol and module lists later.

Out of space allocating per module debug struct *Librarian error*
The librarian ran out of memory while allocating space for the debug information associated with a particular object module. Removing debugging information from some modules being added to the library might resolve the problem.

Output device is full *Librarian error*
Usually this error means that no space is left on the disk.

Overlays generated and no overlay manager included *Linker warning*
This warning is issued if overlays are created but the symbol *__OVRTRAP__* is not defined in any of the object modules or libraries linked in. The standard overlay library (OVERLAY.LIB) defines this symbol.

Overlays only supported in medium, large, and huge memory models *Compiler error*
Only programs using the medium, large, or huge memory models can be overlaid.

Overloadable operator expected *Compiler error*

Almost all C++ operators can be overloaded. The only ones that can't be overloaded are the field-selection dot (.), dot-star (.*), double colon (::), and conditional expression (?:). The preprocessor operators (# and ##) are not C or C++ language operators and thus cannot be overloaded. Other nonoperator punctuation, such as a semicolon (;) cannot be overloaded.

Overloaded *function name* ambiguous in this context *Compiler error*

The only time an overloaded function name can be used without actually calling the function is when a variable or parameter of an appropriate type is initialized or assigned. This error was issued because an overloaded function name has been used in some other context.

Overloaded prefix 'operator *operator*' used as a postfix operator *Compiler warning*

With the latest specification of C++, it is now possible to overload both the prefix and postfix versions of the **++** and **– –** operators. To allow older code to compile, Borland C++ uses the prefix operator and issues this warning whenever only the prefix operator is overloaded, but is used in a postfix context.

P1001 Unable to read file *filename* *Help project error*

The file specified in the project file is unreadable. This is a DOS file error.

P1003 Invalid path specified in Root option *Help project error*

The path specified by the Root option cannot be found. The compiler uses the current working directory.

P1005 Path and filename exceed limit of 79 characters *Help project error*

The absolute pathname, or the combined root and relative pathname, exceed the DOS limit of 79 characters. The file is skipped.

P1007 Root path exceeds maximum limit of 66 characters *Help project error*

The specified root pathname exceeds the DOS limit of 66 characters. The pathname is ignored and the compiler uses the current working directory.

P1009 [FILES] section missing *Help project error*

The [Files] section is required. The compilation is aborted.

P1011 Option *optionname* previously defined *Help project error*

The specified option was defined previously. The compiler ignores the attempted redefinition.

P1013 Project file extension cannot be .HLP *Help project error*

You cannot specify that the compiler use a project file with the .HLP extension. Normally, project files are given the .HPJ extension.

P1015 Unexpected end-of-file *Help project error*

The compiler has unexpectedly come to the end of the project file. There might be an open comment in the project file or an included file.

P1017 Parameter exceeds maximum length of 128 characters *Help project error*

An option, context name or number, build tag, or other parameter on the specified line exceeds the limit of 128 characters. The line is ignored.

P1021 Context number already used in [MAP] section *Help project error*

The context number on the specified line in the project file was previously mapped to a different context string. The line is ignored.

P1023 Include statements nested too deeply *Help project error*

The ***#include*** statement on the specified line has exceeded the maximum of five include levels.

P1025 Section heading *sectionname* unrecognized *Help project error*

A section heading that is not supported by the compiler has been used. The line is skipped.

P1027 Bracket missing from section heading sectionname *Help project error*
The right bracket (]) is missing from the specified section heading. Insert the bracket and compile again.

P1029 Section heading missing *Help project error*
The section heading on the specified line is not complete. This error is also reported if the first entry in the project file is not a section heading. The compiler continues with the next line.

P1030 Section *sectionname* previously defined *Help project error*
A duplicate section has been found in the project file. The lines under the duplicated section heading are ignored and the compiler continues from the next valid section heading.

P1031 Maximum number of build tags exceeded *Help project error*
The maximum number of build tags that can be defined is 30. The excess tags are ignored.

P1033 Duplicate build tag in [BUILDTAGS] section *Help project error*
A build tag in the [BUILDTAGS] section has been repeated unnecessarily

P1035 Build tag length exceeds maximum *Help project error*
The build tag on the specified line exceeds the maximum of 32 characters. The compiler skips this entry.

P1037 Build tag *tagname* contains invalid characters *Help project error*
Build tags can contain only alphanumeric characters or the underscore (_) character. The line is skipped.

P1039 [BUILDTAGS] section missing *Help project error*
The ***BUILD*** option declared a conditional build, but there is no [BuildTags] section in the project file. All topics are included in the build.

P1043 Too many tags in Build expression *Help project error*
The Build expression on the specified line has used more than the maximum of 20 build tags. The compiler ignores the line.

P1045 [ALIAS] section found after [MAP] section *Help project error*
When used, the [Alias] section must precede the [Map] section in the project file. The [Alias] section is skipped otherwise.

P1047 Context string *contextname* already assigned an alias *Help project error*
You cannot do: a=b then a=c<_>(A context string can only have one alias.)

The specified context string has previously been aliased in the [Alias] section. The attempted reassignment on this line is ignored.

P1049 Alias string aliasname already assigned *Help project error*
You cannot do: a=b then b=c

An alias string cannot, in turn, be assigned another alias.

P1051 Context string *contextname* cannot be used as alias string *Help project error*
You cannot do: a=b then c=a

A context string that has been assigned an alias cannot be used later as an alias for another context string.

P1053 Maximum number of font ranges exceeded *Help project error*
The maximum number of font ranges that can be specified is five. The rest are ignored.

P1055 Current font range overlaps previously defined range *Help project error*
A font size range overlaps a previously defined mapping. Adjust either font range to remove any overlaps. The second mapping is ignored.

P1056 Unrecognized font name in Forcefont option *Help project error*
A font name not supported by the compiler has been encountered. The font name is ignored and the compiler uses the default Helvetica font.

P1057 Font name too long *Help project error*
Font names cannot exceed 20 characters. The font is ignored.

P1059 Invalid multiple-key syntax *Help project error*
The syntax used with a ***MULTIKEY*** option is unrecognized.

P1061 Character already used *Help project error*
The specified keyword–table identifier is already in use. Choose another character.

P1063 Characters 'K' and 'k' cannot be used *Help project error*
These characters are reserved for Help's normal keyword table. Choose another character.

P1065 Maximum number of keyword tables exceeded *Help project error*
The limit of five keyword tables has been exceeded. Reduce the number. The excess tables are ignored.

P1067 Equal sign missing *Help project error*
An option is missing its required equal sign on the specified line. Check the syntax for the option.

P1069 Context string missing *Help project error*
The line specified is missing a context string before an equal sign.

P1071 Incomplete line in *sectionname* section *Help project error*
The entry on the specified line is not complete. The line is skipped.

P1073 Unrecognized option in [OPTIONS] section *Help project error*
An option has been used that is not supported by the compiler. The line is skipped.

P1075 Invalid build expression *Help project error*
The syntax used in the build expression on the specified line contains one or more logical or syntax errors.

P1077 Warning level must be 1, 2, or 3 *Help project error*
The ***WARNING*** reporting level can only be set to 1, 2, or 3. The compiler will default to full reporting (level 3).

P1079 Invalid compression option *Help project error*
The ***COMPRESS*** option can only be set to TRUE or FALSE. The compilation continues without compression.

P1081 Invalid title string *Help project error*
The ***TITLE*** option defines a string that is null or contains more than 32 characters. The title is truncated.

P1083 Invalid context identification number *Help project error*
The context number on the specified line is null or contains invalid characters.

P1085 Unrecognized text *Help project error*
The unrecognizable text that follows valid text in the specified line is ignored.

P1086 Invalid font-range syntax *Help project error*
The font-range definition on the specified line contains invalid syntax. The compiler ignores the line. Check the syntax for the ***MAPFONTSIZE*** option.

P1089 Unrecognized sort ordering *Help project error*
You have specified an ordering that is not supported by the compiler. Contact Borland Technical Support for clarification of the error.

Parameter names are used only with a function body *Compiler error*

When declaring a function (not defining it with a function body), you must use either empty parentheses or a function prototype. A list of parameter names only is not allowed.

Example declarations include:

```
int func();              // declaration without prototype--OK
int func(int, int);      // declaration with prototype--OK
int func(int i, int j); // parameter names in prototype--OK
int func(i, j);          // parameter names only--illegal
```

Parameter *number* missing name *Compiler error*

In a function definition header, this parameter consisted only of a type specifier *number* with no parameter name. This is not legal in C. (It is allowed in C++, but there's no way to refer to the parameter in the function.)

Parameter *parameter* is never used *Compiler warning*

The named parameter, declared in the function, was never used in the body of the function. This might or might not be an error and is often caused by misspelling the parameter. This warning can also occur if the identifier is redeclared as an automatic (local) variable in the body of the function. The parameter is masked by the automatic variable and remains unused.

***path* – path is too long** *Librarian error*

This error occurs when the length of any of the library file or module file's *path* is greater than 64 characters.

Pointer to structure required on left side of –> or –>* *Compiler error*

Nothing but a pointer is allowed on the left side of the arrow (->) in C or C++. In C++ a **–>*** operator is allowed.

Possible reference to undefined extern *xxxx::i* in module *module* *Linker warning*

Static data member has been declared but not defined in your application.

Possible unresolved external *sym* referenced from module *mod* *Linker warning*

This warning appears only for static data members of classes that have been declared but not defined.

Possible use of *identifier* before definition *Compiler warning*

Your source file used the named variable in an expression before it was assigned a value. The compiler uses a simple scan of the program to determine this condition. If the use of a variable occurs physically before any assignment, this warning will be generated. Of course, the actual flow of the program might assign the value before the program uses it.

Possibly incorrect assignment *Compiler warning*

This warning is generated when the compiler encounters an assignment operator as the main operator of a conditional expression (that is, as part of an **if, while** or **do-while** statement). More often than not, this is a typographical error for the equality operator. If you want to suppress this warning, enclose the assignment in parentheses and compare the whole thing to zero explicitly. Thus,

```
if (a = b) ...
```

should be rewritten as

```
if ((a = b) != 0) ...
```

Program entry point may not reside in an overlay *Linker error*

Although almost all of an application can be overlaid, the initial starting address cannot reside in an overlay. This error usually means that an attempt was made to overlay the initialization module (C0*x*.OBJ, for instance) by specifying the ***/o*** option before the startup module.

Public *symbol* in module *module1* clashes with prior module *module2* *Librarian error*

A public symbol can appear only once in a library file. A module that is being added to the library contains a public *symbol* that is already in a module of the library and cannot be added. The command-line message reports the *module2* name.

Public *symbol* in module *filename* clashes with prior module *Librarian error*

A public symbol can appear only once in a library file. A module that is being added to the library contains a public *symbol* that is already in a module of the library and cannot be added.

R2001 Unable to open bitmap file *filename* *Help RTF error*

The specified bitmap file is unreadable. This is a DOS file error.

R2003 Unable to include bitmap file *filename* *Help RTF error*

The specified bitmap file could not be found or is unreadable. This is a DOS file error or an out-of-memory condition.

R2005 Disk full *Help RTF error*

The Help resource file could not be written to disk. Create more space on the destination drive.

R2009 Cannot use reserved DOS device name for file *filename* *Help RTF error*

A file has been referred to as COM1, LPT2, PRN, etc. Rename the file.

R2013 Output file *filename* already exists as a directory *Help RTF error*

There is a subdirectory in the Help project root with the same name as the desired Help resource file. Move or rename the subdirectory.

R2015 Output file *filename* already exists as read-only *Help RTF error*

The specified filename cannot be overwritten by the Help resource file because the file has a read-only attribute. Rename the project file or change the file's attribute.

R2017 Path for file *filename* exceeds limit of 79 characters *Help RTF error*

The absolute pathname, or the combined root and relative pathname, to the specified file exceed the DOS limit of 79 characters. The file is ignored.

R2019 Cannot open file *filename* *Help RTF error*

The specified file is unreadable. This is a DOS file error.

R2021 Cannot find file *filename* *Help RTF error*

The specified file could not be found or is unreadable. This is a DOS file error or an out-of-memory condition.

R2023 Not enough memory to build Help file *Help RTF error*

To free up memory, unload any unneeded applications, device drivers, and memory-resident programs.

R2025 File environment error *Help RTF error*

The compiler has insufficient available file handles to continue. Increase the values for FILES= and BUFFERS= in your CONFIG.SYS file and reboot.

R2027 Build tag *tagname* not defined in [BUILDTAGS] section of project file *Help RTF error*

The specified build tag has been assigned to a topic, but not declared in the project file. The tag is ignored for the topic.

R2033 Context string in Map section not defined in any topic *Help RTF error*

There are one or more context strings defined in the project file that the compiler could not find topics for.

R2035 Build expression missing from project file *Help RTF error*

The topics have build tags, but there is no Build= expression in the project file. The compiler includes all topics in the build.

R2037 File *filename* cannot be created, due to previous error(s) *Help RTF error*

The Help resource file could not be created because the compiler has no topics remaining to be processed. Correct the errors that preceded this error and recompile.

R2039 Unrecognized table formatting in topic *topicnumber* of file *filename* *Help RTF error*
The compiler ignores table formatting that is unsupported in Help. Reformat the entries as linear text if possible.

R2041 Jump *context_string* unresolved in topic *topicnumber* of file *filename* *Help RTF error*
The specified topic contains a context string that identifies a nonexistent topic. Check spelling, and that the desired topic is included in the build.

R2043 Hotspot text cannot spread over paragraphs *Help RTF error*
A jump term spans two paragraphs. Remove the formatting from the paragraph mark.

R2045 Maximum number of tab stops reached in topic *topicnumber* of file *filename* *Help RTF error*
The limit of 32 tab stops has been exceeded in the specified topic. The default stops are used after the 32nd tab.

R2047 File *filename* not created *Help RTF error*
There are no topics to compile, or the build expression is false for all topics. There is no Help resource file created.

R2049 Context string text too long in topic *topicnumber* of file *filename* *Help RTF error*
Context string hidden text cannot exceed 64 characters. The string is ignored.

R2051 File *filename* is not a valid RTF topic file *Help RTF error*
The specified file is not an RTF file. Check that you have saved the topic as RTF from your word processor.

R2053 Font *fontname* in file *filename* not in RTF font table *Help RTF error*
A font not defined in the RTF header has been entered into the topic. The compiler uses the default system font.

R2055 File *filename* is not a usable RTF topic file *Help RTF error*
The specified file contains a valid RTF header, but the content is not RTF or is corrupted.

R2057 Unrecognized graphic format in topic *topicnumber* of file *filename* *Help RTF error*
The compiler supports only Windows bitmaps. Check that metafiles or Macintosh formats have not been used. The graphic is ignored.

R2059 Context string identifier already defined in topic *topicnumber* of file *filename* *Help RTF error*
There is more than one context-string identifier footnote for the specified topic. The compiler uses the identifier defined in the first # footnote.

R2061 Context string *contextname* already used in file *filename* *Help RTF error*
The specified context string was previously assigned to another topic. The compiler ignores the latter string and the topic has no identifier.

R2063 Invalid context-string identifier for topic *topicnumber* of file *filename* *Help RTF error*
The context-string footnote contains nonalphanumeric characters or is null. The topic is not assigned an identifier.

R2065 Context string defined for index topic is unresolved *Help RTF error*
The index topic defined in the project file could not be found. The compiler uses the first topic in the build as the index.

R2067 Footnote text too long in topic *topicnumber* of file *filename* *Help RTF error*
Footnote text cannot exceed the limit of 1000 characters. The footnote is ignored.

R2069 Build tag footnote not at beginning of topic *topicnumber* of file *filename* *Help RTF error*
The specified topic contains a build tag footnote that is not the first character in the topic. The topic is not assigned a build tag.

R2071 Footnote text missing in topic *topicnumber* of file *filename* *Help RTF error*
The specified topic contains a footnote that has no characters.

R2073 Keyword string is null in topic *topicnumber* of file *filename* *Help RTF error*
A keyword footnote exists for the specified topic, but contains no characters.

R2075 Keyword string too long in topic *topicnumber* of file *filename* *Help RTF error*
The text in the keyword footnote in the specified topic exceeds the limit of 255 characters. The excess characters are ignored.

R2077 Keyword(s) defined without title in topic *topicnumber* of file *filename* *Help RTF error*
Keyword(s) have been defined for the specified topic, but the topic has no title assigned. Search Topics Found displays Untitled Topic<< for the topic.

R2079 Browse sequence string is null in topic *topicnumber* of file *filename* *Help RTF error*
The browse-sequence footnote for the specified topic contains no sequence characters.

R2081 Browse sequence string too long in topic *topicnumber* of file *filename* *Help RTF error*
The browse-sequence footnote for the specified topic exceeds the limit of 128 characters. The sequence is ignored.

R2083 Missing sequence number in topic *topicnumber* of file *filename* *Help RTF error*
A browse-sequence number ends in a colon (:) for the specified topic. Remove the colon, or enter a "minor" sequence number.

R2085 Sequence number already defined in topic *topicnumber* of file *filename* *Help RTF error*
There is already a browse-sequence footnote for the specified topic. The latter sequence is ignored.

R2087 Build tag too long *Help RTF error*
A build tag for the specified topic exceeds the maximum of 32 characters. The tag is ignored for the topic.

R2089 Title string null in topic *topicnumber* of file *filename* *Help RTF error*
The title footnote for the specified topic contains no characters. The topic is not assigned a title.

R2091 Title too long in topic *topicnumber* of file *filename* *Help RTF error*
The title for the specified topic exceeds the limit of 128 characters. The excess characters are ignored.

R2093 Title titlename in topic *topicnumber* of file *filename* used previously *Help RTF error*
The specified title has previously been assigned to another topic.

R2095 Title defined more than once in topic *topicnumber* of file *filename* *Help RTF error*
There is more than one title footnote in the specified topic. The compiler uses the first title string.

R2501 Using old key-phrase table *Help RTF error*
Maximum compression can only result by deleting the .PH file before each recompilation of the Help topics.

R2503 Out of memory during text compression *Help RTF error*
The compiler encountered a memory limitation during compression. Compilation continues with the Help resource file not compressed. Unload any unneeded applications, device drivers, and memory-resident programs.

R2505 File environment error during text compression *Help RTF error*
The compiler has insufficient available file handles for compression. Compilation continues with the Help resource file not compressed. Increase the values for FILES= and BUFFERS= in your CONFIG.SYS file and reboot.

R2507 DOS file error during text compression *Help RTF error*
The compiler encountered a problem accessing a disk file during compression. Compilation continues with the Help resource file not compressed.

R2509 Error during text compression *Help RTF error*
One of the three compression errors—R2503, R2505, or R2507—has occurred. Compilation continues with the Help resource file not compressed.

R2701 Internal error *Help RTF error*
R2703 Internal error *Help RTF error*
R2705 Internal error *Help RTF error*
R2707 Internal error *Help RTF error*
R2709 Internal error *Help RTF error*
Contact Borland Technical Support for clarification of the error.

Record kind *num* found, expected theadr or lheadr in module *filename* *Librarian error*
The librarian could not understand the header record of the object module being added to the library and has assumed that it is an invalid module.

Record length *len* exceeds available buffer in module *module* *Librarian error*
This error occurs when the record length *len* exceeds the available buffer to load the buffer in module *module*. This occurs when librarian runs out of dynamic memory.

Record type *type* found, expected theadr or lheadr in *module* *Librarian error*
The librarian encountered an unexpected type *type* instead of the expected THEADR or LHEADER record in module *module*.

Redefinition of *macro* is not identical *Compiler warning*
Your source file redefined the named macro using text that was not exactly the same as the first definition of the macro. The new text replaces the old.

Redefinition of target *filename* *MAKE error*
The named file occurs on the left side of more than one explicit rule.

Reference initialized with *type1*, needs lvalue of type *type2* *Compiler error*
A reference variable or parameter that is not declared constant must be initialized with an lvalue of the appropriate type. This error was issued either because the initializer wasn't an lvalue or because its type didn't match the reference being initialized.

Reference member *member* in class without constructors *Compiler error*
A class that contains reference members must have at least one user-defined constructor; otherwise, there would be no way to initialize such members.

Reference member *member* initialized with a non-reference parameter *Compiler error*
An attempt has been made to bind a reference member to a parameter in a constructor. Because the parameter object ceases to exist the moment the constructor returns, the reference member is then left referring to an undefined object. (This is a common mistake that causes crashes and erratic program behavior.)

Reference member *member* is not initialized *Compiler error*
References must always be initialized. A class member of reference type must have an initializer provided in all constructors for that class. This means that you cannot depend on the compiler to generate constructors for such a class, because it has no way of knowing how to initialize the references.

Reference member *member* needs a temporary for initialization *Compiler error*
You provided an initial value for a reference type that was not an lvalue of the referenced type. This requires the compiler to create a temporary for the initialization. Because there is no obvious place to store this temporary, the initialization is illegal.

Reference variable *variable* must be initialized *Compiler error*
This C++ object is declared as a reference but is not initialized. All references must be initialized at their point of declaration.

Register allocation failure *Compiler error*
This is a sign of some form of compiler error. Some expression in the indicated function was so complicated that the code generator could not generate code for it. Try to simplify the offending function. Notify Borland Technical Support if the compiler encounters this error.

Relocation item exceeds 1MB DOS limit *Linker error*

The DOS executable file format doesn't support relocation items for locations exceeding 1MB. Although DOS could never *load* an image this big, DOS extenders can, and thus TLINK supports generating images greater than DOS could load. Even if the image is loaded with a DOS extender, the DOS executable file format is limited to describing relocation items in the first 1MB of the image.

Relocation offset overflow *Linker error*

This error occurs only for 32-bit object modules and indicates a relocation (segment fixup) offset greater than the DOS limit of 64K.

Relocation table overflow *Linker error*

This error occurs only for 32-bit object modules. The file being linked contains more base fixups than the standard DOS relocation table can hold (base fixups are created mostly by calls to far functions).

Resident Name Table is greater than 64K *Linker warning*

The maximum size of the Resident name table is 64K (in accordance with the industry-wide executable specification standard). The linker continues with the link but ignores any subsequent Resident names encountered during linking.

Restarting compile using assembly *Compiler warning*

The compiler encountered an **ASM** with an accompanying **–B** command-line option or #**pragma inline** statement. The compile restarts using assembly language capabilities.

Results are safe in file *filename* *Librarian warning*

The librarian has successfully built the library into a temporary file, but cannot rename the file to the desired library name. The temporary file will not be removed (so that the library can be preserved).

Rule line too long *MAKE error*

An implicit or explicit rule was longer than 4,096 characters.

Segment *segment* exceeds 64K *Linker error*

This message occurs if too much data is defined for a given data or code segment when TLINK combines segments with the same name from different source files.

Segment *segment* is in two groups: *group1* and *group2* *Linker warning*

The linker found conflicting claims by the two named groups. Usually, this happens only in assembly language programs. It means that two modules assigned the segment to two different groups.

Self relative fixup overflowed in module *module* *Linker warning*

This message appears if a self-relative reference (usually a call) is made from one physical segment to another. It usually happens only when employing assembler code, but can occur if you use the segment-naming options in the compiler. If the reference is from one code segment to another, you are safe. If, however, the reference is from a code segment to a data segment, you have probably made a mistake in some assembler code.

Size of *identifier* is unknown or zero *Compiler error*

This identifier was used in a context where its size was needed. For example, a **struct** tag might only be declared (with the **struct** not defined yet), or an **extern** array might be declared without a size. If so, it's illegal to have references to such an item (like **sizeof**) or to dereference a pointer to this type. Rearrange your declaration so that the size of *identifier* is available.

sizeof may not be applied to a bit field *Compiler error*

sizeof returns the size of a data object in bytes, which does not apply to a bit field.

sizeof may not be applied to a function *Compiler error*

sizeof can be applied only to data objects, not functions. You can request the size of a pointer to a function.

Size of the type is unknown or zero *Compiler error*

This type was used in a context where its size was needed. For example, a **struct** tag might only be declared (with the **struct** not defined yet). If so, it's illegal to have references to such an item (like **sizeof**) or to dereference a pointer to this type. Rearrange your declarations so that the size of this type is available.

identifier **specifies multiple or duplicate access** *Compiler error*

A base class can be declared **public** or **private**, but not both. This access specifier can appear no more than once for a base class.

Stack overflow *Run-time error*

The default stack size for Borland C++ programs is 5120 bytes. This should be enough for most programs, but those which execute recursive functions or store a great deal of local data can overflow the stack. You will get this message only if you have stack checking enabled. If you do get this message, you can try increasing the stack size or decreasing your program's dependence on the stack. Change the stack size by altering the global variable *_stklen*. Try switching to a larger memory model to fit the larger stack.

To decrease the amount of local data used by a function, look at the example below. The variable *buffer* has been declared static and does not consume stack space like *list* does.

```
void anyfunction(void) {
   static int buffer[2000];    /* resides in the data segment */
   int list[2000];             /* resides on the stack */
}
```

There are two disadvantages to declaring local variables as static.

1. It now takes permanent space away from global variables and the heap. This is usually only a minor disadvantage.
2. The function can no longer be reentrant. If the function is called recursively or asynchronously and it is important that each call to the function have its own unique copy of the variable, you cannot make it static. This is because every time the function is called, it will use the same exact memory space for the variable, rather than allocating new space for it on each call. You could have a sharing problem if the function is trying to execute from within itself (recursively) or at the same time as itself (asynchronously). For most DOS programs this is not a problem.

Statement missing ; *Compiler error*

The compiler encountered an expression statement without a semicolon following it.

Storage class *storage class* is not allowed here *Compiler error*

The given storage class is not allowed here. Probably two storage classes were specified, and only one can be given.

String type not allowed with this operand *MAKE error*

You have tried to use an operand that is not allowed for comparing string types. Valid operands are ==, !=, <, >, <=, and >=.

Structure passed by value *Compiler warning*

A structure was passed by value as an argument to a function without a prototype. It is a frequent programming mistake to leave an address-of operator (&) off a structure when passing it as an argument. Because structures can be passed by value, this omission is acceptable. This warning provides a way for the compiler to warn you of this mistake.

Structure required on left side of . or .* *Compiler error*

The left side of a dot (.) operator (or C++ dot-star operator) must evaluate to a structure type. In this case it did not.

Structure size too large *Compiler error*

Your source file declared a structure larger than 64K.

Style of function definition is now obsolete *Compiler warning*

In C++, this old C style of function definition is illegal:

```
int func(p1, p2)
int p1, p2;
{...}
```

Subscripting missing] *Compiler error*

The compiler encountered a subscripting expression that was missing its closing bracket. This could be caused by a missing or extra operator, or by mismatched parentheses.

Superfluous & with function *Compiler warning*

An address-of operator (**&**) is not needed with function name; any such operators are discarded.

Suspicious pointer conversion *Compiler warning*

The compiler encountered some conversion of a pointer that caused the pointer to point to a different type. You should use a cast to suppress this warning if the conversion is proper.

Switch selection expression must be of integral type *Compiler error*

The selection expression in parentheses in a **switch** statement must evaluate to an integral type (**char, short, int, long, enum**). You might be able to use an explicit cast to satisfy this requirement.

Switch statement missing (*Compiler error*

In a **switch** statement, the compiler found no left parenthesis after the **switch** keyword.

Switch statement missing) *Compiler error*

In a **switch** statement, the compiler found no right parenthesis after the test expression.

***filename* (*linenum*): Syntax error** *Linker error*

The linker found a syntax error in the module definition file. The file name and line number tell you where the syntax error occurred.

Table limit exceeded *Linker error*

One of linker's internal tables overflowed. This usually means that the programs being linked have exceeded the linker's capacity for public symbols, external symbols, or for logical segment definitions. Each instance of a distinct segment name in an object file counts as a logical segment; if two object files define this segment, then this results in two logical segments.

Target index of FIXUP is 0 in module *module* *Linker error*

This is a translator error.

Template argument must be a constant expression *Compiler error*

A non-type actual template class argument must be a constant expression (of the appropriate type); this includes constant integral expressions, and addresses of objects or functions with external linkage or members.

Template class nesting too deep: 'class' *Compiler error*

The compiler imposes a certain limit on the level of template class nesting; this limit is usually exceeded only through a recursive template class dependency. When this nesting limit is exceeded, the compiler issues this error message for all of the nested template classes, which usually makes it easy to spot the recursion. This is always followed by the fatal error **Out of memory**.

For example, consider the following set of template classes:

```
template<class T> class A
{
   friend class   B<T*>;
};

template<class T> class B
{
   friend class   A<T>;
};

A<int>  x;
```

This code will be flagged with the following errors:

```
Error: Template class nesting too deep: 'B<int * * * * *>'
Error: Template class nesting too deep: 'A<int * * * *>'
Error: Template class nesting too deep: 'B<int * * * *>'
Error: Template class nesting too deep: 'A<int * * *>'
Error: Template class nesting too deep: 'B<int * * *>'
Error: Template class nesting too deep: 'A<int * *>'
Error: Template class nesting too deep: 'B<int * *>'
Error: Template class nesting too deep: 'A<int *>'
Error: Template class nesting too deep: 'B<int *>'
Error: Template class nesting too deep: 'A<int>'
Fatal: Out of memory
```

Template function argument *argument* not used in argument types *Compiler error*

The given argument was not used in the argument list of the function. The argument list of a template function must use all of the template formal arguments; otherwise, there is no way to generate a template function instance based on actual argument types.

Template functions may only have type-arguments *Compiler error*

A function template was declared with a non-type argument. This is not allowed with a template function because there is no way to specify the value when calling it.

Templates can only be declared at file level *Compiler error*

Templates cannot be declared inside classes or functions; they are allowed only in the global scope (file level).

Templates must be classes or functions *Compiler error*

The declaration in a template declaration must specify either a class type or a function.

Temporary used to initialize *identifier* *Compiler warning*
Temporary used for parameter *number* in call to *function* *Compiler warning*
Temporary used for parameter *number* *Compiler warning*
Temporary used for parameter *parameter* *Compiler warning*

In C++, a variable or parameter of reference type must be assigned a reference to an object of the same type. If the types do not match, the actual value is assigned to a temporary of the correct type, and the address of the temporary is assigned to the reference variable or parameter. The warning means that the reference variable or parameter does not refer to what you expect, but to a temporary variable, otherwise unused.

In the following example, function *f* requires a reference to an **int**, and *c* is a **char**:

```
f(int&);
char c;
f(c);
```

Instead of calling *f* with the address of *c*, the compiler generates code equivalent to the C++ source code:

```
int X = c, f(X);
```

Terminated by user *Linker error*
You canceled the link.

The '...' handler must be last *Compiler error*
In a list of catch handlers, if the '...' handler is present, it must be the last handler in the list (that is, it cannot be followed by any more catch handlers).

The combinations '+*' or '*+' are not allowed *Librarian error*
It is not legal to add and extract an object module from a library in one action. The action probably desired is a '+-'.

The constructor *constructor* is not allowed *Compiler error*
Constructors of the form **X::(X)** are not allowed. The correct way to write a copy constructor is **X::(const X&)**.

The value for *identifier* is not within the range of an int *Compiler error*
All enumerators must have values that can be represented as an integer. You attempted to assign a value that is out of the range of an integer. In C++ if you need a constant of this value, use a **const** integer.

'this' can be used only within a member function *Compiler error*
In C++, **this** is a reserved word that can be used only within class member functions.

This initialization is only partly bracketed *Compiler warning*
When structures are initialized, braces can be used to mark the initialization of each member of the structure. If a member itself is an array or structure, nested pairs of braces can be used. When some of the optional braces are omitted, the compiler issues this warning.

Too few arguments in template class name *template* *Compiler error*
A template class name was missing actual values for some of its formal parameters.

Too few parameters in call *Compiler error*
A call to a function with a prototype (via a function pointer) had too few arguments. Prototypes require that all parameters be given.

Too few parameters in call to *function* *Compiler error*
A call to the named function (declared using a prototype) had too few arguments.

Too many commas on command-line *Linker error*
An invalid entry in the command-line was found.

Too many decimal points *Compiler error*
The compiler encountered a floating-point constant with more than one decimal point.

Too many default cases *Compiler error*
The compiler encountered more than one **default** statement in a single **switch**.

Too many default libraries *Compiler error*
The linker can handle a maximum of 128 default libraries.

Too many error or warning messages *Compiler error*
A maximum of 255 errors and warnings can be set before the compiler stops.

Too many error or warning messages *Linker error*
The number of messages reported by the compiler has exceeded its limit. This error indicates that TLINK reached its limit.

Too many errors *Linker error*
The linker encountered more errors than the **–E** switch will permit.

Too many exponents *Compiler error*
The compiler encountered more than one exponent in a floating-point constant.

Too many initializers *Compiler error*
The compiler encountered more initializers than were allowed by the declaration being initialized.

Too many LNAMEs *Linker error*
TLINK has a limit of 256 LNAMES in a single .OBJ file.

Too many rules for target *target* *MAKE error*
MAKE can't determine which rules to follow when building a target because you've created too many rules for the target. For example, the following makefile generates this error message:

```
abc.exe : a.obj
   bcc -c a.c

abc.exe : b.obj

abc.exe : c.obj
   bcc -c b.c c.c
```

Too many storage classes in declaration *Compiler error*
A declaration can never have more than one storage class.

Too many suffixes in .SUFFIXES list *MAKE error*
The limit of 255 allowable suffixes in the suffixes list has been exceeded.

Too many types in declaration *Compiler error*
A declaration can never have more than one of the basic types: **char**, **int**, **float**, **double**, **struct**, **union**, **enum**, or **typedef**-*name*.

Too much global data defined in file *Compiler error*
The sum of the global data declarations exceeds 64K bytes. Check the declarations for any array that might be too large. Also consider reorganizing the program or using **far** variables if all the declarations are needed.

Trying to derive a far class from the huge base *base* *Compiler error*
If a class is declared (or defaults to) **huge**, all derived classes must also be **huge**.

Trying to derive a far class from the near base *base* *Compiler error*
If a class is declared (or defaults to) **near**, all derived classes must also be **near**.

Trying to derive a huge class from the far base *base* *Compiler error*
If a class is declared (or defaults to) **far**, all derived classes must also be **far**.

Trying to derive a huge class from the near base *base* *Compiler error*
If a class is declared (or defaults to) **near**, all derived classes must also be **near**.

Trying to derive a near class from the far base *base* *Compiler error*

If a class is declared (or defaults to) **far**, all derived classes must also be **far**.

Trying to derive a near class from the huge base *base* *Compiler error*

If a class is declared (or defaults to) **huge**, all derived classes must also be **huge**.

Two consecutive dots *Compiler error*

Because an ellipsis contains three dots (**...**), and a decimal point or member selection operator uses one dot (**.**), two consecutive dots cannot legally occur in a C program.

Two operands must evaluate to the same type *Compiler error*

The types of the expressions on both sides of the colon in the conditional expression operator (**?:**) must be the same, except for the usual conversions like **char** to **int**, or **float** to **double**, or **void*** to a particular pointer. In this expression, the two sides evaluate to different types that are not automatically converted. This might be an error or you might merely need to cast one side to the type of the other.

Type *type* is not a defined class with virtual functions *Compiler error*

A dynamic_cast was used with a pointer to a class type that is either undefined or doesn't have any virtual member functions.

Note on type-mismatch errors: When compiling C++ programs, the following type-mismatch error messages are always preceded by another message that explains the exact reason for the type mismatch; this is usually "Cannot convert *type1* to *type2*" but the mismatch could also be due to many other reasons.

Type mismatch in default argument value *Compiler error*
Type mismatch in default value for parameter *parameter* *Compiler error*

The default parameter value given could not be converted to the type of the parameter. The first message is used when the parameter was not given a name. See the previous note on type-mismatch errors.

Type mismatch in parameter *number* *Compiler error*

The function called, via a function pointer, was declared with a prototype; the given parameter *number* (counting left to right from 1) could not be converted to the declared parameter type. See the previous note on type-mismatch errors.

Type mismatch in parameter *number* in call to *function* *Compiler error*

Your source file declared the named function with a prototype, and the given parameter *number* (counting left to right from 1) could not be converted to the declared parameter type. See the previous note on type-mismatch errors.

Type mismatch in parameter *parameter* *Compiler error*

Your source file declared the function called via a function pointer with a prototype, and the named parameter could not be converted to the declared parameter type. See the previous note on type-mismatch errors.

Type mismatch in parameter *parameter* in call to *function* *Compiler error*

Your source file declared the named function with a prototype, and the named parameter could not be converted to the declared parameter type. See entry for **Type mismatch in parameter *parameter*** and the previous note on type-mismatch errors.

Type mismatch in parameter *parameter* in template class name *template* *Compiler error*
Type mismatch in parameter *number* in template class name *template* *Compiler error*

The actual template argument value supplied for the given parameter did not exactly match the formal template parameter type. See the previous note on type-mismatch errors.

Type mismatch in redeclaration of *identifier* *Compiler error*

Your source file redeclared with a different type than was originally declared. This can occur if a function is called and subsequently declared to return something other than an integer. If this has happened, you must declare the function before the first call to it. See the previous note on type-mismatch errors.

Type name expected *Compiler error*

One of these errors has occurred:

- In declaring a file-level variable or a **struct** field, neither a type name nor a storage class was given.
- In declaring a **typedef**, no type for the name was supplied.
- In declaring a destructor for a C++ class, the destructor name was not a type name (it must be the same name as its class).
- In supplying a C++ base class name, the name was not the name of a class.

Type qualifier *identifier* must be a struct or class name *Compiler error*

The C++ qualifier in the construction `qual::identifier` is not the name of a **struct** or **class**.

Unable to create output file *filename* *Compiler error*

The work disk is full or write-protected or the output directory does not exist. If the disk is full, try deleting unneeded files and restarting the compilation. If the disk is write-protected, move the source files to a writable disk and restart the compilation.

Unable to create turboc.$ln *Compiler error*

The compiler cannot create the temporary file TURBOC.$LN because it cannot access the disk or the disk is full.

Unable to execute command: *command* *MAKE error*

A command failed to execute; this might be because the command file could not be found or was misspelled, because there was no disk space left in the specified swap directory, because the swap directory does not exist, or (less likely) because the command itself exists but has been corrupted.

Unable to execute command *command* *Compiler error*

TLINK or TASM cannot be found, or possibly the disk is bad.

Unable to open file *filename* *MAKE error*
Unable to open *filename* *Linker error*

This occurs if the named file does not exist or is misspelled.

Unable to open *filename* for output *Librarian error*

The librarian cannot open the specified file for output. This is usually due to lack of disk space for the target library, or a listing file.

Unable to open include file *filename* *Compiler error*

The compiler could not find the named file. This error can also be caused if an #**include** file included itself, or if you do not have FILES set in CONFIG.SYS on your root directory (try `FILES=20`). Check whether the named file exists.

Unable to open include file *filename* *MAKE error*

The compiler could not find the named file. This error can also be caused if an **!include** file included itself, or if you do not have FILES set in CONFIG.SYS on your root directory (try `FILES=20`). Check whether the named file exists.

Unable to open input file *filename* *Compiler error*

This error occurs if the source file cannot be found. Check the spelling of the name and whether the file is on the proper disk or directory.

Unable to open makefile *MAKE error*

The current directory does not contain a file named MAKEFILE or MAKEFILE.MAK, or it does not contain the file you specified with **–f**.

Unable to redirect input or output *MAKE error*

MAKE was unable to open the temporary files necessary to redirect input or output. If you are on a network, make sure you have rights to the current directory.

Unable to rename *filename* to *filename* *Librarian error*

The librarian builds a library into a temporary file and then renames the temporary file to the target library file name. If there is an error, usually due to lack of disk space, this message is posted.

Undefined label *identifier* *Compiler error*

The named label has a **goto** in the function, but no label definition.

Undefined structure *identifier* *Compiler warning*

The named structure was used in the source file, probably on a pointer to a structure, but had no definition in the source file. This is probably caused by a misspelled structure name or a missing declaration.

Undefined structure *structure* *Compiler error*

Your source file used the named structure on some line before where the error is indicated (probably on a pointer to a structure) but had no definition for the structure. This is probably caused by a misspelled structure name or a missing declaration.

Undefined symbol *identifier* *Compiler error*

The named identifier has no declaration. This could be caused by a misspelling either at this point or at the declaration. This could also be caused if there was an error in the declaration of the identifier.

Undefined symbol *symbol* *Linker error*

The named symbol is referenced in the given module but is not defined anywhere in the set of object files and libraries included in the link. Check to make sure the symbol is spelled correctly.

You will usually see this error from TLINK for Borland C++ symbols if you did not properly match a symbol's declarations of **pascal** and **cdecl** type in different source files, or if you have omitted the name of an .OBJ file your program needs. If you are linking C++ code with C modules, you might have forgotten to wrap C external declarations in `extern "C" {...}`. You might have a case mismatch between two symbols. See the **/C** and **/c** switches.

Unexpected } *Compiler error*

An extra right brace was encountered where none was expected. Check for a missing **{**.

Unexpected char *X* in command line *Librarian error*

The librarian encountered a syntactical error while parsing the command line.

Unexpected end of file *MAKE error*

The end of the makefile was reached without a temporary inline file having been closed.

Unexpected end of file in comment started on *line number* *MAKE error*

The source file ended in the middle of a comment. This is normally caused by a missing close of comment (*/).

Unexpected end of file in conditional started on line *line number* *Compiler error*

The source file ended before the compiler (or MAKE) encountered an **!endif**. The **!endif** was either missing or misspelled.

Union cannot be a base type *Compiler error*

A union cannot be used as a base type for another class type.

Union cannot have a base type *Compiler error*

A union cannot be derived from any other class.

Union member *member* is of type class with constructor *Compiler error*
Union member *member* is of type class with destructor *Compiler error*
Union member *member* is of type class with operator= *Compiler error*

A union cannot contain members that are of type **class** with user-defined constructors, destructors, or operator=.

Unions cannot have virtual member functions *Compiler error*
A union cannot have virtual functions as its members.

Unknown assembler instruction *Compiler warning*
The compiler encountered an inline assembly statement.

Unknown command line switch *X* ignored *Librarian warning*
A forward slash character (/) was encountered on the command line or in a response file without being followed by one of the allowed options.

Unknown language, must be C or C++ *Compiler error*
In the C++ construction

```
extern "name" type func( /*...*/ );
```

the name given in quotes must be "C" or "C++"; other language names are not recognized. For example, you can declare an external Pascal function without having the compiler rename it like this:

```
extern "C" int pascal func( /*...*/ );
```

A C++ (possibly overloaded) function can be declared Pascal and allow the usual compiler renaming (to allow overloading) like this:

```
extern int pascal func( /*...*/ );
```

Unknown option –> *option* *Linker error*
A forward slash character (/), hyphen (–), or DOS switch character was encountered on the command line or in a response file without being followed by one of the allowed options. You might have used the wrong case to specify an option.

Unknown preprocessor directive: *identifier* *Compiler error*
The compiler encountered a # character at the beginning of a line, and the name following was not a legal directive name or the rest of the directive was not well formed.

Unknown preprocessor statement *MAKE error*
A ! character was encountered at the beginning of a line, and the statement name following was not **error, undef, if, elif, include, else,** or **endif.**

Unreachable code *Compiler warning*
A **break, continue, goto** or **return** statement was not followed by a label or the end of a loop or function. The compiler checks **while, do** and **for** loops with a constant test condition, and attempts to recognize loops that cannot fall through.

Unsupported option *string* *Linker error*
You have specified an invalid option to the linker.

Unterminated string or character constant *Compiler error*
The compiler found no terminating quote after the beginning of a string or character constant.

Use '> >' for nested templates instead of '>>' *Compiler warning*
Whitespace is required to separate the closing ">" in a nested template name, but since it is a common mistake to leave out the space, the compiler accepts a ">>" with this warning.

Use . or –> to call *function* *Compiler error*
You tried to call a member function without giving an object.

Use . or –> to call *member*, or & to take its address *Compiler error*
A reference to a nonstatic class member without an object was encountered. Such a member must be used with an object, or its address must be taken with the & operator.

Use :: to take the address of a member function *Compiler error*

If **f** is a member function of class *c*, you take its address with the syntax **&c::f.** Note the use of the class type name, rather than the name of an object, and the **::** separating the class name from the function name. (Member function pointers are not true pointer types, and do not refer to any particular instance of a class.)

Use /e with TLINK to obtain debug information from library *Librarian warning*

The library was built with an extended dictionary and also includes debugging information. TLINK will not extract debugging information if it links using an extended dictionary, so to obtain debugging information in an executable from this library, the linker must be told to ignore the extended dictionary using the **/e** switch. *Note*: The IDE linker does *not* support extended dictionaries; therefore no settings need to be altered in the IDE.

Use of : and :: dependents for target *target* *MAKE error*

You have tried to use the target in both single and multiple description blocks (using both the **:** and **::** operators). Examples:

```
filea: fileb
filea:: filec
```

Use qualified name to access nested type *type* *Compiler warning*

In older versions of the C++ specification, typedef and tag names declared inside classes were directly visible in the global scope. With the latest specification of C++, these names must be prefixed with a **class::** qualifier if they are to be used outside their class' scope. To allow older code to compile, whenever such a name is uniquely defined in one single class, Borland C++ allows its usage without **class::** and issues this warning.

User break *Compiler error*

You pressed *Ctrl+Break* while compiling or linking in the IDE, thus aborting the process. (This is not an error, just a confirmation.)

Value of type void is not allowed *Compiler error*

A value of type **void** is really not a value at all, and thus cannot appear in any context where an actual value is required. Such contexts include the right side of an assignment, an argument of a function, and the controlling expression of an **if**, **for**, or **while** statement.

VIRDEF Name Conflict for *function* *Compiler error*

The compiler must truncate mangled names to a certain length because of a name length limit that is imposed by the linker. This truncation might (in rare cases) cause two names to mangle to the same linker name. If these names happen to both be **VIRDEF** names, the compiler issues this error message. The simplest workaround for this problem is to change the name of *function* so that the conflict is avoided.

Variable *identifier* is initialized more than once *Compiler error*

This variable has more than one initialization. It is legal to declare a file level variable more than once, but it can have only one initialization (even if two are the same).

'virtual' can only be used with member functions *Compiler error*

A data member has been declared with the **virtual** specifier; only member functions can be declared **virtual.**

Virtual function *function1* conflicts with base class *base* *Compiler error*

The compiler encountered a virtual function that has the same argument types as a function in its base class, but the two functions have different return types. This is illegal.

Virtual specified more than once *Compiler error*

The C++ reserved word **virtual** can appear only once in a member function declaration.

void & is not a valid type *Compiler error*

A reference always refers to an object, but an object cannot have the type **void**. Thus the type **void** is not allowed.

Void functions may not return a value *Compiler warning*

Your source file declared the current function as returning **void**, but the compiler encountered a return statement with a value. The value of the return statement will be ignored.

function* was previously declared with the language *language *Compiler error*

Only one language can be used with **extern** for a given function. This function has been declared with different languages in different locations in the same module.

While statement missing (*Compiler error*

In a **while** statement, the compiler found no left parenthesis after the **while** keyword.

While statement missing) *Compiler error*

In a **while** statement, the compiler found no right parenthesis after the test expression.

This occurs if TLINK could not write all of the data it attempted to write. This is almost certainly caused by the disk being full.

Write error on file *filename* *MAKE error*

MAKE couldn't open or write to the file specified in the makefile. Check to ensure that there's enough space left on your disk, and that you have write access to the disk.

Wrong number of arguments in call of macro *mac* *Compiler error*

Your source file called the named macro with an incorrect number of arguments.

A P P E N D I X B

Borland Windows Custom Controls

Before reading this appendix you should be familiar with dialog box controls and the Dialog editor. For a description of these topics, see Chapter 15.

The Borland Windows Custom Controls (BWCC) library contains a custom dialog class and a set of custom dialog controls (buttons, check boxes, group shading boxes, and the like). BWCC adds to the visual impact of your dialog boxes and optimizes their functionality.

Two of the online files included with Resource Workshop provide additional information about BWCC:

- BWCCAPI.RW provides technical information about the BWCC application program interface.
- BWCCSTYL.RW provides some style suggestions for designing Borland-style dialog boxes.

Using the Borland custom dialog class

The custom dialog class, BORDLG, works on both a visual and a functional level:

- It improves the appearance of your dialog window by painting the background with a brush that varies according to the target display device. For screens of VGA and higher resolution, the background is a fine grid of perpendicular white lines, giving the effect of "chiseled steel." For EGA and monochrome screens, the background is white.
- It optimizes the drawing of dialog boxes by calling the custom control drawing routines directly instead of waiting for Windows to paint the controls. This eliminates the typically sluggish drawing of dialog boxes.

To use the custom dialog class,

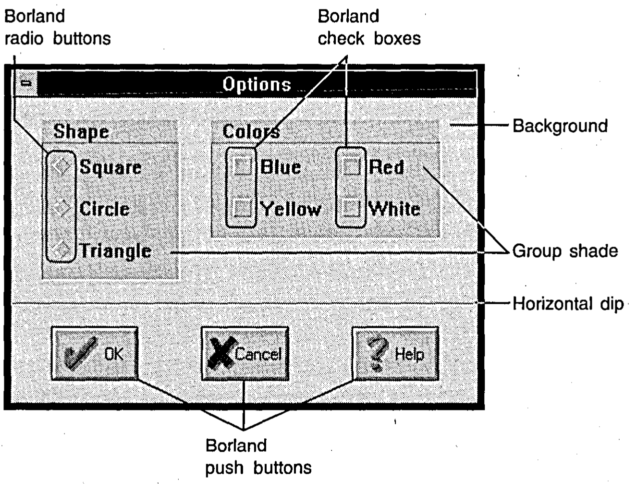
1. Open the dialog resource you want to convert.
2. Double-click the title bar of the dialog to display the Window Style dialog box.
3. Enter "bordlg" as the Class and click OK.

Using Borland controls

Borland controls add a three-dimensional effect to your dialog boxes and give them more visual impact. To the end-user, they appear to function in the same manner as the standard Windows controls, although they include several technical enhancements (described later).

The following figure shows a dialog box converted to BWCC. It uses several Borland controls.

Figure B.1
Dialog box with Borland controls



The following list briefly describes each Borland control and shows its corresponding tool icon. As with standard Windows controls, you can insert Borland controls in your dialogs by picking them from the Tools palette in the Dialog editor.

The description of each control includes its class. To see the class and other settings of any of these controls, display the Generic Control Style dialog box by holding down the *Ctrl* key and double-clicking on the control.

Group shade	A shaded rectangular box that groups other controls visually. It can appear recessed into the dialog box or raised above its surface. Its class is *BorShade.*
Horizontal dip	A horizontal dividing line that gives the impression of being etched into the surface of the dialog box. (You can convert a dip to a bump that appears to be raised above the surface of the dialog box.) Its class is *BorShade.*
Vertical dip	Same as horizontal dip, except it's vertical. Its class is *BorShade.*
Borland push button	A family of push buttons with symbols that have high visual impact, plus an owner-draw option. The Borland push buttons are larger than most standard Windows push buttons. Their class is *BorBtn.*
Borland radio button	A raised, diamond-shaped radio button. When the button is clicked, a black diamond appears in its center and the button shading reverses, giving the impression that the button has been pushed down. There is also an owner-draw option. Its class is *BorRadio.*
Borland check box	A raised check box that displays a check mark instead of an "X." There is also an owner-draw option. Its class is *BorCheck.*
Borland static text	A fixed text string used principally for labeling parts of the dialog box. Its class is *BorStatic.*

Button and check box enchancements

The Borland push buttons, radio buttons, and check boxes have the following functional enhancements over standard Windows controls:

- An additional level of parent window notification and control over keyboard focus and tab movement. If you choose the Parent Notify option in the control's style dialog box, the control sends the appropriate messages from the following list at run time:
  - BBN_SETFOCUS indicates to the parent window that the push button, radio button, or check box has gained keyboard focus through an action other than a mouse click.
  - BBN_SETFOCUSMOUSE indicates to the parent window that the push button, radio button, or check box has gained keyboard focus through a mouse click.
  - BBN_GOTATAB indicates to the parent window that the user has pressed the Tab key while the push button, radio button, or check box

has keyboard focus. The parent can intervene in the processing of the keystroke by returning a nonzero value.

- BBN_GOTABTAB indicates to the parent window that the user has pressed Shift-Tab (back-tab) while the push button, radio button, or check box has keyboard focus. The parent can intervene in the processing of the keystroke by returning a nonzero value.

■ An owner-draw option that allows the parent window to draw the push button, radio button, or check box. Because your application handles drawing the control, it won't necessarily look like a Borland control, but it will have the standard behavior of that class of control.

Using the BWCC style dialog boxes

Four dialog boxes set the style of the BWCC controls:

■ Borland Button Style
■ Borland Radio Button Style
■ Borland Check Box Style
■ Borland Shade Style

To display one of the Style dialog boxes, double-click on the control whose style you want to set.

Each has a control window for entering a caption and a control ID. The button style, radio button style, and check box style dialog boxes have Attributes options for Tab Stop, Disabled, Group, Visible, and Border, as well as Parent Notify and Owner Draw (described earlier in this appendix).

The next four sections describe the features unique to each of the style dialog boxes.

Borland Button Style dialog box

This dialog box lets you choose from the three button types: Pushbutton, Defpushbutton, and Bitmap.

Pushbutton and Defpushbutton

By default, Pushbutton is the selected option. A Defpushbutton has a bold border to identify it to the end-user as the *default button,* which is executed when the user presses the Enter key. (The one exception occurs when keyboard focus is in an Edit Text control for which the Want Return flag has been set. See page 247 for a description of the Want Return flag.)

When you first place a Borland button in your dialog box, its text is Button and it takes the next available control ID. To change the button to one of the standard Borland buttons, change the control ID to one of the preset values in the following table:

Table B.1 Predefined BWCC button controls

ID name	ID value	Type	Image
IDOK	1	OK	Green check mark
IDCANCEL	2	Cancel	Red X
IDABORT	3	Abort	Panic button
IDRETRY	4	Retry	Slot machine
IDIGNORE	5	Ignore	55 mph speed-limit sign
IDYES	6	Yes	Green check mark
IDNO	7	No	Red circle and slash
IDHELP	998	Help	Blue question mark

Bitmap

If you choose the Bitmap option, you can insert a bitmap image (based on its control ID) into the button. To read in a bitmap:

1. Use the Button control to add the generic BWCC button to your dialog box. Note its control ID.
2. Switch to the Bitmap Editor and create a bitmap image. (See Chapter 19 for information about creating bitmaps.)
3. In the Bitmap Editor, choose Resource | Rename to display the Rename Resource dialog box and then do either of the following:
   - In the New Name text box, enter an integer value that equals the control ID of the button plus the appropriate offset from Table B.2.
   - Rename the bitmap and then assign it an identifier whose value equals the control ID of the button plus the appropriate offset from Table B.2. (Creating identifiers is described in Chapter 14.)

The bitmap won't display in the Dialog editor until you close the Bitmap Editor.

4. Close the Bitmap Editor.
5. Return to the Dialog Editor. If the bitmap doesn't immediately appear in the BWCC button, resize the button. The bitmap should then appear.

Table B.2 Bitmap offsets

Button state	Offset for VGA/higher	Offset for EGA/monochrome
Standard	1000	2000
Pressed	3000	4000
Keyboard focus	5000	6000

For example, to display the keyboard focus bitmap for a button whose control ID is 276, enter 5276 for a VGA system or 6276 for an EGA system.

Borland Radio Button Style dialog box

This dialog box lists two button styles:

- *Radio button*. Highlighting and deselection don't happen automatically. The application must call the **CheckRadioButton** function to send a BM_SETCHECK message to highlight the selected button and deselect the other buttons.
- *Auto radio button*. BWCC and Windows combine to handle highlighting the selected button and deselecting the other buttons. This is the default option.

Borland Check Box Style dialog box

This dialog box lists four check box styles:

- *Check box*. The box is not checked automatically. The application must call the **CheckDlgButton** function to send a BM_SETCHECK message to check the selected box.
- *Auto check box*. BWCC and Windows combine to handle checking the selected box. This is the default option.
- *3-state*. The box is not checked automatically. The application must call the **CheckDlgButton** function to send a BM_SETCHECK message to check the selected box.

  The button's three states are on, off, and "indeterminate," which is displayed as a checkerboard pattern. The application determines what is meant by "indeterminate."
- *Auto 3-state*. BWCC and Windows combine to handle checking the selected box.

Borland Shade Style dialog box

This dialog box sets the style for controls you add with any of these three tools: Group Shade, Horizontal Dip, and Vertical Dip. Using the Shade Style radio buttons, you can make the following conversions:

- *Group Shade to Raised Shade*. Group shades and raised shades are used to enclose controls with related functions—like radio buttons and check boxes. Group shades appear recessed below the surface of the dialog box; raised shades appear raised above the surface.
- *Horizontal Dip to Horizontal Bump, Vertical Dip to Vertical Bump*. Dips are intended to act as separators in the dialog box background or in raised shades; bumps are intended as separators in recessed gray shade boxes.

Borland Static Text Style dialog box

Use this dialog box to enter the text and set attributes and control style for Borland static text.

In addition to the standard attributes (Disabled, Group, and Visible), static text has two additional attributes.

- When the Border option is checked, the static text is surrounded by a standard Windows border that uses the current color for the Window Frame (see the Windows Control Panel).
- When the No Underline option is checked, an ampersand (&) appears as a literal character, instead of underlining the next character.

The Control Style options are described in Table 15.12 on page 248. There is one significant difference, however: *all* BWCC static text, including Simple Text, uses the standard BWCC gray background.

Modifying existing applications for BWCC

Resource Workshop lets you modify existing Windows applications with Borland-style custom controls (3D buttons, dialog boxes with the "chiseled steel" look, and so on). There are two steps to this process:

The sections that follow describe these steps in greater detail.

1. Modify your WIN.INI file to load the Borland Windows Custom Control (BWCC) library each time you start Windows.
2. Edit the application in Resource Workshop to change user interface features like dialog boxes, menus, icons, and so on.

Loading BWCC

The BWCC library, which provides support for Borland-style custom controls, must be loaded before an application can use BWCC's features.

Edit the WIN.INI file (located in the Windows main directory) so that Windows loads the file LOADBWCC.EXE into memory at start up. (The installation program puts LOADBWCC.EXE into the compiler's executable file directory and adds this directory to your PATH.)

Add LOADBWCC.EXE to the beginning of the list of files that appears after the "LOAD=" statement. It must appear first in the statement to ensure that BWCC is loaded into memory before any modified applications are executed.

For example, if the LOAD statement in your WIN.INI file is `LOAD=NWPOPUP.EXE AD.EXE`, the statement must be changed to `LOAD=loadbwcc.exe NWPOPUP.EXE AD.EXE`.

Using BWCC in C and C++ programs

If you use the Borland C++ IDE, check BWCC when you create a project target (see Chapter 2 for information on projects in the IDE). You'll also need to add `BWCCGetVersion();` to WinMain.

If you don't use the IDE, you must do all of the following:

- Add a **#include** for BWCC.H to your .C or .CPP file.
- Add BWCC.LIB to your C or C++ IDE project, or insert it in the TLINK Library area before IMPORT.LIB.
- Add `BWCCGetVersion();` to WinMain.

Tips on editing resources

This section discusses considerations to keep in mind when editing resources of existing applications.

- **Accelerators** If you add an accelerator, make sure it returns the same ID value as its corresponding menu command. If you don't, the accelerator will either execute the wrong command or do nothing.
- **Bitmaps, cursors, and icons** You can modify existing bitmaps, cursors, and icons. Don't delete bitmaps, cursors, or icons, and don't try to add new ones. In most cases the application won't be able to use them.
- **Dialog boxes** You can reposition items in a dialog box and convert controls to their Borland custom control counterparts. As you edit, be sure not to change the type of control associated with each control ID value. For example, if control ID 100 is a check box, don't change it to a radio button, because the application will still treat it as a check box.

  In most cases you can remove controls that aren't directly tied to the application's functionality. For example, you can usually remove a caption, a static text item that has no effect on how the application works. Don't remove an edit control; it *does* affect how the application works. Don't add new controls; the application won't be able to use them.
- **Menus** With most applications, you can safely move commands within a menu. Don't, however, move commands from one menu to another. (For example, don't move the Open command from the File menu to the Edit menu.) If you do, the application might be unable to display context-sensitive Help or to check or uncheck the menu commands. Never change the order of the menus in the menu bar. For example, if File is the first menu, don't make it the second.
- **String tables** Use caution when editing existing string tables. Some programs load the strings into buffers of fixed size, and adding text to an existing string could cause the buffer to overflow. Don't add new strings; the application won't be able to use them.

A P P E N D I X C

Precompiled headers

Borland C++ can generate and subsequently use precompiled headers for your projects. Precompiled headers can greatly speed up compilation times.

How they work

When compiling large C and C++ programs, the compiler can spend up to half of its time parsing header files. When the compiler parses a header file, it enters declarations and definitions into its symbol table. If ten of your source files include the same header file, this header file is parsed ten times, producing the same symbol table every time.

Precompiled header files cut this process short. During one compilation, BCC.EXE stores an image of the symbol table on disk in a file called BCDEF.CSM by default (BC32DEF.CSM for BCC32.EXE). (BCDEF.CSM is stored in the same directory as the compiler.) Later, when the same source file is recompiled (or another source file that includes the same header files), the compiler reloads BCDEF.CSM from disk instead of parsing all the header files again. Directly loading the symbol table from disk is over ten times faster than parsing the text of the header files.

Precompiled headers are used only if the second compilation uses one or more of the same header files, the same compiler options, defined macros and so on, as the first compilation.

If, while compiling a source file, Borland C++ discovers that the first **#include**s are identical to those of a previous compilation (of the same source or a different source), it loads the binary image for those **#include**s and parses the remaining **#include**s.

For a given module, either all or none of the precompiled headers are used: if compilation of any included header file fails, the precompiled header file isn't updated for that module.

Drawbacks

When using precompiled headers, BCDEF.CSM can become very big, because it contains symbol table images for all sets of includes encountered in your sources. If you don't have sufficient disk space, you'll get a warning saying the write failed because of the precompiled headers. To fix this, free more disk space and retry the compile. For information on reducing the size of the BCDEF.CSM file, see "Optimizing precompiled headers" on page 405.

If you're using large macros in a makefile in addition to using precompiled headers, there is a limit on the macro size: 4K for 16-bit applications and 16K for 32-bit applications.

If a header contains any code, it can't be precompiled. For example, although C++ class definitions can appear in header files, you should ensure that only inline member functions are defined in the header and heed warnings such as "Functions containing for are not expanded inline".

Using precompiled headers

You can control the use of precompiled headers in any of the following ways:

- From within the IDE, using the Project Options dialog box. The IDE bases the name of the precompiled header file on the project name, creating *PROJECT*.CSM.
- From the command line, using the **–H**, **–H=*filename*,–Hc**, **–H"filename"** and **–Hu** options (see page 61).
- From within your code, using the pragmas **hdrfile** and **hdrstop** (see Chapter 5 in the *Programmer's Guide*).

Setting file names

The compiler uses just one file to store all precompiled headers. The default file name is BCDEF.CSM. You can explicitly set the name with the **–H=*filename*** command-line option or the #pragma**hdrfile** directive.

Caution! If you notice that your .CSM file is smaller than it should be, the compiler might have run out of disk space when writing to the .CSM file. When this happens, the compiler deletes the .CSM to make room for the .OBJ file, then starts creating a new (and therefore shorter) .CSM file. If this happens, free up some disk space before compiling.

Establishing identity

The following conditions must be identical for a previously generated precompiled header to be loaded for a subsequent compilation.

The second or later source file must:

- Have the same set of include files in the same order
- Have the same macros defined to identical values
- Use the same language (C or C++)
- Use header files with identical time stamps; these header files can be included either directly or indirectly

In addition, you must compile the subsequent source file using the same settings for the following options (for example, if you compiled the first file with the small model, the second file must be compiled as small):

- Memory model, including SS != DS (**-m*x***)
- Underscores on externs (**-u**)
- Maximum identifier length (**-iL**)
- Target DOS (default) or Windows (**-W** or **–W*x***)
- DOS overlay-compatible code (**-Y**)
- Virtual table control (**-V*x*** and **–Vm*x***)
- Expand intrinsic functions inline (**-Oi**)
- Templates (**-J*x***)
- String literals in code segment (**-dc**, 16-bit only)
- Debugging information (**-v**, **–vi**, and **–R**)
- Far variables (**-Fx**)
- Generate word alignment (**-a**)
- Pascal calls (**-p**)
- Treat enums as integers (**-b**)
- Default char is unsigned (**-K**)
- Language compliance (**-A**)
- C++ compile (**-P**)

Optimizing precompiled headers

For Borland C++ to most efficiently compile using precompiled headers, follow these rules:

- Arrange your header files in the same sequence in all source files.
- Put the largest header files first.
- Prime BCDEF.CSM with often-used initial sequences of header files.

- Use #pragma **hdrstop** to terminate the list of header files at well-chosen places. This lets you make the list of header files in different sources look similar to the compiler. #pragma **hdrstop** is described in more detail in Chapter 5 in the *Programmer's Guide.*

For example, given the two source files ASOURCE.C and BSOURCE.C, both of which include windows.h and myhdr.h,

ASOURCE.C

```
#include <windows.h>
#include "myhdr.h"
#include "xxx.h"
<...>
```

BSOURCE.C

```
#include "zz.h"
#include <string.h>
#include "myhdr.h"
#include <windows.h>
<...>
```

you would rearrange the beginning of BSOURCE.C to:

Revised BSOURCE.C

```
#include <windows.h>
#include "myhdr.h"
#include "zz.h"
#include <string.h>
<...>
```

Note that windows.h and myhdr.h are in the same order in BSOURCE.C as they are in ASOURCE.C. You could also make a new source called PREFIX.C containing only the header files, like this:

PREFIX.C

```
#include <windows.h>
#include "myhdr.h"
```

If you compile PREFIX.C first (or insert a `#pragma hdrstop` in both ASOURCE.C and BSOURCE.C after the `#include "myhdr.h"` statement) the net effect is that after the initial compilation of PREFIX.C, both ASOURCE.C and BSOURCE.C will be able to load the symbol table produced by PREFIX.C. The compiler will then only need to parse xxx.h for ASOURCE.C and zz.h and string.h for BSOURCE.C.

A P P E N D I X **D**

Using EasyWin

EasyWin is a feature of Borland C++ that lets you compile standard DOS applications that use traditional "TTY style" input and output so they run as true Windows programs. EasyWin does not require you to make any changes to a DOS program in order to run it under Windows.

ConsoleDOS to Windows made easy

To convert your console-based applications that use standard files or iostream functions, check the EasyWin Target Type in TargetExpert in the IDE. If you are using the command-line compiler, use the compiler switch **–W**. Borland C++ notes that your program doesn't contain a *WinMain* function (normally required for Windows applications) and links the EasyWin library. When you run your program in Windows, a standard window is created, and your program takes input and produces output for that window exactly as if it were the standard screen.

Example C program:

```
#include <stdio.h>
main()
{
   printf("Hello, world\n");
   return 0;
}
```

Example C++ program:

```
#include <iostream.h>
main()
{
   cout << "Hello, world\n";
   return 0;
}
```

The EasyWin window can be used any time input or output is requested from or to a TTY device. This means that in addition to stdin and stdout, the stderr, stdaux, and cerr "devices" are all connected to this window.

_InitEasyWin( )

EasyWin's purpose is to convert DOS applications to Windows programs, quickly and easily. However, you might occasionally want to use EasyWin from within a "true" Windows program. For example, you might want to add *printf* functions to your program code to help you debug your Windows program.

To use EasyWin from within a Windows program, make a call to the *_InitEasyWin* function before doing any standard input or output.

You can find the prototype for *_InitEasyWin* in stdio.h and iostream.h.

For example:

```
#include <windows.h>
#include <stdio.h>

#pragma argsused
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                   LPSTR lpszCmdLine, int cmdShow)
{
   _InitEasyWin();

   /* Normal windows setup */
   printf("Hello, world\n");
   return 0;
}
```

Added functions

For your convenience, EasyWin also includes five additional functions that let you specify the X and Y window coordinates for input and output, clear the window or clear to the end of the current line:

See the *Library Reference*, Chapter 3, for a description of the functions available to EasyWin programs.

clreol	(conio.h)	*wherex*	(conio.h)
clrscr	(conio.h)	*wherey*	(conio.h)
gotoxy	(conio.h)		

These functions have the same names (and uses) as functions in conio.h header file. Classes in constrea.h provide conio functionality for use with C++ streams. See the *Programmer's Guide,* Chapter 6, for a complete discussion of constreams and iostreams.

The following routines are portable to EasyWin programs but are not available in 16-bit Windows programs. They are provided to ease porting of existing code into a Windows 16-bit application.

fgetchar	(stdio.h)	*printf*	(stdio.h)
getch	(stdio.h)	*putch*	(conio.h)
getchar	(stdio.h)	*putchar*	(stdio.h)
getche	(stdio.h)	*puts*	(stdio.h)
gets	(stdio.h)	*scanf*	(stdio.h)
kbhit	(conio.h)	*vprintf*	(stdio.h)
perror	(errno.h)	*vscanf*	(stdio.h)

Index

Borland

Corporate Headquarters: 100 Borland Way, Scotts Valley, CA 95066-3249, (408) 431-1000. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom • Part # BCP1240WW21770 • BOR 6270