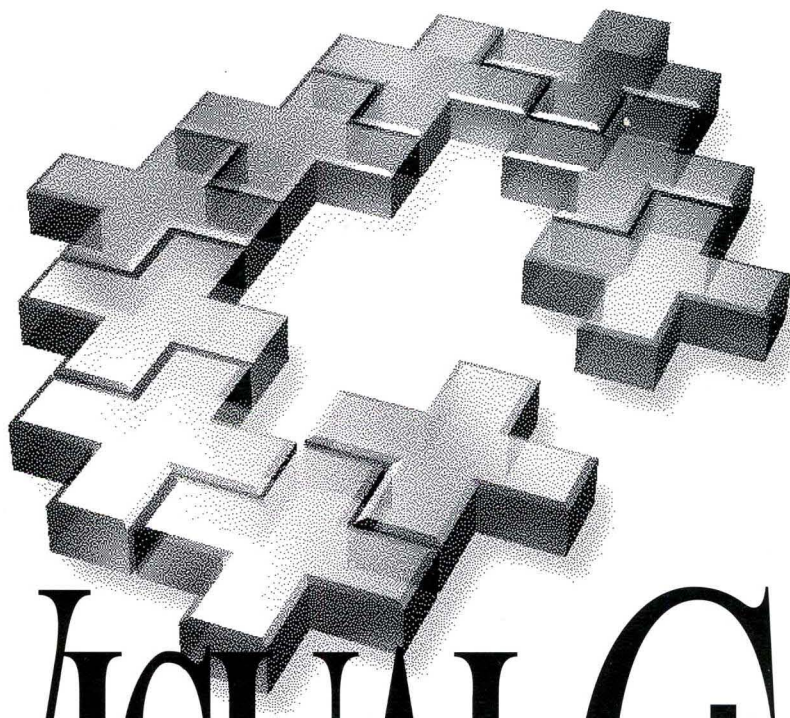


# User's Guide

*Visual Workbench User's Guide*

*App Studio User's Guide*



# Microsoft VISUAL C++ STANDARD EDITION

*Development System for Windows™*

# User's Guides

## **Microsoft® Visual C++™**

**Development System for Windows™**  
**Version 1.0**

This volume contains two separate books:  
Visual Workbench User's Guide  
App Studio User's Guide

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

©1993 Microsoft Corporation. All rights reserved.

Microsoft, MS, MS-DOS, CodeView, FoxPro, Multiplan, PowerPoint, QuickC, and QuickPascal are registered trademarks and Windows, Microsoft Access, QuickAssembler, Microsoft QuickBasic, Visual Basic, and Visual C++ are trademarks of Microsoft Corporation in the USA and other countries.

U.S. Patent No. 4955066

CompuServe is a registered trademark of CompuServe, Inc.  
Flight Simulator is a registered trademark of Bruce Artwick.  
IBM is a registered trademark of International Business Machines Corporation.  
Macintosh is a registered trademark of Apple Computer, Inc.  
OS/2 is a registered trademark licensed to Microsoft.  
Paintbrush is a trademark of ZSoft Corporation.

# Visual Workbench User's Guide





# Contents

<b>Introduction</b> . . . . .	<b>xi</b>
<b>Microsoft Support Services</b> . . . . .	<b>xv</b>

## Part 1 Getting Started

<b>Chapter 1 Installing Microsoft Visual C++</b> . . . . .	<b>3</b>
Microsoft Visual C++ Overview . . . . .	3
What Is Visual C++? . . . . .	3
Comparison of Standard and Professional Editions . . . . .	4
System Requirements . . . . .	5
Installation . . . . .	5
Installation Options . . . . .	7
Setting Command-Line Options for Visual Workbench . . . . .	9
MS-DOS Configuration . . . . .	10
<b>Chapter 2 Building a Sample Application for Windows</b> . . . . .	<b>13</b>
Introducing Projects . . . . .	13
A Sample Windows-Based C++ Application . . . . .	14
<b>Chapter 3 Building a Sample QuickWin Program</b> . . . . .	<b>17</b>
What Is QuickWin? . . . . .	17
A Sample QuickWin Program . . . . .	18
<b>Chapter 4 Developing a Microsoft Visual C++ Application</b> . . . . .	<b>21</b>
What Is a Visual C++ Application? . . . . .	21
Introducing the Tools . . . . .	22
The Applications . . . . .	22
The Wizards . . . . .	23
The Process . . . . .	26
The Application Creation Stage . . . . .	26
The Application Development Stage . . . . .	27
For More Information . . . . .	34
<b>Chapter 5 Fast Track to Visual Workbench</b> . . . . .	<b>35</b>
Menu Summaries . . . . .	36
Managing Files and Printing . . . . .	36
Editing Files . . . . .	37

Window Display and Quick Access . . . . .	38
Working with Projects . . . . .	39
Accessing Symbols and Classes . . . . .	41
Debugging Programs . . . . .	42
Running Tools from Visual Workbench . . . . .	43
Setting Preferences and Custom Options . . . . .	44
Arranging and Opening Windows . . . . .	45
Getting Online Help . . . . .	46
Key Summaries . . . . .	47
Editing Keys . . . . .	47
Toolbar Keys . . . . .	49
Window Management Keys . . . . .	50
Build and Compile Keys . . . . .	50
Browsing Keys . . . . .	50
Debugging Keys . . . . .	51
Alphabetic Guide to Build Options . . . . .	51
Compiler Options . . . . .	51
Linker Options . . . . .	55

## Part 2 Using Visual Workbench

<b>Chapter 6 The Visual Workbench Environment . . . . .</b>	<b>59</b>
Starting and Quitting Visual Workbench . . . . .	60
Visual Workbench Features . . . . .	60
The Toolbar . . . . .	60
The Status Bar . . . . .	63
Arranging and Displaying Windows . . . . .	63
Getting Help . . . . .	65
<b>Chapter 7 Using the Editor . . . . .</b>	<b>75</b>
Managing Source Files . . . . .	75
Creating and Saving Source Files . . . . .	76
Opening and Closing Source Files . . . . .	79
Opening Resource Files . . . . .	80
Moving Around in Files . . . . .	82
Using the Keyboard Commands . . . . .	84
Controlling the Source Window . . . . .	84
Setting Tabs . . . . .	85
Highlighting Language Syntax . . . . .	85
Making a File Read-Only . . . . .	86

---

Finding and Replacing . . . . .	86
Finding Text Using the Find Command. . . . .	87
Finding Text Using the Toolbar . . . . .	88
Replacing Text. . . . .	89
Printing . . . . .	90
<b>Chapter 8 Using Projects . . . . .</b>	<b>93</b>
Project Types . . . . .	94
Creating a Project . . . . .	97
Opening and Closing Projects. . . . .	100
Adding and Deleting Project Files. . . . .	100
Using Include Dependencies . . . . .	101
Project Compiler and Linker Options . . . . .	103
Building a Project . . . . .	103
Using a Workspace . . . . .	104
Using the Last Workspace Used. . . . .	105
Saving a Workspace . . . . .	105
Loading a Workspace . . . . .	106
Loading an Initial Workspace . . . . .	106
Using External Projects . . . . .	107
<b>Chapter 9 Customizing Build Options. . . . .</b>	<b>111</b>
The Compiler and Linker Options Dialog Boxes . . . . .	111
Getting Help on Options . . . . .	113
Default Compiler and Linker Options . . . . .	115
Compiler Options . . . . .	118
Code Generation . . . . .	118
Custom Options . . . . .	121
Custom Options (C++) . . . . .	124
Debug Options. . . . .	127
.Listing Files . . . . .	129
Memory Model . . . . .	130
Optimizations. . . . .	132
P-Code Generation. . . . .	136
Precompiled Headers . . . . .	138
Preprocessor . . . . .	142
Segment Names . . . . .	144
Windows Prolog/Epilog . . . . .	146

Linker Options . . . . .	148
Input . . . . .	149
Memory Image . . . . .	151
Miscellaneous . . . . .	153
Output . . . . .	155
Windows Libraries . . . . .	157
Resource Compiler Options . . . . .	158
Disable Load Optimization . . . . .	159
Custom Options . . . . .	159
Defines . . . . .	159
<b>Chapter 10 Using the Browser . . . . .</b>	<b>161</b>
Creating a Browser Database . . . . .	161
Opening a Browser Database . . . . .	163
Querying a Browser Database . . . . .	165
Using the Browse Window . . . . .	166
Using Menu Commands . . . . .	168
Browsing Classes and Functions . . . . .	169
Overview of Graphical Browser Query Types . . . . .	169
Expanding and Collapsing Graphs . . . . .	171
Browsing Classes . . . . .	172
Browsing Function Relationships . . . . .	174
Browsing Definitions and References . . . . .	176
<b>Chapter 11 Debugging Programs . . . . .</b>	<b>179</b>
Using the Debugging Windows . . . . .	180
Debugging During Building . . . . .	181
Using the Visual Workbench Debugger . . . . .	182
Preparing a Debug Version of a Program . . . . .	182
Setting and Removing Breakpoints . . . . .	183
Controlling Program Execution . . . . .	189
Using the Watch Window . . . . .	190
Using QuickWatch . . . . .	192
Modifying a Variable . . . . .	193
Using Show Call Stack . . . . .	194
Using the Registers Window . . . . .	195
Other Debugging Features . . . . .	196
Hard/Soft Mode Debugging . . . . .	196
Debug Display Options . . . . .	198

---

<b>Chapter 12 Customizing Visual Workbench</b> . . . . .	<b>199</b>
Modifying the Tools Menu . . . . .	199
Adding Commands to the Tools Menu . . . . .	199
Editing a Tools Menu Command . . . . .	201
Tips For Using MS-DOS Tools . . . . .	201
Using Argument Macros . . . . .	202
Setting Directories. . . . .	203
Changing Syntax Coloring . . . . .	205
Making Global Display Changes . . . . .	205
Source File Syntax Coloring . . . . .	206
Setting Font Type and Size. . . . .	207
<b>Chapter 13 Using Visual Workbench with Other Visual C++ Tools</b> . . . . .	<b>209</b>
Using AppWizard . . . . .	209
Opening and Closing AppWizard. . . . .	210
Specifying the Project Name and Location . . . . .	210
Selecting Options . . . . .	212
Modifying Classes . . . . .	214
AppWizard-Generated Files . . . . .	215
Running App Studio . . . . .	218
Using ClassWizard . . . . .	220
Creating New Classes. . . . .	221
Creating Message Handlers for Windows Messages . . . . .	223
<b>Index</b> . . . . .	<b>225</b>

# Figures and Tables

## Figures

Figure 1.1	The Installation Options Dialog Box . . . . .	6
Figure 2.1	The Open Project Dialog Box . . . . .	14
Figure 2.2	The Project Options Dialog Box . . . . .	14
Figure 3.1	QuickWin C++ Program Code . . . . .	19
Figure 3.2	The Project Options Dialog Box . . . . .	20
Figure 3.3	QWINTTEST Output. . . . .	20
Figure 4.1	Visual Workbench . . . . .	22
Figure 4.2	App Studio . . . . .	23
Figure 4.3	AppWizard . . . . .	24
Figure 4.4	ClassWizard . . . . .	25
Figure 4.5	Creating the Visual C++ Framework Code. . . . .	26
Figure 4.6	Creating and Editing User-Interface Objects. . . . .	28
Figure 4.7	Connecting User-Interface Objects to Code . . . . .	29
Figure 4.8	The Visual C++ Development Process. . . . .	30
Figure 4.9	The Visual Workbench Browse Window . . . . .	31
Figure 4.10	The Visual Workbench Debug Windows . . . . .	33
Figure 6.1	The Visual Workbench Environment . . . . .	59
Figure 6.2	Cascaded Windows . . . . .	64
Figure 6.3	Tiled Windows . . . . .	64
Figure 6.4	Minimized Windows . . . . .	65
Figure 6.5	The Secondary and Primary Help Windows . . . . .	68
Figure 6.6	The Visual Workbench Help Contents Screen. . . . .	69
Figure 6.7	A Visual Workbench Topic Screen . . . . .	70
Figure 6.8	A Reference Help Contents Screen . . . . .	71
Figure 7.1	The Save As Dialog Box . . . . .	77
Figure 7.2	The Editor Dialog Box . . . . .	79
Figure 7.3	The Editor Dialog Box . . . . .	85
Figure 7.4	The Find Dialog Box . . . . .	87
Figure 7.5	The Replace Dialog Box. . . . .	89
Figure 7.6	The Print Dialog Box . . . . .	90
Figure 7.7	A Printer Setup Dialog Box . . . . .	91
Figure 8.1	The Edit - <i>Projectname</i> Dialog Box . . . . .	99
Figure 8.2	The Workspace Dialog Box . . . . .	107
Figure 8.3	The External Project Options Dialog Box. . . . .	108

---

Figure 9.1	The C/C++ Compiler Options Dialog Box	112
Figure 9.2	Help on the Options String	114
Figure 9.3	Help on Option Controls	115
Figure 9.4	Compiler Options: Code Generation	118
Figure 9.5	Compiler Options: Custom Options	122
Figure 9.6	Compiler Options: Custom Options (C++)	124
Figure 9.7	Compiler Options: Debug Options	127
Figure 9.8	Compiler Options: Listing Files	129
Figure 9.9	Compiler Options: Memory Model	130
Figure 9.10	Compiler Options: Optimizations	132
Figure 9.11	Compiler Options: P-Code Generation	136
Figure 9.12	Compiler Options: Precompiled Headers	138
Figure 9.13	Compiler Options: Preprocessor	143
Figure 9.14	Compiler Options: Segment Names	145
Figure 9.15	Compiler Options: Windows Prolog/Epilog	147
Figure 9.16	The Linker Options Dialog Box	148
Figure 9.17	Linker Options: Input	149
Figure 9.18	Linker Options: Memory Image	151
Figure 9.19	Linker Options: Miscellaneous	154
Figure 9.20	Linker Options: Output	155
Figure 9.21	Linker Options: Windows Libraries	157
Figure 9.22	Resource Compiler Options Dialog Box	158
Figure 10.1	C/C++ Compiler Options: Listing Files	162
Figure 10.2	The Browse Window	164
Figure 10.3	Browse Window Query Group	166
Figure 10.4	Class Graphs	170
Figure 10.5	Call and Caller Graphs	170
Figure 10.6	Expanded and Collapsed Nodes	171
Figure 10.7	Derived Class Graph of the CWnd Class	173
Figure 10.8	Base Class Graph of the CDialog Class	174
Figure 10.9	A Call Graph	175
Figure 11.1	Debugging Windows	181
Figure 11.2	The Project Options Dialog Box	182
Figure 11.3	Setting a Breakpoint in SORTDEMO.C	184
Figure 11.4	The Breakpoints Dialog Box	185
Figure 11.5	The Messages Dialog Box	188
Figure 11.6	The Watch Window	191
Figure 11.7	The Watch Window with Expanded and Collapsed Variables	192
Figure 11.8	The QuickWatch Dialog Box	192



Figure 11.9 The Modify Variable Dialog Box . . . . .	194
Figure 11.10 The Call Stack Dialog Box . . . . .	195
Figure 11.11 The Registers Window . . . . .	195
Figure 12.1 The Tools Dialog Box . . . . .	200
Figure 12.2 The Add Tool Dialog Box. . . . .	200
Figure 12.3 The Directories Dialog Box . . . . .	204
Figure 12.4 The Color Dialog Box . . . . .	205
Figure 12.5 The Font Dialog Box . . . . .	207
Figure 13.1 The MFC AppWizard Dialog Box. . . . .	211
Figure 13.2 AppWizard's Options Dialog Box. . . . .	212
Figure 13.3 AppWizard's Classes Dialog Box . . . . .	214
Figure 13.4 ClassWizard . . . . .	220
Figure 13.5 Classes Derived from CCmdTarget . . . . .	221
Figure 13.6 ClassWizard's Add Class Dialog Box . . . . .	222
Figure 13.7 Creating Message Handlers for Windows Messages . . . . .	224

## Tables

Table 5.1 Insertion Point Movement Keys . . . . .	47
Table 5.2 Text Selection Keys . . . . .	47
Table 5.3 Insert, Copy, and Tab Keys . . . . .	48
Table 5.4 Delete Keys . . . . .	48
Table 5.5 Text Scrolling Keys . . . . .	48
Table 5.6 Search Keys . . . . .	49
Table 5.7 Toolbar Keys . . . . .	49
Table 5.8 Visual Workbench Window Management Keys . . . . .	50
Table 5.9 Build and Compile Keys . . . . .	50
Table 5.10 Browsing Keys . . . . .	50
Table 5.11 Debugging Keys . . . . .	51
Table 5.12 Alphabetic Guide to Compiler Options . . . . .	52
Table 5.13 Alphabetic Guide to Linker Options . . . . .	55
Table 6.1 Toolbar Buttons . . . . .	62
Table 9.1 Default Compiler Options for Windows Project Types . . . . .	116
Table 9.2 Default Compiler Options for MS-DOS Project Types . . . . .	116
Table 9.3 Default Linker Options for Windows Project Types . . . . .	117
Table 9.4 Default Linker Options for MS-DOS Project Types . . . . .	117
Table 10.1 Wildcard Types in Symbol Names . . . . .	167
Table 11.1 Debugging Windows. . . . .	180
Table 11.2 Debug Menu Commands . . . . .	189
Table 11.3 Additional Debugging Features . . . . .	198
Table 12.1 Visual Workbench Argument Macros . . . . .	202

# Introduction

Microsoft® Visual C++™ represents an evolutionary step in the progress of high-performance program development tools for the Microsoft Windows™ operating system. This comes about because of two milestones: the use of fully integrated Windows-hosted development tools and the adaptation of the popular “visual” user-interface-driven paradigm to the traditional C/C++ development process.

Microsoft Visual Workbench is the cornerstone of the Visual C++ development platform. It is a powerful development environment on its own, containing several integrated tools, including an editor, debugger, and graphical browser. But Visual Workbench also fits into the larger development strategy by acting as the central point from which all other development activities are performed. From Visual Workbench menus you can:

- Invoke the Visual C++ build tools, such as the compiler and linker.
- Run the App Studio resource editor to develop user-interface components.
- Run AppWizard and ClassWizard to help you develop Microsoft Foundation Class Library applications.
- Run your own tools, which can be installed on the Tools menu.

Visual Workbench strikes the balance of giving you the tools you need most and, at the same time, making them easy to use by keeping things simple.

## About This Manual

This book describes how to install Visual C++ and how to use the Visual Workbench integrated development environment to develop applications using Visual C++. There are two parts to this book.

Part 1, “Getting Started,” helps you install Visual C++, become familiar with Visual Workbench, and understand the general roles and relationships of the main Visual C++ development tools. It contains the following chapters:

- Chapter 1 describes how to install Visual C++ and how to reinstall specific components later. It also provides the basic configuration and system requirements for Visual C++, as well as configuration requirements for running Visual C++ from the command line.
- Chapters 2 and 3 give you a quick guide through building and running two sample programs: a Microsoft Foundation Class Library sample application and a simple QuickWin C++ application. You can use these short chapters to verify that your installation is working and to become familiar with Visual Workbench.

- Chapter 4 helps you get acquainted with the concepts and tools behind the Visual C++ development process, which is designed to make developing Microsoft Foundation Class Library applications easier than ever.
- Chapter 5 provides a quick reference guide to the Visual Workbench features. It contains several tables listing menu commands and common procedures associated with them, toolbar buttons, and shortcut keys. It also has an alphabetical listing of compiler and linker options that is cross-referenced to the dialog boxes used in setting these options.

Part 2, “Using Visual Workbench,” provides the detailed information and procedures to help you use Visual Workbench. It contains the following chapters:

- Chapter 6 describes the basic components of the environment, such as the toolbar, status bar, and Help system.
- Chapter 7 describes how to use the editor to create, open, save, and close files, edit text, move around in files, set tabs, and search for text, among other editing tasks.
- Chapters 8 and 9 provide all the information you need to use projects, configure compiler and linker options, and build and run your applications. Chapter 8 describes creating and using projects to build your applications as well as the use of workspaces to save and recall environment configurations. Chapter 9 is the complete reference to the dialog boxes used to set compiler and linker options.
- Chapter 10 describes the Visual Workbench browser. This chapter tells you how to query the browser database from both the Browse window and from within a source file, and how to work with graphical data displayed for C++ class graphs and C or C++ function graphs.
- Chapter 11 provides information on the Visual Workbench integrated debugger, which is compatible with the Microsoft CodeView® debugger. The chapter tells you how to set breakpoints, monitor variables and expressions in the QuickWatch dialog box, Watch window, and Locals windows, monitor registers, step through code, and more.
- Chapter 12 tells you how to customize Visual Workbench syntax coloring and font settings, how to install tools on the Tools menu, and how to set the directories used during building, browsing, and debugging.
- Chapter 13 brings together Visual Workbench and the other main Visual C++ tools associated with it. It tells you how to open App Studio from within Visual Workbench, how to use AppWizard to generate Microsoft Foundation Class Library skeleton project files, and how to use ClassWizard features you are likely to encounter most often from within Visual Workbench.

# Document Conventions

This book uses the following typographic conventions:

Example	Description
STDIO.H	Uppercase letters indicate filenames, segment names, registers, and terms used at the operating-system command level.
<b>char, _setcolor, __far</b>	Bold type indicates C and C++ keywords, operators, language-specific characters, and library routines. Within discussions of syntax, bold type indicates that the text must be entered exactly as shown.  Many functions and constants begin with either a single or double underscore. These are part of the name and are mandatory. For example, to have the <b>__cplusplus</b> manifest constant be recognized by the compiler, you must enter the leading double underscore.
<i>expression</i>	Words in italics indicate placeholders for information you must supply, such as a filename. Italic type is also used occasionally for emphasis in the text.
<code>[[option]]</code>	Items inside double square brackets are optional.
<code>#pragma pack {1   2}</code>	Braces and a vertical bar indicate a choice among two or more items. You must choose one of these items unless double square brackets ([[ ]]) surround the braces.
<code>#include &lt;io.h&gt;</code>	This font is used for examples, user input, program output, and error messages in text.
<code>CL [[option...]]file...</code>	Three dots (an ellipsis) following an item indicate that more items having the same form may appear.
<code>while() { . . . }</code>	A column or row of three dots tells you that part of an example program has been intentionally omitted.
CTRL+ENTER	Small capital letters are used to indicate the names of keys on the keyboard. When you see a plus sign (+) between two key names, you should hold down the first key while pressing the second.  The carriage-return key, sometimes marked as a bent arrow on the keyboard, is called ENTER.
“argument”	Quotation marks enclose a new term the first time it is defined in text.

<b>Example</b>	<b>Description</b>
"C string"	Some C constructs, such as strings, require quotation marks. Quotation marks required by the language have the form " " and ' ' rather than “ ” and ‘ ’.
Color Graphics Adapter (CGA)	The first time an acronym is used, it is usually spelled out.

---

# Microsoft Support Services

Microsoft offers a variety of no-charge and fee-based support options to help you get the most from your Microsoft products. For an explanation of these options, please see one of the following sections:

- If you are in the United States, see “Support Services Within the United States.”
- If you are outside the United States, see “Support Services Worldwide.”

## Support Services Within the United States

If you have a question about Microsoft Visual C++, one of the following resources may help you find an answer:

- The index in the product documentation and other printed product documentation.
- Context-sensitive online Help available from the Help menu.
- The README files that come with your product disks. These files provide general information that became available after the books in the product package were published.
- Electronic options such as CompuServe forums or the Microsoft Download Service.

If you cannot find the information you need, you can obtain product support through several methods as discussed below. In addition, information is provided about training and consulting services available to you.

For information about Microsoft incremental and annual fee-based support service options, call Microsoft Inside Sales at (800) 227-4679, extension 11700, Monday through Friday, between 6:30 AM and 5:30 PM Pacific time.

---

**Note** Microsoft’s support services are subject to Microsoft’s prices, terms, and conditions in place in each country at the time the services are used.

---

## Microsoft Forums on CompuServe

Microsoft Product Support Services is available on several CompuServe forums. The Microsoft Developer Services Area includes forums and information designed specifically for developers. To access the Microsoft Developer Services

Area, type G0 MSDS at any ! prompt. Some of the services available within MSDS are:

- The Developer Specific Knowledge Base contains up-to-date technical information about Microsoft developer-specific products compiled by Microsoft developer support engineers. The knowledge base is full-text searchable and includes information on helpful tips, product bug lists, fix lists, documentation errors, and Microsoft press releases. To access the Developer Specific Knowledge Base, type G0 MDKB at any ! prompt.
- The Microsoft Software Library is a collection of binary files, sample code, technical specifications, and utilities. The entire library is keyword-searchable and the files can be downloaded for use locally. To access the Microsoft Software Library, type G0 MSL at any ! prompt.
- Developer forums covering programming issues and other developer-specific information on the Windows operating system, languages, tools, and utilities are available. You can use these forums to exchange messages with Microsoft developer support engineers and experienced users of Microsoft development products. For example, the MSLANG forum provides support and information on Microsoft Visual C++ and other Microsoft language products and the WINSDK forum provides information about developing in the Windows environment.

For an introductory CompuServe membership kit specifically for Microsoft users, dial (800) 848-8199 and ask for operator 230.

## Microsoft Product Support Services

You can reach Microsoft Product Support Services, Monday through Friday, between 6:00 AM and 6:00 PM Pacific time.

- We offer phone support that is best utilized for getting you up and running at (206) 635-7007.

When you call, you should be at your computer with Microsoft Visual C++ running and the product documentation at hand. Be prepared to give the following information:

- The version of Microsoft Visual C++ you are using.
- The type of hardware you are using, including network hardware, if applicable.
- The operating system you are using.
- The exact wording of any messages that appeared on your screen and the error number, if any.

- A description of what happened and what you were trying to do when the problem occurred.
- A description of how you tried to solve the problem.

## Microsoft Download Service

Microsoft Product Support Services operates a download bulletin board service (BBS) that contains application notes, drivers, utilities, and other useful information. The phone number is (206) 936-6735, and the BBS is accessible seven days a week, from 2:30 AM to 1:00 AM Pacific time. To access the Microsoft Download Service, you'll need a modem and terminal emulation software. For 1200, 2400, and 9600 baud modems, set parity to none, data bits to 8, and stop bits to 1.

## Microsoft FastTips

Microsoft FastTips is an automated system you can access by touch-tone telephone that provides recorded answers to common questions about some Microsoft products. You can also receive copies of this information on a fax machine. In addition, FastTips offers a library of comprehensive technical notes that can be faxed or mailed. FastTips is available 7 days a week, 24 hours a day, including holidays.

- For assistance with Microsoft Visual C++, dial (206) 635-4694.

After you reach FastTips, use the following keys to move through the automated system:

- To advance to the next message, press the ASTERISK (\*) key.
- To repeat the current message, press the 7 key.
- To return to the beginning of FastTips, press the POUND SIGN (#) key.

## Microsoft Developer Services Team

For help with nontechnical questions related to Microsoft developer products, services, and support, call the Developer Services Team at (800) 227-4679, extension 11771, Monday through Friday, between 6:30 AM and 5:30 PM Pacific time. Service representatives can help you get started with Microsoft development tools, provide information on beta test programs, and inform you about upcoming Microsoft conferences, events, and training courses. You can request timely, informative literature on many developer-related topics and products, purchase development kits for products, and sign up for fee-based technical support programs.



## Microsoft Product Support for the Deaf and Hard-of-Hearing

Microsoft Product Support Services is available for the deaf and hard-of-hearing, Monday through Friday, between 6:00 AM and 6:00 PM Pacific time.

Using a special TDD/TT modem, dial (206) 635-4948.

## Product Training and Consultation Services

Within the United States, Microsoft offers the following services for training and consultation.

### Consultant Referral Service

Microsoft's Consultant Relations Program can refer you to an independent consultant in your area. These consultants are skilled in:

- Macro development and translation.
- Database development.
- Custom interface design.

For information about the consultants in your area, call the Microsoft Consultant Relations Program at (800) 227-4679, extension 56042, Monday through Friday, between 6:30 AM and 5:30 PM Pacific time.

### Microsoft Consulting Services

Microsoft Consulting Services (MCS) assists management and technical staff through all phases of a project: effective planning, rapid technology transfer, high-quality design, and integration into an organization's existing business systems. Microsoft consultants are available to work on projects of all sizes. MCS also provides on-site consultants who serve as full-time members of client development and support teams.

- Call MCS West at (415) 905-0235, Monday through Friday, between 8:00 AM and 5:00 PM Pacific time, or MCS East at (617) 487-6400, Monday through Friday, between 8:00 AM and 5:30 PM Eastern time.

### Microsoft University

Microsoft University (MSU) provides hands-on courses to help programmers and support engineers minimize their learning curves and maximize their ability to take full advantage of the latest Microsoft systems software. MSU technical training is for software developers, support and systems engineers, network

administrators, information system (IS) managers, and data-processing professionals.

- Call MSU at (206) 828-1507, Monday through Friday, between 6:30 AM and 5:00 PM Pacific time.

## Support Services Worldwide

If you are outside the United States and have a question about Microsoft Visual C++, Microsoft offers a variety of no-charge and fee-based support options. To solve your problem, you can:

- Consult the index in the product documentation and other printed product documentation.
- Check context-sensitive online Help available from the Help menu.
- Check the README files that come with your product disks. These files provide general information that became available after the books in the product package were published.
- Consult electronic options such as CompuServe forums or bulletin board systems, if available.

If you cannot find a solution, you can receive information on how to obtain product support by contacting the Microsoft subsidiary office that serves your country.

---

**Note** Microsoft's support services are subject to Microsoft's prices, terms, and conditions in place in each country at the time the services are used.

---

## Calling a Microsoft Subsidiary Office

When you call, you should be at your computer with Microsoft Visual C++ running and the product documentation at hand. Be prepared to give the following information:

- The version of Microsoft Visual C++ you are using.
- The type of hardware you are using, including network hardware, if applicable.
- The operating system you are using.
- The exact wording of any messages that appeared on your screen.
- A description of what happened and what you were trying to do when the problem occurred.
- A description of how you tried to solve the problem.

Microsoft subsidiary offices and the countries they serve are listed below.

Area	Telephone Numbers
Argentina	Microsoft de Argentina S.A. Phone: (54) (1) 814-0356 Fax: (54) (1) 814-0372
Australia	Microsoft Pty. Ltd. Phone: (61) (02) 870-2200 Fax: (02) 805-1108 Bulletin Board Service: (612) 870-2348 Technical Support: (61) (02) 870-2131 Sales Information Centre: (02) 870-2100
Austria	Microsoft Ges.m.b.H. Phone: 0222 - 68 76 07 Fax: 0222 - 68 16 2710 Information: 060 - 89 - 247 11 101 Prices, updates, etc.: 060 - 89 - 3176 1199 CompuServe: msce (Microsoft Central Europe) Technical support: Windows, Windows for Workgroups, Microsoft Mail: 0660 - 65 - 10 Microsoft Excel for Windows, Microsoft Excel for OS/2®, PowerPoint® for Windows: 0660 - 65 - 11 Word for MS-DOS®, Windows Write: 0660 65 - 12 Word for Windows, Word for OS/2: 0660 - 65 - 13 Works for MS-DOS, Works for Windows, Publisher, WorksCalc, WorksText: 0660 - 65 - 14 C PDS, FORTRAN PDS, Pascal, Macro Assembler PDS, QuickC®, QuickC for Windows, QuickPascal®, QuickAssembler™, Profiler: 0660 - 65 - 15 COBOL PDS, Basic PDS, Microsoft QuickBasic™, Visual Basic™: 0660 - 65 - 16 MS-DOS: 0660 - 65 - 17 Macintosh Software: 0660 - 65 - 18 Microsoft Project for Windows, Microsoft Project for MS-DOS, Multiplan®, Mouse, Flight Simulator, Paintbrush, Chart: 0660 - 67 - 3 FoxPro®, Microsoft Access™: 0660 - 67 - 61
Baltic States	See Germany
Belgium	Microsoft NV Phone: 02-7322590 Fax: 02-7351609 Technical Support Bulletin Board Service: 02-7350045 (1200/2400/9600 baud, 8 bits, no parity, 1 stop bit, ANSI terminal emulation) (Dutch speaking) Technical Support: 02-5133274 (English speaking) Technical Support: 02-5023432 (French speaking) Technical Support: 02-5132268 Technical Support Fax: (31) 2503-24304
Bermuda	See Venezuela
Bolivia	See Argentina

---

<b>Area</b>	<b>Telephone Numbers</b>
Brazil	Microsoft Informatica Ltda. Phone: (55) (11) 530-4455 Fax: (55) (11) 240-2205 Technical Support Phone: (55) (11) 533-2922 Technical Support Fax: (55) (11) 241-1157 Technical Support Bulletin Board Service: (55) (11) 543-9257
Canada	Microsoft Canada Inc. Phone: 1 (416) 568-0434 Fax: 1 (416) 568-4689 Technical Support Phone: 1 (416) 568-3503 Technical Support Facsimile: 1 (416) 568-4689 Technical Support Bulletin Board Service: 1 (416) 507-3022
Caribbean Countries	See Venezuela
Central America	See Venezuela
Chile	Microsoft Chile S.A. Ave. Presidente Kennedy 5146, Las Condes Santiago, Chile Tel: 56 2 218 5771 Fax: 56 2 218 5747
Colombia	Microsoft Columbia Carrera 9 # 99-02 Piso 2 Bogota, D.C., Colombia Tel: (571) 618 2245 Soporte Tecnico: (571) 618 2255 Fax:(571) 618 2269
Denmark	Microsoft Denmark AS Phone: (45) (44) 89 01 00 Fax: (45) (44) 68 55 10
Ecuador	See Venezuela
England	Microsoft Limited Phone: (44) (734) 270000 Fax: (44) (734) 270002 Upgrades: (44) (81) 893-8000 Technical Support: Main Line (All Products): (44) (734) 271000 Windows Direct Support Line: (44) (734) 271001 Database Direct Support Line: (44) (734) 271126 MS-DOS 5 Warranty Support: (44) (734) 271900 MS-DOS 5 Fee Support Line: (44) (891) 315500 OnLine Service Assistance: (44) (734) 270374 Bulletin Board Service: (44) (734) 270065 (2400 Baud) Fax Information Service: (44) (734) 270080

Area	Telephone Numbers
Finland	Microsoft OY Phone: (358) (0) 525 501 Fax: (358) (0) 522 955
France	Microsoft France Phone: (33) (1) 69-86-46-46 Telex: MSPARIS 604322F Fax: (33) (1) 64-46-06-60 Technical Support Phone: (33) (1) 69-86-10-20 Technical Support Fax: (33) (1) 69-28-00-28
French Polynesia	See France
Germany	Microsoft GmbH Phone: 089 - 3176-0 Telex: (17) 89 83 28 MS GMBH D Fax: 089 - 3176-1000 Information: 0130 - 5099 Prices, updates, etc.: 089 - 3176 1199 Bulletin board, device drivers, tech notes : BTX: microsoft# or *610808000# CompuServe: msce (Microsoft Central Europe) Technical support: Windows, Windows for Workgroups, Microsoft Mail: 089 - 3176 - 1110 Microsoft Excel for Windows, Microsoft Excel for OS/2, PowerPoint for Windows: 089 - 3176 - 1120 Word for MS-DOS, Windows Write: 089 - 3176 - 1130 Word for Windows, Word for OS/2: 089 - 3176 - 1131 Works for MS-DOS, Works for Windows, Publisher, WorksCalc, WorksText: 089 - 3176 - 1140 C PDS, FORTRAN PDS, Pascal, Macro Assembler PDS, QuickC, QuickC for Windows, QuickPascal, QuickAssembler, Profiler: 089 - 3176 - 1150 COBOL PDS, Basic PDS, Microsoft QuickBasic, Visual Basic: 089 - 3176 - 1151 MS-DOS: 089 - 3176 - 1152 Macintosh Software: 089 - 3176 - 1160 Microsoft Project for Windows, Microsoft Project for MS-DOS, Multiplan, Mouse, Flight Simulator, Paintbrush, Chart: 089 - 3176 - 1170 FoxPro, Microsoft Access: 089 - 3176 - 1180
Hong Kong	Microsoft Hong Kong Limited Technical Support: (852) 804-4222 Fax: (852) 560-2217
Ireland	See England
Israel	Microsoft Israel Ltd. Tuval 34 Ramat-Gan 52522 Israel Phone: 972-3-575-7034 Fax: 972-3-752-7065

---

Area	Telephone Numbers
Italy	Microsoft SpA Phone: (39) (2) 269121 Telex: 340321 I Fax: (39) (2) 21072020 Technical Support: Microsoft Excel for Windows, Microsoft Project for Windows, Works for Windows: (39) (2) 26901361 Word, Works for MS-DOS: (39) (2) 26901362 Windows, PowerPoint, Publisher, Windows for Workgroups, Works : (39) (2) 26901363 Basic, COBOL, Visual Basic, MS-DOS-based, Microsoft Access, Fox Products: (39) (2) 26901364 C, FORTRAN, Pascal, Macro Assembler (MASM), and SDKs: (39) (2) 26901354 LAN Manager, SQL Server, Microsoft Mail, Microsoft Mail Gateways: (39) (2) 26901356
Japan	Microsoft Company Ltd. Phone: (81) (3) 3363-1200 Fax: (81) (3) 3363-1281 Technical Support: MS-DOS-based Applications: (81) (3) 3363-0160 Windows-based Applications: (81) (3) 3363-5040 Language Products (Microsoft C, Macro Assembler [MASM], QuickC): (81) (3) 3363-7610 Language Products (Basic, FORTRAN, Visual Basic, Quick Basic): (81) (3) 3363-0170 All Products Technical Support Fax: (81) (3) 3363-9901
Korea	Microsoft CH Phone: (82) (2) 552-9505 Fax: (82) (2) 555-1724 Technical Support: (82) (2) 563-9230
Liechtenstein	See Switzerland (German speaking)
Luxemburg	Microsoft NV Phone: (32) 2-7322590 Fax: (32) 2-7351609 Technical Support Bulletin Board Service: (31) 2503-34221 (1200/2400/9600 baud, 8 bits, No parity, 1 stop bit, ANSI terminal emulation) (Dutch speaking) Technical Support: (31) 2503-77877 (English speaking) Technical Support: (31) 2503-77853 (French speaking) Technical Support: (32) 2-5132268 Technical Support Fax: (31) 2503-24304
México	Microsoft México, S.A. de C.V. Phone: (52) (5) 325-0910 Fax: (52) (5) 280-0198 Technical Support: (52) (5) 325-0912 Sales: (52) (5) 325-0911

Area	Telephone Numbers
Netherlands	Microsoft BV Phone: 02503-13181 Fax: 02503-37761 Technical Support Bulletin Board Service: 02503-34221 (1200/2400/9600 baud, 8 bits, No parity, 1 stop bit, ANSI terminal emulation) (Dutch speaking) Technical Support: 02503-77877 (English speaking) Technical Support: 02503-77853 Technical Support Fax: 02503-24304
New Zealand	Technology Link Centre Phone: 64 (9) 358-3724 Fax: 64 (9) 358-3726 Technical Support Applications: 64 (9) 357-5575
Northern Ireland	See England
Norway	Microsoft Norway AS Phone: (47) (2) 95 06 65 Fax: (47) (2) 95 06 64 Technical Support: (47) (2) 18 35 00
Papua New Guinea	See Australia
Paraguay	See Argentina
Peru	See Venezuela
Portugal	MSFT, Lda. Phone: (351) 1 4412205 Fax: (351) 1 4412101
Puerto Rico	See Venezuela
Republic of China	Microsoft Taiwan Corp. Phone: (886) (2) 504-3122 Fax: (886) (2) 504-3121 Technical Support: (886) (2) 504-3188
Republic of Ireland	See England
Scotland	See England
Spain	Microsoft Iberica SRL Phone: (34) (1) 804-0000 Fax: (34) (1) 803-8310 Technical Support: (34) (1) 803-9960

---

Area	Telephone Numbers
Sweden	Microsoft AB Phone: (46) (8) 752 56 00 Fax: (46) (8) 750 51 58 Technical Support: Applications: (46) (8) 752 68 50 Development and Network products: (46) (8) 752 60 50 MS-DOS: (46) (071) 21 05 15 (SEK 4.55/min) Sales Support: (46) (8) 752 56 30 Bulletin Board Service: (46) (8) 750 47 42 Fax Information Service: (46) (8) 752 29 00
Switzerland	(German speaking) Microsoft AG Phone: 01 - 839 61 11 Fax: 01 - 831 08 69 Information: 0049 - 89 - 247 11 101 Prices, updates, etc.: 0049 - 89 - 3176 1199 CompuServe: msce (Microsoft Central Europe) Technical support: Windows, Windows for Workgroups, Microsoft Mail: 01 - 342 - 4085 Microsoft Excel for Windows, Microsoft Excel for OS/2, PowerPoint for Windows: 01 - 342 - 4082 Word for MS-DOS, Windows Write: 01 - 342 - 4083 Word for Windows, Word for OS/2: 01 - 342 - 4087 Works for MS-DOS, Works for Windows, Publisher, WorksCalc, WorksText: 01 - 342 - 4084 C PDS, FORTRAN PDS, Pascal, Macro Assembler PDS, QuickC, QuickC for Windows, QuickPascal, QuickAssembler, Profiler: 01 - 342 - 4036 COBOL PDS, Basic PDS, Microsoft QuickBasic, Visual Basic: 01 - 342 - 4086 MS-DOS: 01 - 342 - 2152 Macintosh Software: 01 - 342 - 4081 Microsoft Project for Windows, Microsoft Project for MS-DOS, Multiplan, Mouse, Flight Simulator, Paintbrush, Chart: 01 - 342 - 0322 FoxPro, Microsoft Access: 01 / 342 - 4121  (French speaking) Microsoft SA, office Nyon Phone: 022 - 363 68 11 Fax: 022 - 363 69 11 Technical support: 022 - 738 96 88
Uruguay	See Argentina
Venezuela	Corporation MS 90 de Venezuela S.A. Phone: 0058.2.914739 Fax: 0058.2.923835
Wales	See England





# Getting Started

---

Chapter 1	Installing Microsoft Visual C++ . . . . .	3
Chapter 2	Building a Sample Application for Windows . . . . .	13
Chapter 3	Building a Sample QuickWin Program . . . . .	17
Chapter 4	Developing a Microsoft Visual C++ Application . . . . .	21
Chapter 5	Fast Track to Visual Workbench. . . . .	35



# Installing Microsoft Visual C++

This chapter introduces Microsoft Visual C++ and its integrated development environment, Microsoft Visual Workbench. It outlines the minimum hardware and software requirements necessary to install and use Visual C++ and also points out the differences between Visual C++ Standard Edition and Visual C++ Professional Edition. You can use this chapter to install either version of Visual C++ or to update components.

## Microsoft Visual C++ Overview

Before installing Visual C++, you might find it useful to read a short overview of the compiler and development environment you are about to use. Also, you may be interested in the differences between Visual C++ Standard Edition and Visual C++ Professional Edition and how this relates to Visual Workbench, which is used by both versions.

## What Is Visual C++?

Microsoft Visual C++ is a tool for building and debugging Windows-based applications and libraries in an integrated Windows environment. Visual C++ makes it much easier to tackle the complex job of developing applications for Windows by incorporating high-level C++ application framework classes with integrated Windows-hosted development tools.

The Windows-hosted development tools used in Visual C++ include Visual Workbench, App Studio, AppWizard, ClassWizard, and several other utilities accessed from, or used by, Visual Workbench.

During the development process, you use AppWizard to create the C++ Microsoft Foundation Class Library source files for your project. To create and edit resources such as dialog boxes, menus, toolbars, and controls, you use App Studio. You use ClassWizard to add C++ framework code for classes and message maps for either resources or View and Document classes. For an overview of the development process, see Chapter 4, “Developing a Microsoft Visual C++ Application.”

## Comparison of Standard and Professional Editions

Visual C++ is available in two versions: Visual C++ Standard Edition and Visual C++ Professional Edition. The same integrated development environment is used by both, with a few minor differences. This book describes both versions.

Visual C++ Standard Edition incorporates all the features necessary to develop complete C++ framework-based applications for Windows. In fact, it uses the same integrated development tools, Microsoft Foundation classes, run-time libraries, and Windows libraries as Visual C++ Professional Edition. It also provides all necessary reference documentation from the Windows Software Development Kit (SDK) in online Help files.

Visual C++ Professional Edition is an enhanced version of Visual C++ that adds features such as an optimizing compiler, support for MS-DOS and p-code targeting, documentation for command-line tools, and SDK tools and sample programs. The professional edition provides several additional tools, such as the Microsoft CodeView® debugger (both MS-DOS and Windows versions) and the Microsoft Source Profiler.

There are a only few areas where Visual Workbench differs between models:

- **Project Types:** Visual C++ Professional Edition supports several target types not supported by Visual C++ Standard Edition. In the areas where you assign a project type, namely in the New Project and Project Options dialog boxes, the following extra project types appear in Visual C++ Professional Edition:
  - Windows P-code application
  - MS-DOS application
  - MS-DOS P-code application
  - MS-DOS Overlaid application
  - MS-DOS COM application
- **Compiler Options:** Visual C++ Professional Edition supports the optimizing compiler. Therefore, in the C/C++ Compiler Options dialog box, accessed from the Project Options dialog box, there are several compiler options available for Visual C++ Professional Edition that are either disabled or have limited performance for Visual C++ Standard Edition.
- **CodeView Installation:** The Microsoft CodeView for Windows debugger is provided only with Visual C++ Professional Edition and is optionally installed on the Visual Workbench Tools menu during setup.

# System Requirements

Visual C++ requires the following minimum configuration:

- An IBM Personal Computer, or 100 percent compatible, running MS-DOS version 5.0 or later.
- A VGA monitor.
- An 80386 or higher processor.
- Four megabytes of available memory (6 megabytes recommended).
- A hard disk with enough disk space to install the options you need. The Setup program lets you select installation options and provides you with the disk-space requirements for the options you select. It then checks to make sure you have enough space before copying files.
- A 1.2-MB, 5.25-inch disk drive, or a 1.44-MB, 3.5-inch disk drive.
- Microsoft Windows or Microsoft Windows for Workgroups version 3.1 running in enhanced mode (command-line utilities can be run outside Windows but you need Windows to install Visual C++).

# Installation

The Setup program provided by Microsoft Visual C++ performs all tasks necessary for installing the Visual C++ components. You can install everything at once or install just a subset and upgrade it later with additional libraries, sample programs, help files, or other components.

Visual C++ tools, such as the compiler and linker, are designed to be used with Visual Workbench, the integrated development environment. However, if you are installing the professional edition and want to run the tools from the command line, you need to change your AUTOEXEC.BAT file to set the PATH, LIB, and INCLUDE environment variables. This is described in “MS-DOS Configuration” on page 10.

## Initial Installation

For extensive help on any of the dialog boxes presented by the Setup program during the installation, press the F1 key or choose the Help button in the dialog box.

### ► To run Setup:

1. Place Disk 1 in drive A.
2. If Windows is loaded, choose Run from the Program Manager File menu and type A:\SETUP in the Command Line box.

Setup prompts you with a dialog box that describes the program and lets you continue or exit.

3. Choose Continue.

The Installation Options dialog box appears (see Figure 1.1). Use this dialog box to configure the installation to fit your system.

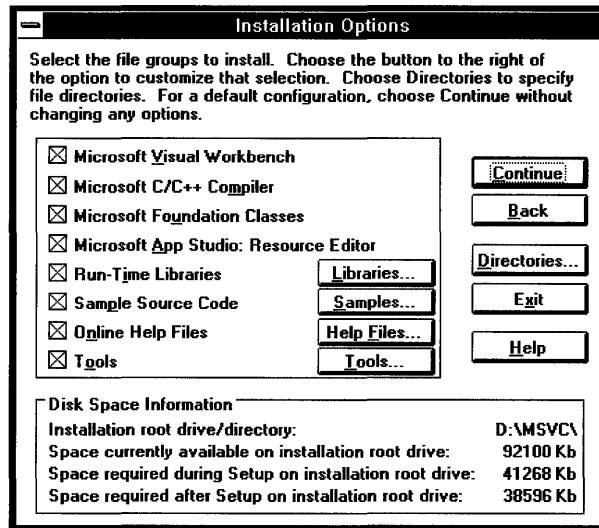


Figure 1.1 The Installation Options Dialog Box

4. In the Disk Space Information group at the bottom of the Installation Options dialog box, compare the disk space listed as available with the disk space listed as required.

If you have enough disk space on drive C to install the entire Visual C++ set of files, you may want to perform a default installation (go on to step 5).

If the default installation requires more disk space than is available on drive C, you have several choices. You can:

- Use the Directory Options dialog box to change the drive to a larger drive.
- Use the Library Options dialog box to remove a memory model or target file type.
- Remove elements listed in the Sample Source Options, Help File Options, and Tool Options dialog boxes.

Disk-space information is provided in each dialog box to help you make the best decision.

5. To perform a default installation, without customization, choose Continue.

Or, if you wish to customize the installation, select whatever installation options you want and then choose Continue.

“Installation Options” on this page provides an overview of each of the installation options. You can also press the F1 key for online help on installation options.

After you choose Continue in the Installation Options dialog box, Setup prompts you for identification information, then builds the file list, checks for sufficient disk space, and copies files from drive A to your hard drive.

## Reinstalling Visual C++

After you have installed Visual C++, you can reinstall it to build additional libraries or copy any other files you didn't install the first time. The following procedure shows you how to reinstall a specific part of Visual C++ and shows, as an example, how you would add the Compact Memory Model library to your existing Visual C++ configuration.

### ► To reinstall a specific part of Visual C++:

1. Run Setup as described in “Initial Installation.”
2. In the Installation Options dialog box, clear all check boxes except the category you want to install.

For example, clear all check boxes except Run-Time Libraries.

3. Choose the command button that corresponds to your category.

For example, choose Libraries. The dialog box for the category you chose appears.

4. Clear all default options and select the new option or options.

For example, in the Memory Models group, clear Small/Tiny, Medium, and Large/Huge and select Compact.

5. Choose OK to close the category dialog box.
6. Choose Continue to proceed with the installation.

## Installation Options

You can customize your installation of Visual C++. For example, you can choose what drive you want the files installed on, which libraries you want built, and which sample files and help files you want installed. Customizing your installation can be useful if you want to minimize disk-space usage. You can install just the components you want now and add components at a later time. This section describes each of the dialog boxes available from the Installation Options dialog box.

### Directories

Use the Directory Options dialog box to change the directories specified for any of the Visual C++ components. The Install Directory sets the drive and root directory



name. Any changes you make to the path in the Install Directory text box are automatically made to all other directory paths when you move the focus out of the box.

The Disk Space Information group at the bottom of the dialog box shows you how much space is available on each hard drive selected in the Drive list box so you can determine which drive to use.

## Libraries

Use the Library Options dialog box to choose which memory models, math support, and target types to build libraries for. You can choose from among the following options:

- Memory models: Small/Tiny, Medium, Compact, and Large/Huge
- Math support: Emulation, 80x87, or Alternate (math support for Visual C++ Professional Edition only)
- Targets: Windows .EXE, Windows .DLL, QuickWin .EXE, and MS-DOS .EXE (MS-DOS .EXE for Visual C++ Professional Edition only)

## Samples

Sample programs can be installed or omitted by category. The following sample file categories are available in the Sample Source Options dialog box:

- User's Guide Samples
- Windows C++ (MFC) Samples
- Windows C (SDK) Samples (SDK sample programs for Visual C++ Professional Edition only)

## Help Files

Help files for an application or library are installed by default when the application or library is selected in the Installation Options dialog box and the Online Help Files check box is also selected. The various Help files and what they contain are as follows:

- Class Library Help (reference help for the Microsoft Foundation Class Library)
- Windows 3.1 SDK Help (reference help for the Software Development Kit and Windows APIs)
- C Lang/Libs Help (reference help for the C and C++ languages and run-time libraries)
- Pen/Multimedia Help (reference help for the Pen Windows and Multimedia APIs—Visual C++ Professional Edition only)

## Tools

The Tool Options dialog box is available with Visual C++ Professional Edition only. It provides check boxes for Windows-hosted tools and MS-DOS–hosted tools you can install. Windows-hosted tools include:

- CodeView for Windows
- Debug Kernel
- Pen Files
- Windows Profiler
- Redistributable Files
- Analysis Tools
- Help Compiler
- Font Editor

MS-DOS tools include:

- CodeView
- MS-DOS Profiler

## Setting Command-Line Options for Visual Workbench

After you have installed Visual Workbench, you can use the Properties dialog box (opened from the Program Manager File menu in Windows) to specify command-line options for Visual Workbench. You may find this useful if you want to load a particular project or set of source files every time you start Visual Workbench.

► **To set command-line options for Visual Workbench:**

1. Move the focus to the Visual C++ icon.
2. From the Program Manager File menu, choose Properties.
3. In the Command Line box, following MSVC.EXE, add a space followed by each filename you want Visual Workbench to load automatically.

Visual Workbench uses the first file with an extension .MAK as the project file and assumes all other files are source files. For example, the following entry in the Command Line box would load the MYPROJ project and the source files MYPROJ.CPP and MYPROJ.H, all located in the C root directory:

```
C:\MSVC\BIN\MSVC.EXE C:\MYPROJ.MAK C:\MYPROJ.CPP C:\MYPROJ.H
```

**Note** The most recently used project is always loaded automatically when you start Visual Workbench so you don't need to use Visual Workbench command-line options to load your current project. Also, by default, the most recently used workspace is loaded when you start Visual Workbench, which automatically loads the files and window configuration from your last session. See Chapter 8, "Using Projects," for more information on using workspaces.

---

## MS-DOS Configuration

Since Visual C++ is designed primarily to be used with Visual Workbench, the Setup program doesn't configure your system to run Visual C++ components from the MS-DOS command line. However, Visual C++ Professional Edition does provide command-line tools and documentation. So if you want to use the compiler, linker, NMAKE, or any of the other Visual C++ components from the command line, you need to either make changes to your AUTOEXEC.BAT file or run the batch file MSVC\BIN\MSVCVARS.BAT to temporarily set your environment variables.

To update your AUTOEXEC.BAT file for MS-DOS command-line tools, use the following procedure. Then exit Windows and either reboot your system or run AUTOEXEC.BAT manually to make the changes effective. This procedure assumes that Visual C++ is installed on drive C in a directory called MSVC. Substitute any differences in the drive or directory for your installation of Visual C++.

► **To change AUTOEXEC.BAT to run tools from the command line:**

1. Add the following to the PATH variable (separated from the previous path by a semicolon):

```
C:\MSVC\BIN
```

2. Add the following line:

```
SET INCLUDE=C:\MSVC\INCLUDE;C:\MSVC\MFC\INCLUDE
```

Or, if there is already an INCLUDE variable for use elsewhere on your system, add the following to your INCLUDE statement (separated by a semicolon):

```
C:\MSVC\INCLUDE;C:\MSVC\MFC\INCLUDE
```

3. Add the following line:

```
SET LIB=C:\MSVC\LIB;C:\MSVC\MFC\LIB
```

Or, if there is already a LIB variable for use elsewhere on your system, add the following to your LIB statement (separated by a semicolon):

```
C:\MSVC\LIB;C:\MSVC\MFC\LIB
```

CL, LINK, NMAKE, CVPACK, and BSCMAKE are all 32-bit MS-DOS–extended programs that must be run on an 80386 or higher processor running in protected mode. You can run them from the command line either in an MS-DOS session from within Windows or outside of Windows at the MS-DOS prompt. These applications are 32-bit hosted only and do not generate 32-bit code.

The file DOSXNT.EXE must be in the same directory as these files or on the path. (It is installed in the MSVC\BIN directory by default.) In addition, to run these programs under Windows, the following command must be in the [386Enh] section of SYSTEM.INI:

```
DEVICE=C:\MSVC\BIN\DOSXNT.386
```

and the file DOSXNT.386 must be in C:\MSVC\BIN. This is done automatically when you install Visual C++. If you install Visual C++ using a different drive or directory, you'll need to modify the DEVICE command to match your installation.

See the Tools TechNote Viewer for more information on using these tools from the command line.



---

## CHAPTER 2

# Building a Sample Application for Windows

The first thing many programmers want to do after installing a new language and development environment is to perform a test build to make sure everything is installed correctly and to see how the build process works. A “test drive” gives you confidence in your setup and helps you get acquainted with the new development environment.

This chapter takes you directly through a build of a Microsoft Foundation Class Library sample program, after a short introduction to projects. It assumes that you have installed the Microsoft Foundation Class Library and sample programs and that the samples reside in a directory called `\MSVCMFC\SAMPLES` on the hard disk drive on which Microsoft Visual C++ is installed.

## Introducing Projects

Projects are the cornerstone of Microsoft Visual Workbench. A project references all the source files and libraries that make up a program, as well as the compiler and linker commands that build the program. It is composed of a makefile (.MAK), which is compatible with the Microsoft Program Maintenance Utility (NMAKE), and a status file (.VCW), which contains Visual Workbench information. A project is identified by its makefile; the makefile has the same base name as the project with a .MAK extension.

All sample programs have project files associated with them. To learn how to create your own projects, read Chapter 8, “Using Projects.”

Project files from Microsoft Programmer’s WorkBench (PWB) can be used from within Visual Workbench by loading them as external projects. This does not give you access to compiler and linker options (other than Release versus Debug mode), but it does let you migrate quickly from PWB to Visual C++ and build, run, and debug your PWB applications from within Visual Workbench. To learn more about using external projects, see “Using External Projects” on page 107.

## A Sample Windows-Based C++ Application

To familiarize yourself with the steps of building any program that already has a Visual Workbench project, you can use any of the sample projects in the Microsoft Foundation Class Library samples directory. The HELLO program is used in this example.

► **To build and run a sample Windows-based C++ program:**

1. From the Project menu, choose Open.

The Open Project dialog box appears.

2. Select the HELLO.MAK project file (see Figure 2.1).

This file is in the \MSVC\MFC\SAMPLES\HELLO directory.

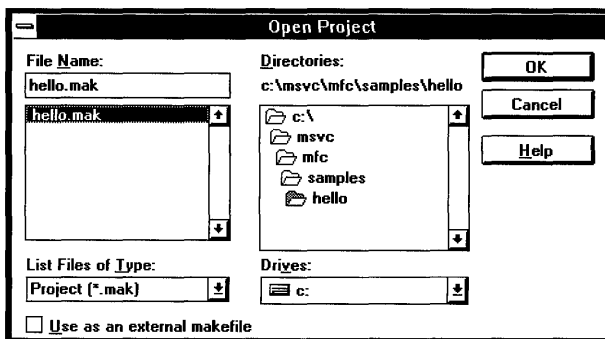


Figure 2.1 The Open Project Dialog Box

3. Choose OK.

4. From the Options menu, choose Project.

The Project Options dialog box appears (see Figure 2.2).

5. From the Build Mode options, select Release or Debug (select Debug for this example for a faster build).

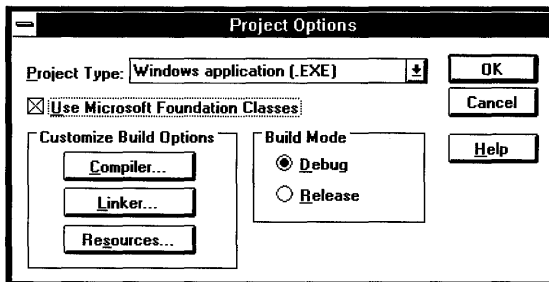


Figure 2.2 The Project Options Dialog Box

6. Choose OK to close the Project Options dialog box.
7. From the Project menu, choose Build HELLO.EXE.

Or click the Build button on the toolbar (see page 60 for more information on the toolbar).

The build occurs entirely in the background, so you are free to continue working in Visual Workbench (although some menu commands are disabled). During the build, output from each of the build utilities, such as the compiler and linker, appears in Visual Workbench's Output window.

When the build is complete, Visual Workbench displays the final results on its status bar. In this case, the status bar should indicate 0 errors and 0 warnings.

8. To run the program without invoking the internal debugger, choose Execute HELLO.EXE from the Project menu.

To run the program in the debugger, choose Go from the Debug menu or click the Run button on the toolbar.

Choose Exit from the program's File menu to quit the program and return to Visual Workbench when you finish experimenting with the program.





# Building a Sample QuickWin Program

Most programmers learn to write C or C++ programs using standard input/output (I/O) functions like `printf()` and `scanf()` (or the C++ iostream objects `cin` and `cout`) to communicate with users of their programs. Hence, many programmers are more comfortable using the standard I/O model than the Windows API programming model. There are lots of reasons why programmers still rely on the MS-DOS I/O model, even when programming in Windows. It is convenient for prototyping code and for quick jobs that don't justify the development of a full-blown Windows interface.

The QuickWin library is the fastest and easiest way to merge MS-DOS programming with Windows. You can write standard I/O programs and have them run in a multiple document interface (MDI) QuickWin window. You can even use all the functions in the MS-DOS GRAPHICS.LIB library. You can use the QuickWin library to port many of your existing MS-DOS programs to Windows without incurring the development overhead associated with Windows.

This chapter shows you the quickest way to use Microsoft Visual Workbench to write, build, and run a QuickWin program.

## What Is QuickWin?

A QuickWin application is a standard I/O program with a Windows shell. It runs only with the Windows operating system. You build your program as a QuickWin project type; then when you run the program from Windows, a QuickWin MDI window appears with a window dedicated to all your program output functions. This window is like a video monitor; you can write to the window and perform input and output operations with standard MS-DOS input/output routines.

QuickWin is an easy way for users of Visual C++ Standard Edition to write MS-DOS-style programs without having to upgrade to the MS-DOS targeting provided with Visual C++ Professional Edition. If you are learning C++ or C

programming, and are using a tutorial such as the *C++ Tutorial*, which uses the standard I/O programming model, you'll find QuickWin useful for experimenting with sample programs.

The following section walks you through a simple QuickWin C++ example. The QuickWin library, however, is capable of handling most well-behaved standard I/O programs, including programs that use the Microsoft GRAPHICS.LIB library. Of course, QuickWin cannot interpret programs that use hardware-specific functions such as BIOS interrupts or functions that directly manipulate video memory.

For a complete description of the QuickWin library, see Chapter 7 of *Programming Techniques*.

## A Sample QuickWin Program

To see how easy it is to use QuickWin, try typing a small sample program in a source window and then building it as a QuickWin program. Or, if you would prefer to try out the sample QuickWin program provided on disk, open the file named QWGDEMO.CPP located in the \MSVC\SAMPLES\QWGDEMO directory and follow the instructions detailed in the procedure titled "To build and run a QuickWin program."

---

**Note** You need the QuickWin library to build QuickWin programs. If you haven't installed this library, you can rerun the Setup program to install it (see page 7 to learn how to install a library using Setup).

---

► **To write a QuickWin program:**

1. If you have a project open, close it by choosing Close from the Project menu.
2. From the File menu, choose New.

An empty source window opens.

3. Type the following code in the source window:

```
#include <iostream.h>
void main()
{
    cout << "Hello C++ world\n";
}
```

If you need help using the editor, see Chapter 7, “Using the Editor.” Figure 3.1 shows the source window with the sample code.

A screenshot of a text editor window titled "<1> UNTITLED 1\*" showing C++ source code. The code is as follows:

```
#include <iostream.h>
void main()
{
    cout << "Hello C++ world\\n";
}
```

**Figure 3.1** QuickWin C++ Program Code

4. From the File menu, choose Save As.  
The Save As dialog box appears.
5. Type the name for the file, QWINTEST.CPP, in the File Name box and choose OK.

---

**Note** Use the extension .CPP or .CXX, or the compiler will not recognize the program as a C++ program.

---

Now that you’ve entered the program, you can build an executable program. For the next procedure, use either the QWINTEST.CPP program from the previous procedure or the QWGDEMO.CPP program (which you must open first). Note that this procedure builds the QuickWin program without creating or using a project. If you plan to develop extensive QuickWin applications, it is suggested that you create and use projects (see Chapter 8, “Using Projects,” for more information).

► **To build and run a QuickWin program:**

1. Click in the source window containing the QuickWin program to make it the active window.
2. From the Options menu, choose Project.  
The Project Options dialog box appears (see Figure 3.2).
3. In the Project Type list box, select QuickWin application (.EXE).
4. From the Build Mode options, select Debug or Release (for this example, select Debug).
5. Choose OK to close the Project Options dialog box.

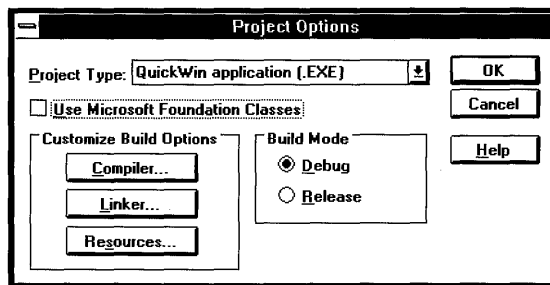


Figure 3.2 The Project Options Dialog Box

6. From the Project menu, choose Build *Targetname* (where *Targetname* is either QWINTEST.EXE or QWGDEMO.EXE in this example depending on the program you are building).

Or click the Build button on the toolbar.

The build process automatically generates the module-definition file required for a QuickWin program.

When the build is finished, there should be 0 errors and 0 warnings reported on the status bar. If there are any errors (probably created by typing errors), correct them and repeat the build. If there are link or include-file errors, make sure the paths for the library files and include files are correct in the Directories dialog box (opened from the Options menu).

7. From the Project menu, choose Execute *Targetname* (where *Targetname* is either QWINTEST.EXE or QWGDEMO.EXE).

The QuickWin window should appear with your program's output in one of its MDI windows (see Figure 3.3).

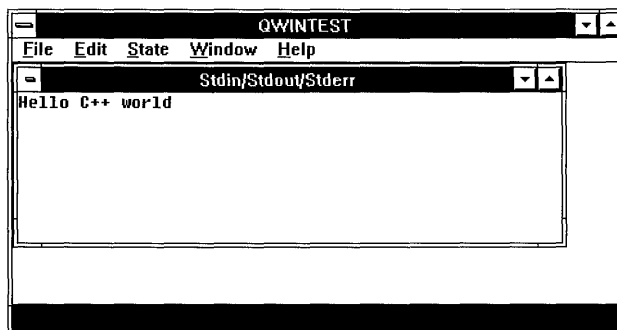


Figure 3.3 QWINTEST Output

After you have experimented with the program, you can close the QuickWin application by choosing Exit from its File menu or by pressing CTRL+C. This will return you to Visual Workbench.

# Developing a Microsoft Visual C++ Application

This chapter discusses the process of using Visual Workbench and App Studio to create a Visual C++ application. It presents an overview of the development process and puts all the tools in perspective.

It is highly recommended that you complete the Scribble tutorial in the *Class Library User's Guide* to learn how to develop a Visual C++ application. You can read this chapter as an overview to the tutorial or as a quick refresher.

## What Is a Visual C++ Application?

A Visual C++ application is an application for Windows that you design and develop using the Microsoft Foundation Class Library, the Microsoft Visual C++ build tools, and the Visual Workbench and App Studio Windows-hosted development tools.

By using a totally integrated environment, you approach programming your application the way that users approach using your program—from the visual interface elements. Visual C++ calls these elements “user-interface objects.” You first design the user-interface objects and then use Visual C++ tools to create and manage the code to support them. Visual C++ tools help automate the tedious and error-prone process of deriving classes, creating member functions, and mapping them to messages. This automation lets you concentrate on designing the resources for your application and writing the functional code to handle messages.

Of course, you can also use Visual C++ tools to develop standard Windows SDK applications in C or C++, since Visual C++ includes a text editor, project manager, build utility, browser, debugger, and resource editor. But, if you are familiar with standard SDK programming for Windows, you'll find that the Visual C++ tools make the transition to object-oriented program development, and the Microsoft Foundation Class Library, easier than you might have imagined.

# Introducing the Tools

The Visual C++ tools consist of two applications, Visual Workbench and App Studio, and two wizards, AppWizard and ClassWizard. If you have Visual C++ Professional Edition, you have additional auxiliary tools. However, since these are not fundamental to developing a Visual C++ application using the Microsoft Foundation classes, they are not discussed in this chapter.

## The Applications

Visual Workbench and App Studio are the two Windows-hosted applications that work together to help you develop Visual C++ applications.

### Visual Workbench

Visual Workbench is the main editing and debugging tool and acts as the anchor for the programming environment by creating and maintaining project information (see Figure 4.1). It incorporates a text editor, project manager, browser, and debugger in a single integrated development environment.

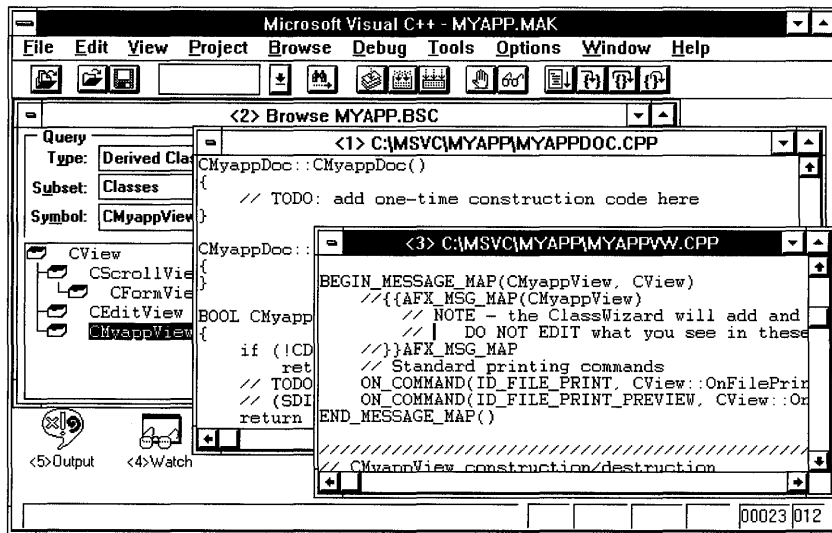


Figure 4.1 Visual Workbench

## App Studio

App Studio lets you create and edit all your application's resources, including dialog boxes, menus, icons, bitmaps, cursors, and more, in a single environment (see Figure 4.2). It can be used as a stand-alone resource editor to read and generate resource files for standard Windows SDK program development. Or it can be integrated with ClassWizard, AppWizard, and Visual Workbench to develop Visual C++ applications using the Microsoft Foundation classes.

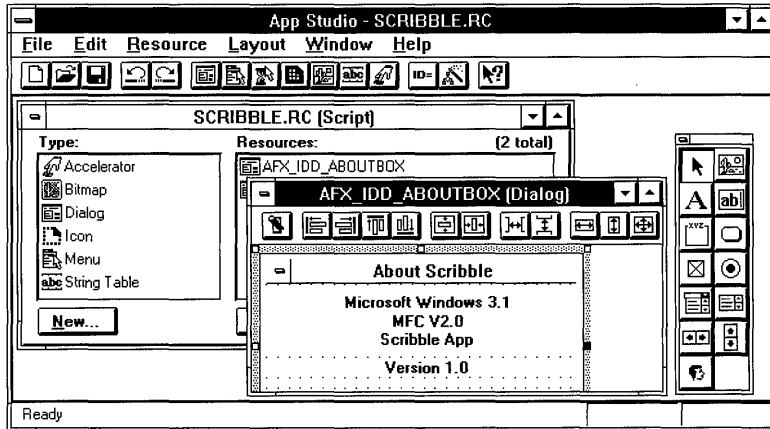


Figure 4.2 App Studio

## The Wizards

Two Visual C++ wizards, AppWizard and ClassWizard, help you design, create, and implement Visual C++ applications using the Microsoft Foundation classes. Wizards can be thought of as helpers that relieve you of much of the tedious work in application development.

### AppWizard

AppWizard is a tool that generates a complete suite of source files and resource files for a Microsoft Foundation Class Library application (see Figure 4.3). It is important that AppWizard be used first during the development of a Visual C++ application.



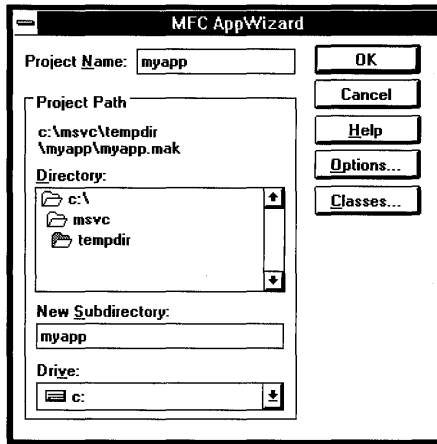


Figure 4.3 AppWizard

By selecting options in AppWizard, you can create skeleton C++ source files with differing levels of functionality. If you enable all AppWizard options, generate a project, and then build it without adding a single line of code, you'll get a Windows-based application with a surprising amount of functionality. A fully optioned AppWizard-generated application has:

- A multiple document interface (MDI).
- Menus and dialog boxes for opening and saving files, printing, and print preview.
- Support for object linking and embedding (OLE).
- Support for Microsoft Visual Basic™ custom controls (custom VBX controls).
- Support for Help.
- A functional toolbar and status bar.

AppWizard is accessed from the Visual Workbench Project menu.

## ClassWizard

ClassWizard (see Figure 4.4) is a tool that allows you to:

- Create new classes.
- Map messages to class-member functions.
- Map controls to class-member variables.

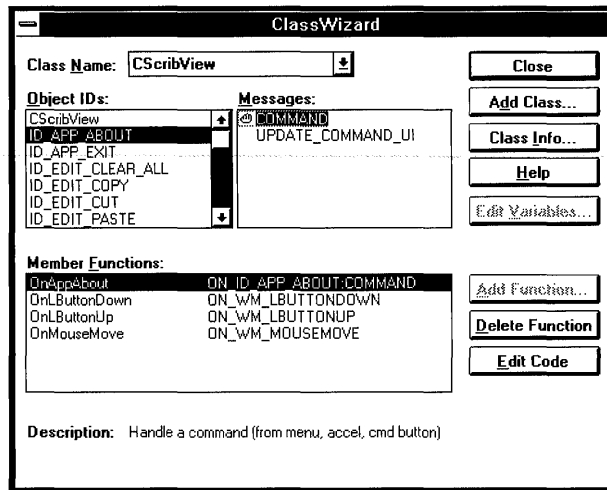


Figure 4.4 ClassWizard

Creating a new class using ClassWizard is as simple as selecting a base class from a list of available base classes and typing in the name of the new class. ClassWizard automatically creates the necessary source files and declaration and implementation code to derive the new class.

You use ClassWizard most often to “bind” user-interface objects to code. After creating the user-interface objects in App Studio, you use ClassWizard to create member functions and message maps to handle messages from these objects.

For example, if you add a menu item “Test” to the Edit menu in App Studio and then open ClassWizard, you can select the class to which you want the message-handler function for that object added, select the resource identifier for the Test menu item (**ID\_EDIT\_TEST** in the Object ID list), specify that it is a **COMMAND** in the Messages list, and choose the Add Function button. After presenting you with a message box to allow you to alter the function name, ClassWizard inserts the message-map entry, function prototype, and skeleton function code.

You can also easily create message-handler functions for standard Windows messages, such as **WM\_MOUSEMOVE**. ClassWizard displays a list of standard Windows messages for each of the classes that can handle messages. To hook up a message with a message-handler member function, you just select a class, select the message you want to handle, and choose the Add Function button.

Finally, ClassWizard can be used to automate development of most of the code needed for getting data into and out of dialog boxes and for checking the validity of user input to a dialog-box edit control. ClassWizard's dialog-data-exchange feature creates source code to map data from dialog-box controls to class-member variables. The dialog-data-validation feature creates source code to respond to incorrect user entries in a dialog box.

ClassWizard is accessed from either Visual Workbench or App Studio.

## The Process

The process for developing a Visual C++ application using the tools just described can be broken into two stages: application creation and application development. The first stage is straightforward, while the second contains several steps that are iterative and involve many components.

## The Application Creation Stage

The first step in creating a Visual C++ application is to generate a set of application starter files using AppWizard (see Figure 4.5). These starter files are in a format that can later be recognized by ClassWizard.

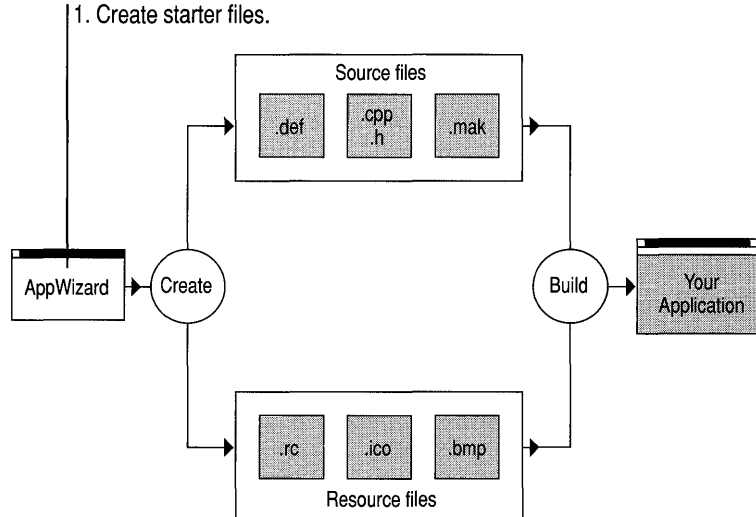


Figure 4.5 Creating the Visual C++ Framework Code

To run AppWizard, you start Visual Workbench and then choose AppWizard from the Project menu. You then specify a name for the project and select a location for the source files. At this point, you can also select from the available options in AppWizard's Options dialog box. When first exploring AppWizard, you might want to start with the default options.

When you choose OK, AppWizard creates all the files required for a standard Visual C++ application, including source files, resource files, and a Visual Workbench project file as shown in Figure 4.5. Visual Workbench then loads the project. At this point, you can immediately compile and link the files by choosing Build *Targetname* (where *Targetname* represents the name of the project you created) from Visual Workbench's Project menu.

For more information on using AppWizard, see Chapter 13, "Using Visual Workbench with Other Visual C++ Tools," in this book and Chapter 2, "Creating a New Application with AppWizard," in the *Class Library User's Guide*.

## The Application Development Stage

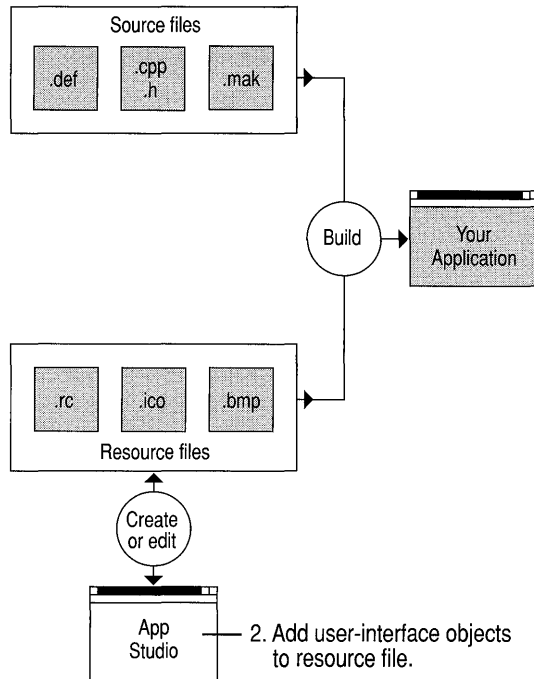
This section describes the process of developing a Visual C++ application using Visual Workbench, App Studio, AppWizard, and ClassWizard. Since so much of the development process involves Visual Workbench, the Visual Workbench tools are also introduced.

The development stage of any Windows-based application includes the familiar passes involved in editing source and resource files, compiling and linking, testing, and debugging. Because these activities are iterative and interwoven during a normal development cycle, they can't easily be serialized into steps.

There is an order involved, however, in using App Studio, ClassWizard, and Visual Workbench, mostly because you always create user-interface objects first in App Studio, then use ClassWizard to create the code shell, and then use Visual Workbench to write the functional code. This order, normally repeated many times during typical development, is examined here.

## Creating and Editing User-Interface Objects

Figure 4.6 shows the starter files that are created by AppWizard. Although AppWizard starts you out with some basic user-interface objects (such as menus, a toolbar, and so on) when it generates the resource files, you will undoubtedly need to add user-interface objects of your own. To do this, you use App Studio, shown editing the resource files in Figure 4.6.

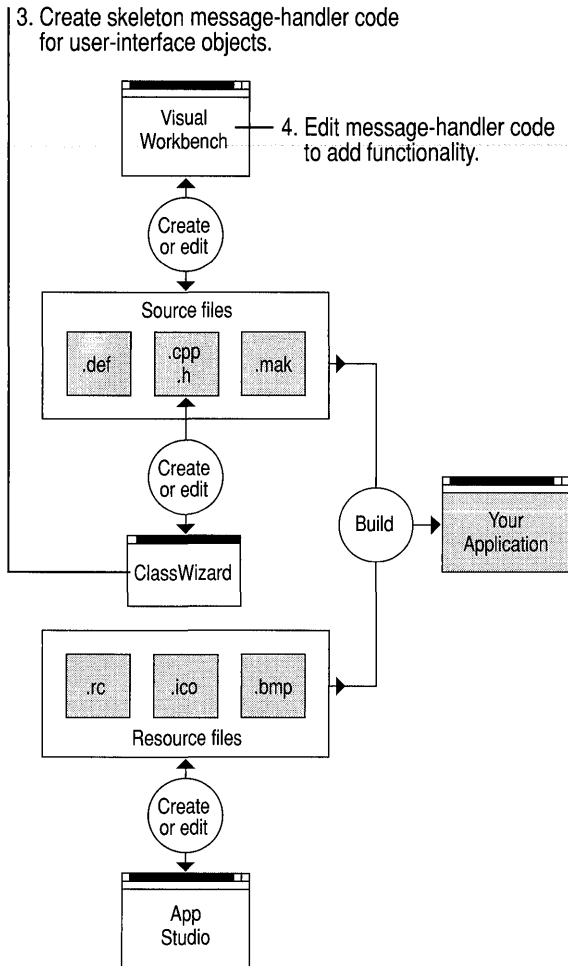


**Figure 4.6** Creating and Editing User-Interface Objects

You can easily open App Studio from within Visual Workbench by choosing App Studio from the Tools menu. This opens App Studio and passes the name of the resource file (.RC) to it. To learn how to use App Studio to create and edit user-interface objects, see the *App Studio User's Guide*.

## Connecting User-Interface Objects to Code

After creating the user-interface objects in App Studio, the next step is to create the Microsoft Foundation Class Library code that supports them. Figure 4.7 shows the additional two tools, ClassWizard and Visual Workbench, that come into play at this point.



**Figure 4.7** Connecting User-Interface Objects to Code

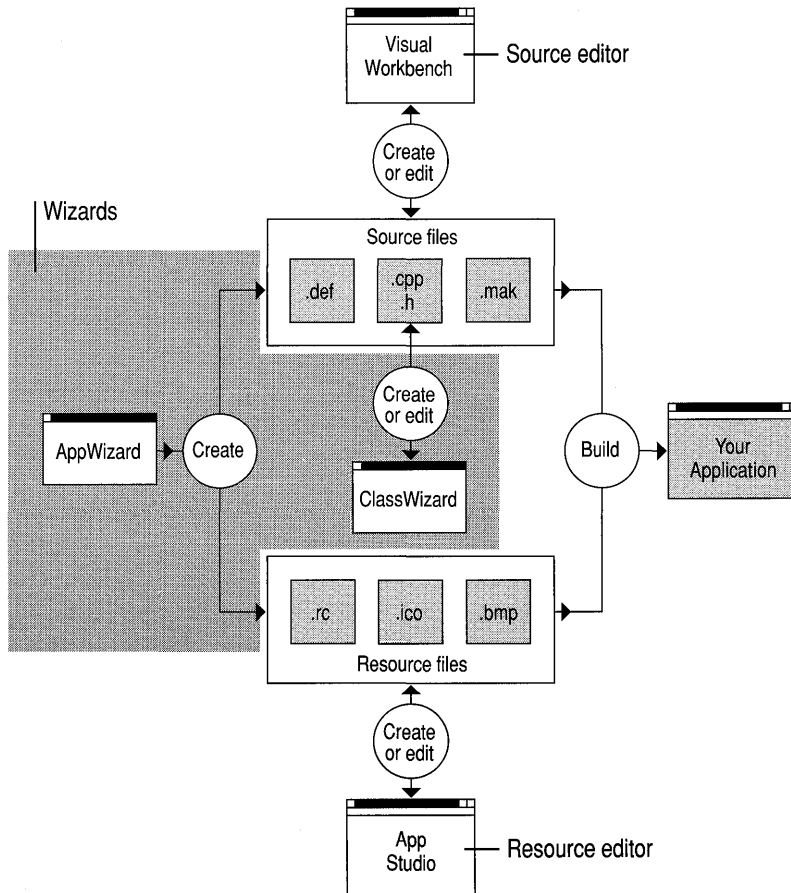
You use ClassWizard to generate the message-handler functions and message maps for each user-interface object created in App Studio.

With the skeleton prototype and function code inserted by ClassWizard, you use Visual Workbench to actually add the code. ClassWizard even lets you jump directly into Visual Workbench at the proper location to add code.

Depending on your style, you may prefer to connect each user-interface object to code as it is created in App Studio, or wait until you are finished in App Studio before using ClassWizard and Visual Workbench. The integrated nature of these three tools adapts easily to your own style of working.

## The Entire Process

Figure 4.8 shows how AppWizard, App Studio, ClassWizard, and Visual Workbench all work together. If you have developed applications for Windows prior to Visual C++, you can see the “standard” development process by mentally removing ClassWizard and AppWizard from the picture—where Visual Workbench represents the source editor and App Studio represents the resource editor (or editors) you might have used to create source and resource files. This development model has been widely used and is still available and fully supported in Visual C++.



**Figure 4.8 The Visual C++ Development Process**

AppWizard simply helps you by creating customized starter files in a location of your choice. ClassWizard adds another dimension to this standard model by keeping track of source code and user-interface objects and therefore lets you derive

classes, connect resource identifiers to code, and edit source code from a single vantage point.

What isn't depicted in this illustration is that Visual Workbench is the central tool that coordinates all the other tools by maintaining the project information. Visual Workbench is not just where you edit files; it is also where you manage the source code, and build and debug the application. The next two sections describe the tools you use in Visual Workbench during application development.

## Managing Your Source Code

ClassWizard and the Visual Workbench source browser are source-code management tools that allow you to access your source code from a structured viewpoint.

As discussed earlier, ClassWizard keeps track of all resource objects and member functions. It lets you immediately jump to the message-handler source code from ClassWizard so that you can edit it.

The Visual Workbench browser is another Visual C++ tool for managing source code. You open the browser by opening the Browse window in Visual Workbench (see Figure 4.9). You can use the browser to:

- Graphically display hierarchical class trees of derived or base classes.
- Graphically display all the functions that call, or are called by, a particular function.
- Display a list of source-code locations where references to a symbol are made and where a symbol is defined.
- Display member-function and member-variable lists for C++ classes.
- Jump directly to definitions and references either from list entries in the Browse window or from a selected symbol in a source file.

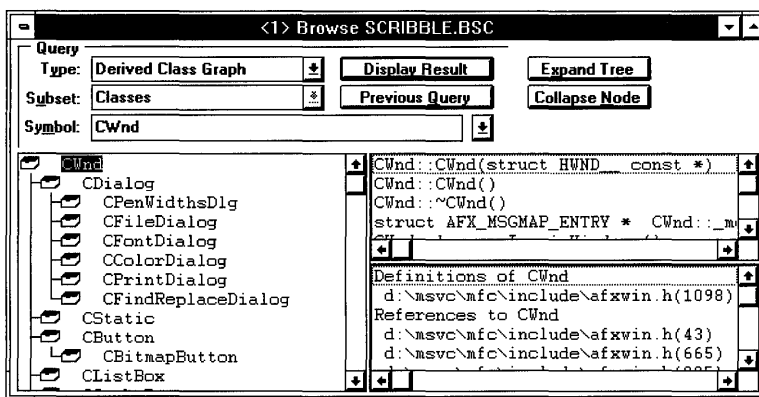


Figure 4.9 The Visual Workbench Browse Window



Like ClassWizard, you can use the browser to look at your code from a different viewpoint than normal text mode. The browser shows you relationships between base and derived classes and between calling and called functions. You can also jump directly from the Browse window to source code simply by double-clicking a reference or definition in the Browse window. Or, without even using the Browse window, you can select a symbol in a source file, jump to its definition or first reference, visit all references to the symbol, and return to the original location, all using menu commands or shortcut keys.

See Chapter 10, “Using the Browser,” to learn more about the Visual Workbench browser.

## Building and Running Your Application

Visual Workbench helps you build, run, and debug your application with as little interruption as possible since these tasks are repeated so frequently. You can build a project in Visual Workbench by:

- Choosing the Build or Rebuild All toolbar button.
- Choosing the Build *Targetname* or Rebuild All *Targetname* command from the Project menu.

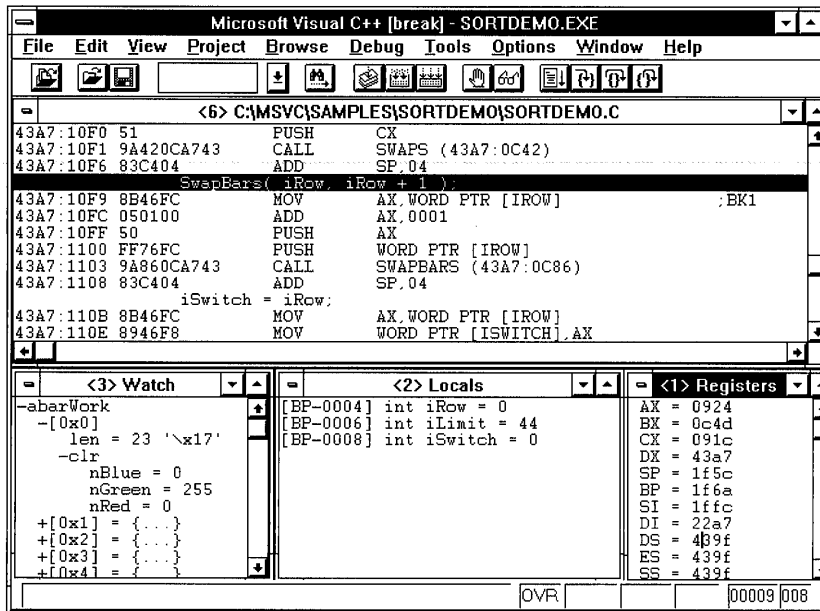
When the build is complete, you can run the program in the debugger (assuming it includes debug information) by:

- Choosing the Run toolbar button.
- Choosing the Go command from the Debug menu.

You can run the program outside the debugger by choosing Execute *Targetname* from the Project menu. The steps for building and running an application are provided in Chapter 2, “Building a Sample Application for Windows,” and in Chapter 8, “Using Projects.” You can find toolbar descriptions in Chapter 6, “The Visual Workbench Environment.”

## Debugging Your Application

You debug your application from within Visual Workbench. The Visual Workbench debugger is a Windows-hosted debugger that is integrated with Visual Workbench and is compatible with Microsoft CodeView (see Figure 4.10).



**Figure 4.10** The Visual Workbench Debug Windows

The Visual Workbench debugger has many powerful features, including:

- Breakpoints for breaking a program at a location, on an expression evaluation, or on a Windows message or class of messages.
- A QuickWatch dialog box for examining and changing variable values.
- A Watch window for examining specific variables and expressions.
- A Locals window for examining local variables.
- A Registers window for examining and changing hardware register values.
- Tracing commands to step over, step into, or step out of functions.
- Mixed source and assembly listings and assembly-line tracing.

To use the debugger, you build your application with the Debug configuration. You then set breakpoints and run the application in a debug session from Visual Workbench.

When the debugger reaches a breakpoint, you have several options. You can examine variables or expressions using the Watch window, the Locals window, or the QuickWatch dialog box. Or you can single-step through the code, choosing to step over functions or trace into functions that are encountered.

For a complete description of the Visual Workbench debugger see Chapter 11, “Debugging Programs.”

## For More Information

The topics introduced in this chapter are covered in detail in a number of places in the Visual C++ printed documentation. Please refer to the following documentation for more information:

- The *Class Library User's Guide* contains a comprehensive tutorial on developing a Visual C++ application.
- Chapters 1 through 6 of the *Class Library Reference* cover conceptual information on using the Microsoft Foundation classes.
- Chapters 6 through 12 of the *Visual Workbench User's Guide* describe how to use general Visual Workbench features.
- Chapter 13 of the *Visual Workbench User's Guide* describes AppWizard in detail and discusses both using ClassWizard and running App Studio from within Visual Workbench.
- Chapters 3 through 8 of the *App Studio User's Guide* describe how to use general App Studio features.
- Chapter 9 of the *App Studio User's Guide* discusses using ClassWizard from within App Studio and presents a comprehensive description of ClassWizard.

# Fast Track to Visual Workbench

Chapters 6 through 13 provide comprehensive information and procedures for using all parts of Visual Workbench. Often, however, all you need is a pointer in the right direction, and perhaps a comment about the procedure, to get started. The tables in this chapter will help you quickly get to the right menu command, dialog box, window, or compiler or linker control to accomplish your task.

This chapter contains the following sections:

- Menu Summaries
- Key Summaries
- Alphabetic Guide to Build Options

You can use this chapter as an introduction to the Visual Workbench menus, shortcut keys, and compiler and linker options. Or you can refer to the tables later for a quick reference.

# Menu Summaries

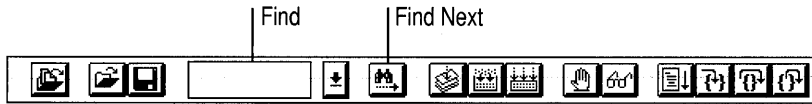
This section summarizes many of the Visual Workbench procedures and provides a comprehensive menu overview. Since most menu items are referenced to a complete description, you can also use this section as an index into the *Visual Workbench User's Guide*.

## Managing Files and Printing



To	<b>File</b>		Comments	See
Create a new source file	<b>New</b>	<b>Ctrl+N</b>	Creates an empty source window that will be named when it is saved.	p. 76
Open a source file	<b>Open...</b>	<b>Ctrl+O</b>	The Open button on the toolbar is equivalent. To open a project file, click the Project Files button on the toolbar and choose the file from the drop-down list that appears.	p. 79
Close a source file	<b>Close</b>		Or double-click the file's Control-menu box.	p. 82
Save a source file	<b>Save</b>	<b>Ctrl+S</b>	The Save command and the Save button on the toolbar are inactive until you alter text in the source window.	p. 77
Rename a source file	<b>Save As...</b>		Opens the Save As dialog box, which prompts you for a filename.	p. 76
Save all open source files	<b>Save All</b>		Saves all source files currently opened.	p. 77
Print the active window	<b>Print...</b>		Opens the Print dialog box. If text is selected, this prints just the selected text.	p. 90
Set print margins, headers, and footers	<b>Page Setup...</b>		Opens the Page Setup dialog box. This is similar to the page setup capability of other applications for Windows.	p. 91
Exit Visual Workbench	<b>Exit</b>		Exits the development environment and prompts you to save any modified files.	p. 60

## Editing Files

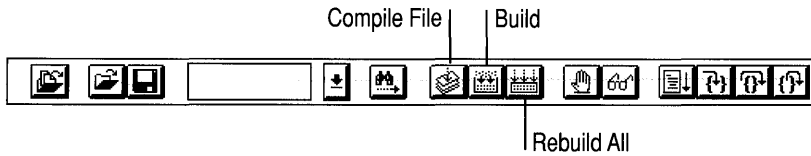


To	<b>Edit</b>		Comments	See
Reverse the last edit action	<b><u>U</u>ndo</b>	<b>Ctrl+Z</b>	To set the undo buffer size, see the Editor dialog box (Options menu).	Help
Reverse the last undo	<b><u>R</u>edo</b>	<b>Ctrl+A</b>	Must be used before any other editing is done.	Help
Delete to the Clipboard	<b><u>C</u>ut</b>	<b>Ctrl+X</b>	Overwrites the current Clipboard contents.	Help
Copy to the Clipboard	<b><u>C</u>opy</b>	<b>Ctrl+C</b>	Overwrites the current Clipboard contents.	Help
Paste from the Clipboard	<b><u>P</u>aste</b>	<b>Ctrl+V</b>	Pastes text only.	Help
Delete text from a file	<b><u>D</u>elete</b>	<b>Del</b>	Does not save to the Clipboard.	Help
Search for text	<b><u>F</u>ind...</b>	<b>Alt+F3</b>	Opens the Find dialog box. Or you can use the Find box on the toolbar to select from the last 16 items searched. To repeat the last search, click the Find Next button on the toolbar.	p. 86
Search and replace text	<b><u>R</u>eplace...</b>		Opens the Replace dialog box.	p. 89
Move to the corresponding brace	<b><u>F</u>ind <u>M</u>atching <u>B</u>race</b>	<b>Ctrl+]</b>	Moves the insertion point from a brace or parenthesis to its matching brace or parenthesis (forward or backward).	p. 83
Write-protect a file	<b><u>R</u>ead <u>O</u>nly</b>		Useful for viewing files that you don't want to accidentally alter.	p. 86

## Window Display and Quick Access

To	<b>View</b>		Comments	See
Go to a specific line	<u>L</u> ine...		Opens the Line dialog box. Type a line number and press ENTER.	p. 82
Display assembly code	<u>M</u> ixed Source/Asm	<b>Ctrl+F7</b>	Displays assembly code mixed with source code.	p. 198
Jump to the next build error	<u>N</u> ext Error	<b>F4</b>	Places insertion point in source file at place of next build error (opens file if necessary).	p. 181
Jump to the last build error	<u>P</u> revious Error	<b>Shift+F4</b>	Places insertion point in source file at previous error (opens file if necessary).	p. 181
Turn bookmark on or off	<u>T</u> oggle Bookmark	<b>Ctrl+F2</b>	Line is highlighted when bookmark is set.	p. 82
Jump to the next bookmark	<u>N</u> ext Bookmark	<b>F2</b>	Places insertion point at next bookmark.	p. 83
Jump to the previous bookmark	<u>P</u> revious Bookmark	<b>Shift+F2</b>	Places insertion point at previous bookmark.	p. 83
Remove all bookmarks	<u>C</u> lear All Bookmarks		Clears all bookmarks set by Toggle Bookmark or by the Set Bookmarks on All button in the Find dialog box.	p. 83
Toggle toolbar display	<u>T</u> oolbar		Checked when toolbar is displayed.	p. 60
Toggle status bar display	<u>S</u> tatus Bar		Checked when status bar is displayed.	p. 63
Override syntax coloring	<u>S</u> yntax Coloring		Select C, C++, or None from pop-up menu to set syntax coloring for a single source file. To set global syntax coloring, use the Color dialog box from the Options menu.	p. 205

## Working with Projects



To	<b>Project</b>		Comments	See
Create a new application using AppWizard	<b>App Wizard...</b>		Opens the MFC AppWizard dialog box, which you use to create a suite of project files to be used with ClassWizard.	p. 210
Create a new project	<b>New...</b>		Opens the New Project dialog box, where you name the project, select a project type, and add files to the project. The new project is automatically saved to disk.	p. 98
Open an existing project	<b>Open...</b>		Closes any current project. Loads browser database, if available, for the project.	p. 100
Add files to or delete files from a project	<b>Edit... PRJNAME.MAK</b>		To change the project type, use the Project command on the Options menu.	p. 99
Close the current project	<b>Close</b>		Other ways to close a project are opening another project, creating a project, or exiting Visual Workbench.	p. 100
Compile the active source file	<b>Compile File FILENAME</b>	<b>Ctrl+F8</b>	Compiles active source file specified by <i>FILENAME</i> . Also available on the toolbar.	Help
Build the project using dependency rules	<b>Build TARGETNAME</b>	<b>Shift+F8</b>	Builds only the files not up-to-date and creates a target file. Also available on the toolbar.	p. 103
Build the project from start	<b>Rebuild All TARGETNAME</b>	<b>Alt+F8</b>	Builds all the files regardless of dependencies and creates a target file. Also available on the toolbar.	p. 104
Abort building a project	<b>Stop Build</b>		Cancels the build as soon as the build tool being used is finished.	p. 104

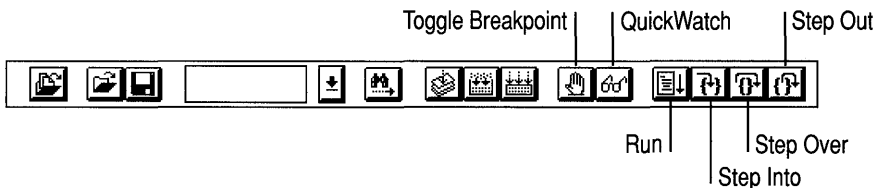


To	Project		Comments	See
Run a program outside the debugger	<b>Execute <i>TARGETNAME</i></b>	<b>Ctrl+F5</b>	<i>TARGETNAME</i> is the name of the executable file produced by building the current project. See the Debug menu's Go command to run a program in the debugger.	p. 104
Update include dependency list for active file	<b>Scan <u>D</u>ependencies <i>FILENAME</i></b>		Recursively scans all files included by the active source file and updates the list.	p. 102
Update include dependency list for entire project	<b>Scan <u>A</u>ll Dependencies</b>		This is also done automatically when a project list is created or edited.	p. 102
Load a workspace	<b><u>L</u>oad Workspace <i>PRJNAME.WSP</i></b>		Loads one of three previously saved workspaces associated with the current project.	p. 106
Save a workspace	<b><u>S</u>ave Workspace <i>PRJNAME.WSP</i></b>		Saves one of three workspaces associated with the current project.	p. 105

## Accessing Symbols and Classes

To	<b>Browse</b>		Comments	See
Jump to first place where symbol is defined	<b>Go to <u>D</u>efinition</b>	<b>F11</b>	Select symbol in source window or Browse window.	p. 168
Jump to first place where symbol is referenced	<b>Go to <u>R</u>eference</b>	<b>Shift+F11</b>	Select symbol in source window or Browse window.	p. 168
Jump to the next reference or definition	<b><u>N</u>ext</b>	<b>Ctrl+NumPad+</b>	Depends on whether Go to Definition or Go to Reference was used last.	p. 168
Jump to the previous reference or definition	<b><u>P</u>revious</b>	<b>Ctrl+NumPad-</b>	Depends on whether Go to Definition or Go to Reference was used last.	p. 168
Jump to location of selected symbol	<b>Pop <u>C</u>ontext</b>	<b>Ctrl+NumPad*</b>	Returns to the symbol that was used before the last Go to Definition or Go to Reference.	p. 169
Open the Browse window	<b><u>O</u>pen [<i>PRJNAME.BSC</i>]</b>		Opens the Browse window with the database listed or, if no database name is listed, opens the Open File dialog box.	p. 163
Create a new class or map a Windows message to a member function	<b><u>C</u>lassWizard...</b>	<b>Ctrl+W</b>	Opens a dialog box that lets you create new classes and automate the creation of message maps and message-handler member functions.	p. 222

## Debugging Programs



To	<b>Debug</b>		Comments	See
Start or continue a program using the debugger	<b>Go</b>	<b>F5</b>	Runs program associated with the current project.	p. 189
Reload program and start	<b>Restart</b>	<b>Shift+F5</b>	Use when execution has paused at a breakpoint, between steps, or when program has completed. Discards variable values.	p. 189
Quit a debugging session	<b>Stop Debugging</b>	<b>Alt+F5</b>	Stops the debugger at any time, whether the program is running or paused. If this is dimmed (unavailable), the program being debugged has focus or has finished.	p. 189
Single-step through every line of the program	<b>Step Into</b>	<b>F8</b>	Enters functions when encountered.	p. 190
Single-step through program but skip functions	<b>Step Over</b>	<b>F10</b>	Runs functions when encountered and stops immediately after.	p. 190
Run to first instruction after call to current function	<b>Step Out</b>	<b>Shift+F7</b>	Continues program out of function then stops at line after function call.	p. 190
Run to the location of the insertion point	<b>Step to Cursor</b>	<b>F7</b>	Treats the insertion point location as a breakpoint.	p. 190
Display the call stack	<b>Show Call Stack...</b>		The Call Stack dialog box lists all function calls that lead to current statement.	p. 194
Set or clear a breakpoint	<b>Breakpoints...</b>		The Toggle Breakpoint button on the toolbar also sets and clears breakpoints.	p. 185

To	<b>Debug</b>		Comments	See
Open QuickWatch to view the value of a variable,	<u>QuickWatch...</u>	<b>Shift+F9</b>	The insertion point must be on a variable with the program paused at a breakpoint or between steps.	p. 192
Or add a variable to the Watch window,			To add a variable, open QuickWatch on the variable and choose the Add to Watch Window button.	p. 192
Or change a variable value			To change a variable value, open QuickWatch on the variable and choose the Modify button to open the Modify Variable dialog box.	p. 193

## Running Tools from Visual Workbench

To	<b>Tools</b>		Comments	See
Create menus, dialog boxes, icons, and other resources	<u>App Studio</u>		App Studio lets you visually design and construct the user-interface objects for your program.	p. 23
Run CodeView for Windows	<u>CodeView</u>		CodeView is only provided with Visual C++ Professional Edition.	p. 179
Add a tool to or remove a tool from the Tools menu			See the Tools command on the Options menu.	p. 199

## Setting Preferences and Custom Options

To	Options	Comments	See
Set custom compiler and linker options,	<b>Project...</b>	Choose the Compiler, Linker, or Resources button. Use F1 in the dialog boxes for help.	p. 111
Or set the build mode (debug vs. release),		You can set the build mode for Visual Workbench projects or external projects.	p. 111
Or set the project type		Project type can also be set when you create a new project.	p. 94
Specify an .EXE file to debug a DLL,	<b>Debug...</b>	You must specify a host program to debug any dynamic-link library.	p. 179
Or set hard or soft mode debugging,		In hard mode, the debugger traps all input from the mouse and keyboard when in break mode.	p. 196
Or toggle hexadecimal display		Format for display of all variable values wherever shown (for example, Watch window).	p. 198
Set directories for include files, libraries, executable files, help files, and MFC source files	<b>Directories...</b>	Visual Workbench prefixes directory information to any existing PATH, INCLUDE, or LIB environment variable. The MFC source file directory is required when debugging or browsing MFC programs.	p. 204
Set miscellaneous options	<b>Editor...</b>	Options include tab settings, scroll bar enabling, save before build, prompt before save, and undo buffer size.	p. 85
Set workspace options	<b>Workspace...</b>	Lets you set the menu names and shortcut keys and other options for workspaces.	p. 104
Add a tool to, or remove a tool from, the Tools menu	<b>Tools...</b>	You can use this feature to integrate a favorite or familiar editor with Visual Workbench.	p. 199
Change syntax coloring,	<b>Color...</b>	The Color dialog box lets you customize colors for all syntax elements.	p. 205
Or turn syntax coloring off for all files		Clear the Syntax Coloring check box. See the Syntax Coloring command on the View menu to select syntax coloring for a single file.	p. 206
Change the font in a window	<b>Font...</b>	Select from a variety of fonts and sizes and apply bold and/or italic styles.	p. 207

## Arranging and Opening Windows

To	Window	Comments	See
Arrange windows as overlapped	<b>Cascade</b>	To bring an overlapped window to the top, select the window from this menu or press ALT+F6 to cycle through the windows.	p. 63
Arrange windows as side-by-side	<b>Tile</b>	Useful to show several debug windows at the same time.	p. 63
Open an additional window on an open source file	<b>Duplicate</b>	Useful for viewing different sections of the same source file. Fonts can be unique in each window.	p. 79
Close all open windows	<b>Close All</b>	Prompts you to save files that have been changed since they were opened.	Help
Open the Watch window (or bring it to the top)	<b>Watch</b>	Watch window variables and expressions are saved with the project.  To add a variable or expression, type it into the Watch window during a debug session. (Or use QuickWatch to add the variable.)	p. 190  p. 190
Open the Locals window (or bring it to the top)	<b>Locals</b>	Shows all local variables and their values.	p. 180
Open the Registers window (or bring it to the top)	<b>Registers</b>	Shows all registers and flags. To change a register value, tab to the register and type over the current value. To toggle a flag value, tab to the flag and press the SPACEBAR.	p. 195
Open the Output window (or bring it to the top)	<b>Output</b>	The output from build utilities, including errors and warnings, appears here, as well as output from <b>OutputDebugString()</b> calls during debugging sessions.	p. 180

## Getting Online Help

To	<b>Help</b>	Comments	See
Find procedural help in Visual Workbench	<b><u>V</u>isual Workbench</b>	Opens Visual Workbench Help at the top-level Contents screen.	p. 65
Find help on specific compiler or linker options	<b><u>B</u>uild Tools</b>	Help covers the C/C++ Compiler Options and Linker Options dialog boxes and module-definition-file statements.	p. 71
Find help on the C or C++ language and functions	<b><u>C</u>/C++ Language</b>	C run-time functions and C/C++ language descriptions.	p. 65
Find help on the Microsoft Foundation Class Library	<b><u>F</u>oundation Classes</b>	Description of classes, member functions, and macros.	p. 65
Find help on Windows version 3.1 API functions	<b><u>W</u>indows 3.1 SDK</b>	Descriptions of all Windows version 3.1 APIs, messages, and macros.	p. 65
Search for a keyword in Help	<b><u>S</u>earch for Help On...</b>	Opens the Search dialog box with the selected word or word at the insertion point as a keyword and lets you choose the Help file to search.	p. 66
Get product support	<b><u>O</u>btaining Technical Support</b>	How to contact Microsoft Product Support to help solve your problems.	p. xv
Find information about your copy of Visual Workbench	<b><u>A</u>bout Visual C++...</b>	Displays a dialog box that shows software version and registered owner.	—

# Key Summaries

The tables in this section provide keyboard information and shortcuts for performing tasks such as editing, scrolling, text selection, searching, and debugging.

## Editing Keys

**Table 5.1 Insertion Point Movement Keys**

To move the insertion point	Press
One character left	LEFT ARROW
One character right	RIGHT ARROW
One word left	CTRL+LEFT ARROW
One word right	CTRL+RIGHT ARROW
One line up	UP ARROW
One line down	DOWN ARROW
To the first indentation of current line	HOME
To the first indentation of next line	CTRL+ENTER
To the end of line	END
To the beginning of file	CTRL+HOME
To the end of file	CTRL+END

**Table 5.2 Text Selection Keys**

To select	Press
Character to the left	SHIFT+LEFT ARROW
Character to the right	SHIFT+RIGHT ARROW
One word to the left	SHIFT+CTRL+LEFT ARROW
One word to the right	SHIFT+CTRL+RIGHT ARROW
Current line	SHIFT+DOWN ARROW
Line above	SHIFT+UP ARROW
To end of line	SHIFT+END
To beginning of line	SHIFT+HOME
One screen up	SHIFT+PAGE UP
One screen down	SHIFT+PAGE DOWN
To beginning of file	SHIFT+CTRL+HOME
To end of file	SHIFT+CTRL+END



**Table 5.3 Insert, Copy, and Tab Keys**

<b>To</b>	<b>Press</b>
Turn keyboard insert mode on or off	INS
Copy selected text to Clipboard, keeping it	CTRL+C CTRL+INS
Copy selected text to Clipboard, deleting it	CTRL+X SHIFT+DEL
Insert contents of Clipboard	CTRL+V SHIFT+INS
Copy current line to Clipboard, deleting it	CTRL+Y
Insert one blank line below	END, then ENTER
Insert one blank line above	HOME, then ENTER
Undo the last edit	CTRL+Z ALT+BACKSPACE
Redo the last edit	CTRL+A
Insert a tab	TAB
Toggle display of tab symbols	CTRL+ALT+T

**Table 5.4 Delete Keys**

<b>To</b>	<b>Press</b>
Delete one character to the left	BACKSPACE
Delete one character to the right	DEL
Delete to the end of the word	CTRL+T
Delete selected text and copy to Clipboard	CTRL+X SHIFT+DEL

**Table 5.5 Text Scrolling Keys**

<b>To scroll</b>	<b>Press</b>
Up one line at a time	CTRL+UP ARROW
Down one line at a time	CTRL+DOWN ARROW
Up one page at a time	PAGE UP
Down one page at a time	PAGE DOWN

**Table 5.6 Search Keys**

<b>To</b>	<b>Press</b>
Find the selected text	CTRL+F3
Repeat the last find	F3
Open the Find dialog box	ALT+F3
Search backward	SHIFT+F3
Find the next error	F4
Find the previous error	SHIFT+F4
Find matching brace	CTRL+]
Find next bookmark	F2
Find previous bookmark	SHIFT+F2

## Toolbar Keys

**Table 5.7 Toolbar Keys**

<b>To access the</b>	<b>Press</b>
Project Files button	CTRL+P
Open button	CTRL+O
Save button	CTRL+S
Find box	CTRL+F
Find box: search forward	RETURN
Find box: search backward	SHIFT+RETURN
Find Next button	F3
Compile File button	CTRL+F8
Build button	SHIFT+F8
Rebuild All button	ALT+F8
Toggle Breakpoint button	F9
QuickWatch button	SHIFT+F9
Run button	F5
Step Into button	F8
Step Over button	F10
Step Out button	SHIFT+F7

## Window Management Keys

**Table 5.8 Visual Workbench Window Management Keys**

To	Press
Switch to the next document window	CTRL+F6
Switch to the previous document window	SHIFT+CTRL+F6
Switch to the next window (includes all windows)	F6
Switch to the previous window (includes all windows)	SHIFT+F6
Switch to the previously active window	CTRL+TAB
Close the active window	CTRL+F4

## Build and Compile Keys

**Table 5.9 Build and Compile Keys**

To	Press
Compile the active source file	CTRL+F8
Build the project using dependency rules	SHIFT+F8
Build the project from the start	ALT+F8

## Browsing Keys

**Table 5.10 Browsing Keys**

To	Press
Jump to the definition of selected symbol	F11
Jump to the first reference of selected symbol	SHIFT+F11
Jump to the next reference in browser list	CTRL+NUMPAD+
Jump to the previous reference in browser list	CTRL+NUMPAD-
Return to original symbol location	CTRL+NUMPAD*
Expand active node one level	NUMPAD+
Collapse active node one level	NUMPAD-
Expand all nodes in a branch	NUMPAD*
Expand an entire graph	ALT+X

## Debugging Keys

**Table 5.11 Debugging Keys**

To	Press
Restart program execution from beginning	SHIFT+F5
Continue execution from current statement	F5
Execute program to insertion point position	F7
Execute next statement, tracing into function calls	F8
Single-step, stepping over functions calls	F10
Execute program out of current function and stop on first line after function call	SHIFT+F7
Open QuickWatch dialog box	SHIFT+F9
Open Modify Variable dialog box	CTRL+F9
Toggle hexadecimal display	ALT+F9
Toggle breakpoint	F9
Toggle mixed mode	CTRL+F7

## Alphabetic Guide to Build Options

The tables in this section cross-reference the CL and LINK command-line options with their corresponding dialog-box controls in the C/C++ Compiler Options and Linker Options dialog boxes. Visual Workbench uses these dialog boxes to assemble the command-line options passed to the NMAKE utility during a build.

By mapping each of the command-line options in an existing makefile to its corresponding control in the C/C++ Compiler Options or Linker Options dialog box, you can quickly bring your existing projects into Visual Workbench. Be sure to check the default options set by Visual Workbench for each project type if you are converting an existing project (see Tables 9.1 through 9.4 beginning on page 116 for the default options for each project type). Of course, you can also build existing projects from within Visual Workbench as external projects (see page 107).

See Chapter 9, “Customizing Build Options,” for descriptions of these compiler and linker options, default options set by Visual Workbench, and how to use the C/C++ Compiler Options and Linker Options dialog boxes.

## Compiler Options

Compiler options are set in Visual Workbench using the C/C++ Compiler Options dialog box. To open this dialog box, choose Project from the Options menu and then choose the Compiler button.

If you cannot find a particular CL option listed in Table 5.12, you can use the Other Options text box—available in the Custom Options category of the C/C++ Compiler Options dialog box—to enter most command-line options that do not have a matching dialog-box control.

**Table 5.12 Alphabetic Guide to Compiler Options**

<b>Option</b>	<b>Category</b>	<b>Control</b>
/AC	Memory Model	Model: Compact
/AH	Memory Model	Model: Huge
/AL	Memory Model	Model: Large
/AM	Memory Model	Model: Medium
/AS	Memory Model	Model: Small
/AT	Memory Model	Model: Tiny
/D	Preprocessor	Symbols and Macros to Define
/f	Code Generation	Code Generator: Fast
/f-	Code Generation	Code Generator: Optimizing
/Fa	Listing Files	Assembly
/Fc	Listing Files	Include Source and Machine Code
/Fl	Listing Files	Include Machine Code
/Fpa	Code Generation	Floating-Point Calls: Alternate Math
/Fpc	Code Generation	Floating-Point Calls: Coprocessor Calls
/FPc87	Code Generation	Floating-Point Calls: 80x87 Calls
/Fpi	Code Generation	Floating-Point Calls: Use Emulator
/Fpi87	Code Generation	Floating-Point Calls: Inline 80x87 Inst
/FR	Listing Files	Include Local Variables
/Fr	Listing Files	Browser Information
/G	Code Generation	CPU
/GA	Windows Prolog/Epilog	Generate Prolog/Epilog For: Protected Mode Application Functions
/Gc	Code Generation	Calling Convention: Pascal
/Gd	Code Generation	Calling Convention: C/C++
/GD	Windows Prolog/Epilog	Generate Prolog/Epilog For: Protected Mode DLL Functions
/GEe	Windows Prolog/Epilog	Protected Mode Options: Generate for __far Functions
/GEf	Windows Prolog/Epilog	Protected Mode Options: Emit Linker EXPDEF Records
/Gf	Custom Options	Eliminate Duplicate Strings

**Table 5.12** Alphabetic Guide to Compiler Options (*continued*)

<b>Option</b>	<b>Category</b>	<b>Control</b>
/Gp	P-Code Generation	Number of P-Code Entry Tables
/Gs	Code Generation	Disable Stack Checking
/Gt	Memory Model	New Segment Data Size Threshold
/GW	Windows Prolog/Epilog	Generate Prolog/Epilog For: Real Mode <code>__far</code> Non-Callback Functions
/Gw	Windows Prolog/Epilog	Generate Prolog/Epilog For: Real Mode <code>__far</code> Functions
/Gx-	Memory Model	Assume 'extern' and Uninitialized Data 'far'
/Gy	Custom Options	Enable Function-Level Linking
/I	Preprocessor	Include Path
/Mq	Custom Options	QuickWin Support
/ND	Segment Names	Data Segment
/NM	Segment Names	Module Name
/nologo	Custom Options	Suppress Display of Sign-On Banner
/NQ	Segment Names	P-Code Segment
/NT	Segment Names	Code Segment
/NV	Segment Names	V-Table Segment
/O1	Optimizations	Minimize Size
/O2	Optimizations	Maximize Speed
/Oa	Optimizations	Custom Optimizations: Assume no aliasing
/Ob0	Optimizations	Inline Expansion of Functions: Disable
/Ob1	Optimizations	Inline Expansion of Functions: Only <code>__inline</code>
/Ob2	Optimizations	Inline Expansion of Functions: Any suitable
/Od	Optimizations	Disable (Debug)
/Oe	Optimizations	Custom Optimizations: Global register allocation
/Of-	P-Code Generation	Disable P-Code Quoting
/Og	Optimizations	Custom Optimizations: Global-level common subexpression optimization
/Oi	Optimizations	Custom Optimizations: Generate intrinsic functions
/Ol	Optimizations	Custom Optimizations: Loop optimization
/Op	Optimizations	Custom Optimizations: Improve float consistency
/Oq	P-Code Generation	P-Code Optimization On

**Table 5.12** Alphabetic Guide to Compiler Options (*continued*)

<b>Option</b>	<b>Category</b>	<b>Control</b>
/Or	Optimizations	Custom Optimizations: Enable single point function exit
/Os	Optimizations	Custom Optimizations: Favor small code
/Ot	Optimizations	Custom Optimizations: Favor fast code
/Ov-	P-Code Generation	Sort Local Variables in Occurrence Order
/OV	Optimizations	Inline Function Size
/Ow	Optimizations	Custom Optimizations: Assume aliasing across function calls
/Ox	Optimizations	Custom Optimizations: Full optimization
/Oz	Optimizations	Custom Optimizations: Allow potentially unsafe loop optimizations
/u	Preprocessor	Undefine All Symbols
/U	Preprocessor	Individual Symbols to Undefine
/ymb	Custom Options (C++)	Representation Method: Best-case always
/vmg	Custom Options (C++)	Representation Method: General-purpose always
/vmm	Custom Options (C++)	General Purpose Representation: Point to multiple inheritance classes
/vms	Custom Options (C++)	General Purpose Representation: Point to single inheritance classes
/vmv	Custom Options (C++)	General Purpose Representation: Point to any class
/vd	Custom Options (C++)	Disable Construction Displacements
/W	Custom Options	Warning Level
/WX	Custom Options	Warnings as Errors
/X	Preprocessor	Ignore Standard Places of Include Files
/Yc	Precompiled Headers	Precompile through Header (C and/or C++), Precompile with Source (C and/or C++)
/Yu	Precompiled Headers	Precompile through Header (C and/or C++), Precompile with Source (C and/or C++)
/YX	Precompiled Headers	Automatic Use of Precompiled Headers
/Z7	Debug Options	Full (C7 Compatible)

**Table 5.12** Alphabetic Guide to Compiler Options (*continued*)

Option	Category	Control
/Za	Custom Options	Disable Microsoft Language Extensions (checked)
/Zd	Debug Options	Partial (Line Numbers Only)
/Ze	Custom Options	Disable Microsoft Language Extensions (not checked)
/Zi	Debug Options	Full, Use Program Database
/Zn	Listing Files	Don't Pack Information
/Zp	Code Generation	Structure Member Byte Alignment
/Zr	Code Generation	Check Pointers

## Linker Options

Linker options are set in Visual Workbench using the Linker Options dialog box. To open this dialog box, choose Project from the Options menu and then choose the Linker button.

If you cannot find a particular LINK option listed in Table 5.13, you can use the Other Options text box—available in the Miscellaneous category of the Linker Options dialog box—to enter most command-line options that do not have a matching dialog-box control.

**Table 5.13** Alphabetic Guide to Linker Options

Option	Category	Control
/ALIGN	Output	Segment Alignment
/CO	Output	Generate Debugging Information
/EXEPACK	Memory Image	Pack EXE File
/FARCALL	Memory Image	Translate Far Calls
/INFO	Output	Produce More Detailed Output
/LINE	Output	Include Line Numbers/Addresses in MAP
/MAP	Output	Create MAP File
/NOD	Input	Ignore Default Libraries Specific Libraries to Ignore
/NOE	Input	Prevent Use of Extended Dictionary
/NOI	Input	Distinguish Letter Case
/NOLOGO	Miscellaneous	Suppress Display of Sign-On Banner



**Table 5.13** Alphabetic Guide to Linker Options (*continued*)

<b>Option</b>	<b>Category</b>	<b>Control</b>
/NOPACKF	Memory Image	Don't Remove Unreferenced Packaged Functions
/ONERROR:NOEXE	Output	Prevent Creation of EXE on Linker Error
/PACKC	Memory Image	Pack Code
/PACKD	Memory Image	Pack Data
/SEG	Memory Image	Max. Number of Segments
/STACK	Memory Image	Stack Size
/TINY	Output	Produce COM file

# Using Visual Workbench

Chapter 6	The Visual Workbench Environment . . . . .	59
Chapter 7	Using the Editor . . . . .	75
Chapter 8	Using Projects . . . . .	93
Chapter 9	Customizing Build Options . . . . .	111
Chapter 10	Using the Browser . . . . .	161
Chapter 11	Debugging Programs . . . . .	179
Chapter 12	Customizing Visual Workbench . . . . .	199
Chapter 13	Using Visual Workbench with Other Visual C++ Tools . . . . .	209



## CHAPTER 6

# The Visual Workbench Environment

Microsoft Visual Workbench is an integrated programming environment that runs with the Microsoft Windows operating system. It integrates a text editor, browser, compiler, linker, debugger, make utility, and Help database.

A single development environment (see Figure 6.1) is used for both Visual C++ Standard Edition and Visual C++ Professional Edition with a few modifications to accommodate the extended capabilities of the professional edition. These modifications are limited to the C/C++ Compiler Options dialog box, because of the differences in compiler options, and the two dialog boxes where you set project types (New Project, accessed from the Project menu, and Project Options, accessed from the Options menu). Otherwise, Visual Workbench is identical for both versions.

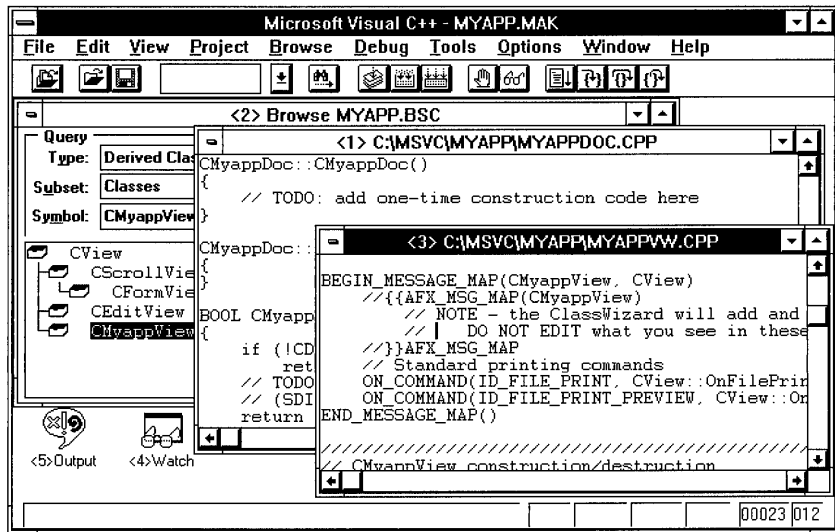


Figure 6.1 The Visual Workbench Environment

This chapter briefly introduces Visual Workbench. It describes how to start and quit Visual Workbench, how to arrange windows, and how to access the Help system. It also describes the toolbar and the status bar.

## Starting and Quitting Visual Workbench

After you have installed Visual Workbench, start it by double-clicking its icon in the Program Manager window.

If you had a project open when you last exited Visual Workbench, it is automatically loaded. Or, if you have installed Visual Workbench with command-line options to load a project or source files, the project or files will automatically load into Visual Workbench.

To quit Visual Workbench, choose Exit from the File menu. This returns you to the Windows Program Manager.

## Visual Workbench Features

Visual Workbench uses a multiple document interface (MDI) window that can contain several different types of child windows to let you edit source code, debug programs, and get status and error information. In addition, there is a toolbar for quick access to the most-often-used functions and a status bar for displaying build information and descriptions of toolbar buttons and menu items.

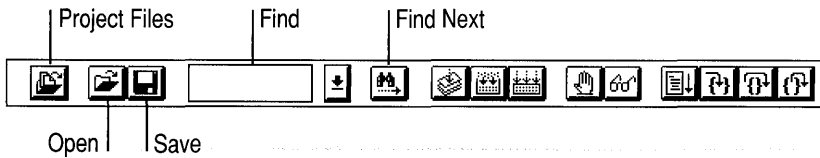
## The Toolbar

The toolbar appears beneath the menu bar. It provides shortcut commands for opening and saving files, finding text, and building, running, and debugging a program.

You can hide or display the toolbar with the Toolbar command on the View menu. When this command is checked, the toolbar appears. When it is turned off (not checked), the toolbar is hidden.

The controls (buttons and list boxes) on the toolbar are grouped according to function. There are toolbar buttons and list boxes for:

- Opening and saving source files.
- Finding text.
- Compiling files and building projects.
- Setting breakpoints and examining variables.
- Controlling program execution during debugging.



### Project Files

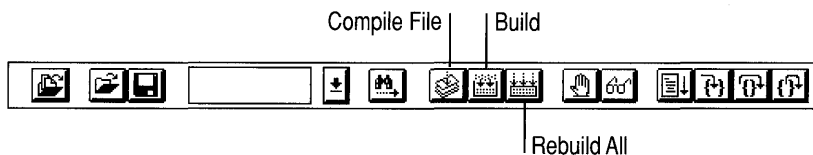
The Project Files button opens a drop-down list that gives you a convenient way to open source files associated with your active project. The drop-down list contains all editable files in the project list. Click the Project Files button, then click any filename in the list to open a source window on the file.

### Open, Save

Clicking the Open button opens the Open File dialog box, the same as choosing Open from the File menu. The Save button becomes available (not dimmed) when you make changes to the active source file and lets you save the file. (Chapter 7, “Using the Editor,” has more information on opening and saving files.)

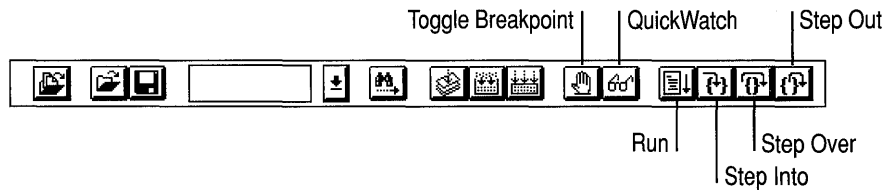
### Find, Find Next

The Find drop-down list box and the Find Next button let you quickly find text strings in a source file. To search for text in a source file, type the text in the Find box or select a text string from the drop-down list, which contains the last 16 text strings searched for. Then press ENTER, double-click the text string, or click the Find Next toolbar button. To repeat the last search, click the Find Next toolbar button. This lets you find multiple occurrences of a text string by simply clicking the Find Next button. Use the SHIFT key with any of these procedures to search backwards through the file.



### Compile File, Build, Rebuild All

You use these three build buttons on the toolbar to compile files and build projects. The Compile File button compiles the active file. Use the Build button to build the parts of the project modified since the last build, or the Rebuild All button to force a complete rebuild of all project files.



### Toggle Breakpoint, QuickWatch

These are two of the six debugging buttons on the toolbar. The Toggle Breakpoint button turns a breakpoint on or off at the insertion point. When the program reaches a breakpoint, or is paused between trace steps, you can use the QuickWatch button to examine and modify variables.

### Run, Step Into, Step Over, Step Out

The four debugging buttons grouped on the far right control program execution during a debugging session, letting you run the program to a breakpoint, single-step through the program, and handle function execution in various ways. See the following table for a description of each of the debugging buttons.

Table 6.1 provides a brief summary of the toolbar buttons.

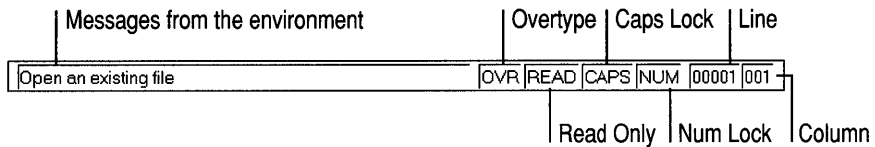
**Table 6.1** Toolbar Buttons

Button	Action
Project Files	Displays a list of project files to open.
Open	Displays the Open File dialog box.
Save	Saves the active source file to disk.
Find Next	Searches for another occurrence of text last searched for.
Compile File	Compiles the active source file.
Build	Builds the current project as modified since the last build.
Rebuild All	Rebuilds the current project from the start.
Toggle Breakpoint	Sets or clears a breakpoint at the insertion point.
QuickWatch	Displays the QuickWatch dialog box.
Run	Starts or continues program execution.
Step Into	Single-steps through instructions in the program. If this command is used when you reach a function call, the function is entered in single-step mode.
Step Over	Single-steps through instructions in the program. If this command is used when you reach a function call, the function is executed without stepping through the function instructions.
Step Out	Runs or continues a function to its completion and stops on the instruction immediately following the function call.

When a button cannot be used (such as is the case for many buttons when no project is open), the button is dimmed.

## The Status Bar

A status bar appears at the bottom of the main Visual Workbench window. The status bar provides information about Visual Workbench and the active source window.



The status bar can be displayed or hidden using the Status Bar command on the View menu. When displayed, it shows the following information about the active window:

- Summary help about the current operation.
- Messages from the environment: when you are viewing compile-time errors, for example, the current error message is displayed here.
- OVR: shows whether the editor is in overtype or insertion mode.
- READ: shows that the current file is read-only.
- CAPS: shows the state of the CAPS LOCK key.
- NUM: shows the state of the NUM LOCK key.
- Line: shows the current line.
- Column: shows the current column.

## Arranging and Displaying Windows

Visual Workbench uses the standard commands used by most Windows-based applications for displaying and arranging windows. These commands appear on the Window menu and include:

Command	Action
Cascade	Overlaps all open windows. The top of each window appears below the previous window's title bar. The active window appears at the front. See Figure 6.2.
Tile	Resizes windows so that all open windows appear on the screen without overlapping. See Figure 6.3.



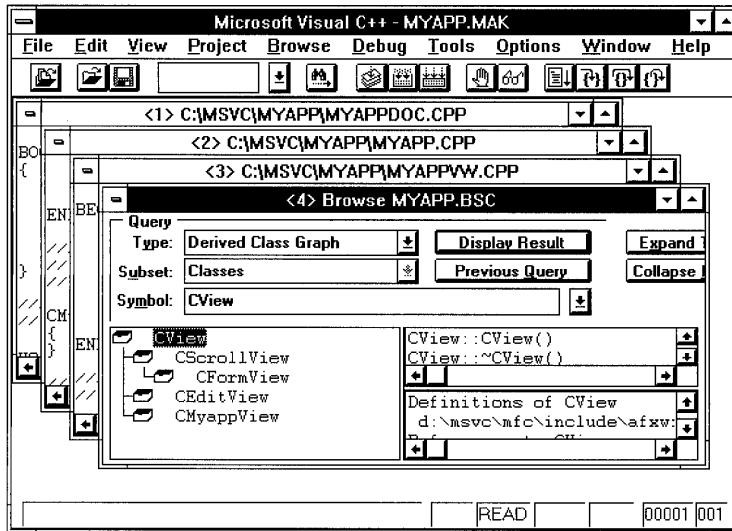


Figure 6.2 Cascaded Windows

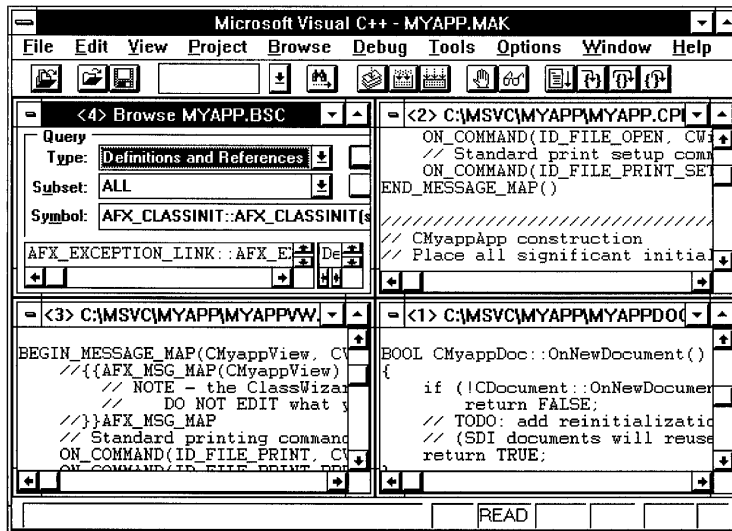


Figure 6.3 Tiled Windows

Visual Workbench windows support the standard Control-menu commands, including the Minimize command. When a window is minimized, an icon representing the window appears at the bottom of the main Visual Workbench window (see Figure 6.4). Double-click the icon to restore the window.

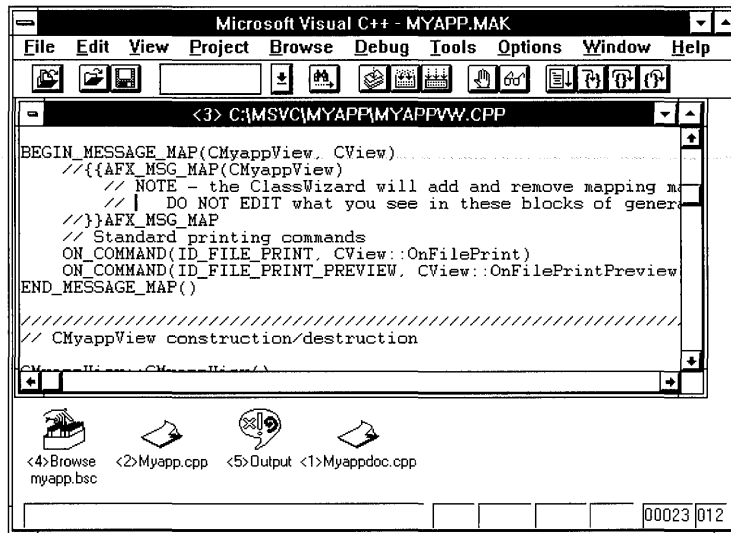


Figure 6.4 Minimized Windows

## Getting Help

Visual Workbench features an easy-to-use online reference system that provides access to information about Visual Workbench, build tools, C and C++ programming languages, C run-time libraries, Windows APIs, Microsoft Foundation classes, module-definition file statements, resource file statements, and compiler and linker errors.

You can get help in several different ways:

- Press the F1 key with the insertion point on a language keyword, library routine, or error number.
- In a menu, press F1 on a highlighted menu command.
- In a dialog box, press F1 or choose the Help button.
- From the Help menu, choose Search for Help On (or choose the Search button in a Help window).
- From the Help menu, choose a contents category.

The first four of these methods represent a “bottoms up” approach to finding help. That is, you go directly to the point in the Help system that documents the language element or Visual Workbench feature you need help on. You can press F1 to get immediate context-sensitive help on a language element, error code, or Visual Workbench feature.

Or, if you choose Search for Help On from the Help menu, you can search for help on keywords, functions, constants, class members, or errors. When a search topic appears in more than one Help file, the Search dialog box lets you choose the reference Help file to search. To search the file that contains Help on the Visual Workbench environment, you need to first open the file by choosing Visual Workbench from the Help menu, by pressing F1 in any Visual Workbench dialog box, or by pressing F1 with a Visual Workbench menu item highlighted. You then choose the Search button in the resulting Help window.

The fifth method of finding help (choosing a contents category from the Help menu) differs from the others in that it represents a “top down” approach. The Help menu contains several commands that open a Help window with a contents screen pertaining to the Help category:

- Visual Workbench
- Build Tools
- C/C++ Language
- Foundation Classes
- Windows 3.1 SDK

After choosing one of the commands from the Help menu, you navigate the Help system from the top contents screen down to the individual Help topics.

This is useful when you want to browse for information or need to see subjects from a global viewpoint before going to a specific section. This part of the Help system has been designed to make it easy to navigate through the vast amount of online information in Visual C++. Because this part of the Help system contains several new features, you might want to take a few minutes to read about it before trying it out. “Navigating Help from the Contents Screen” on the next page explains the new features.

## Getting Context-Sensitive Help

The F1 key gives you immediate help on language elements, error codes, and Visual Workbench features. The status bar provides a description of every menu command and toolbar button. Toolbar buttons, however, require a little care to display the button's description in the status bar without invoking the button.

### ► To get help on a keyword or an error:

1. Place the insertion point anywhere on the keyword or error.
2. Press F1.

If information on the keyword or error can be found in more than one Help file, a Search dialog box opens with a list of Help files from which to choose. Select a Help file and choose OK.

► **To get a Help description of a menu command using F1:**

1. Click the menu title to open the menu (or use the menu access key).
2. Use the arrow key to highlight the command.
3. Press F1.

You also get a brief description in the status bar of any toolbar button that you select. To see the description without activating the button (for buttons that are not dimmed) use the following procedure.

► **To get a status-bar definition of a toolbar button:**

1. Depress the toolbar button (move the mouse pointer over the button and press the mouse button).
2. With the toolbar button depressed, read the status-bar text.
3. Move the mouse pointer off the toolbar button before releasing the mouse button (releasing the mouse button with the mouse pointer on the toolbar button activates the button).

Every dialog box in Visual Workbench has an associated Help topic that describes the function of the dialog box and its elements.

► **To get a Help description of any dialog box:**

1. Open the dialog box.
2. Press F1 or choose Help.

Note that the C/C++ Compiler Options and Linker Options dialog boxes have additional Help capabilities to assist you in customizing compiler and linker options for your program. (See “Getting Help on Compiler and Linker Options” on page 71 to learn about this additional help capability.)

## **Navigating Help from the Contents Screen**

The Help system is divided into several sections, each covering a different major category of information. The Help menu contains several commands corresponding to the various categories of Help.

► **To open a contents screen on any category:**

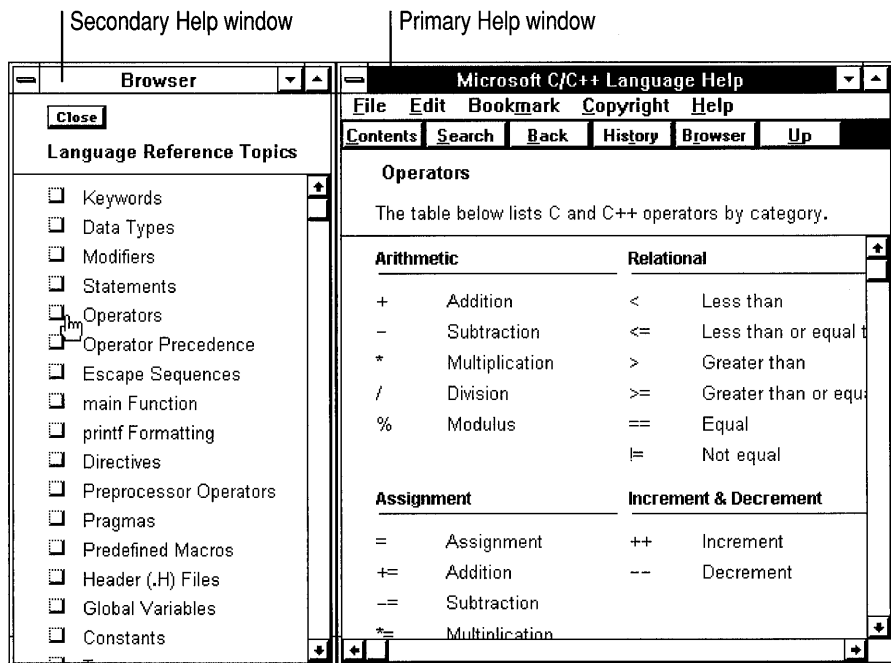
- Choose a Help category from the Help menu.  
–Or–
- If you already have a Help window open, click the Contents button.

The Visual Workbench command on the Help menu opens an environment Help window for Visual Workbench. All other commands open reference Help windows.

## Using the Secondary Help Window

The environment and reference Help systems are much alike and similar to other Help systems for Windows you have probably used. What is unique about Visual C++ Help is that you have the ability to open a secondary Help window that contains only topic links (see Figure 6.5). Choosing a topic link in the secondary window opens that Help topic in the primary Help window.

Visual Workbench Help uses the secondary Help window to display an index of all topics in the current category. Reference Help uses the secondary window to display a list of subcategory topics in the current category. In both Help systems, you choose the category for secondary window topics from the primary Help window's contents screen.



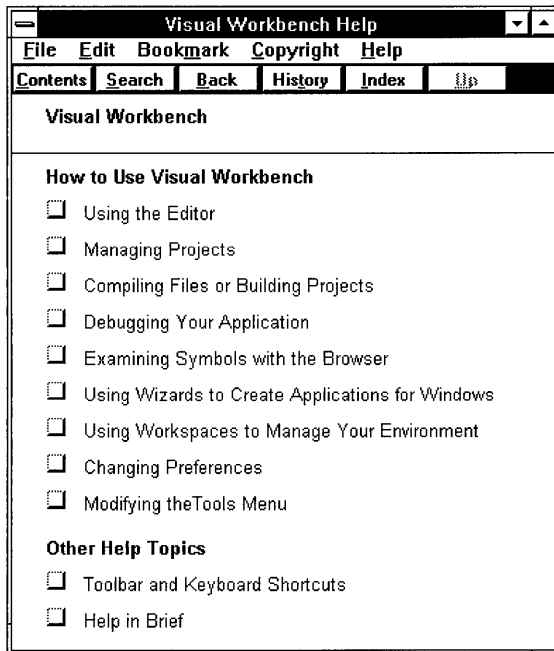
**Figure 6.5** The Secondary and Primary Help Windows

Having two Help windows open to browse for information lets you quickly jump around in the Help text, clicking entries in the secondary Help window and viewing the resulting text in the primary Help window. This can be useful when you know the general category where the information should be but are not sure of the exact location. In Visual Workbench Help, where the secondary window is used to provide an index of the current subject, it is much like keeping your thumb in a book's index while you open the book to different index references to look up information.

You may want to adjust the secondary Help window and the primary Help window to best utilize your screen area. You can then minimize the secondary Help window and close (or minimize) the primary Help window when not using them. (If you minimize the secondary Help window, instead of closing it, you will not need to resize and move it each time you use it.)

## Visual Workbench Help

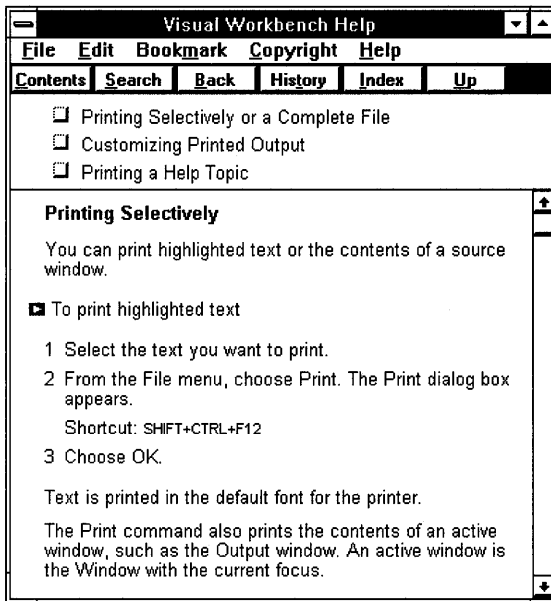
The Help window for the Visual Workbench environment has a contents window with several categories of Help (see Figure 6.6).



**Figure 6.6** The Visual Workbench Help Contents Screen

For Visual Workbench environment Help, each category you choose from the contents screen brings up a list of procedural topics with a brief description of the topic and a Help icon. Click the icon for the task that interests you and the topic screen appears.

Most topic screens have two parts: a scrolling region on the bottom, which contains all the Help text on the topic, and a nonscrolling region on the top, which contains indexes into the Help text in the scrolling region (see Figure 6.7). This lets you get to the information you want either by clicking the indexing button or by scrolling through the text.

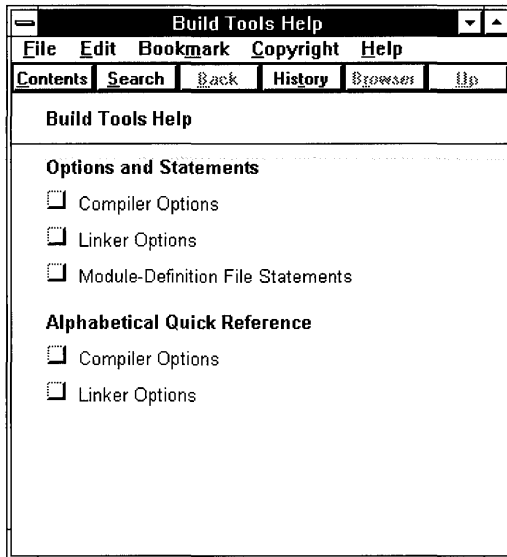


**Figure 6.7** A Visual Workbench Topic Screen

To get back to the Visual Workbench Help contents screen, choose the Contents button. To open the secondary Index window, choose the Index button at the top of the Help window.

## Reference Help Windows

When you choose any of the categories from the Help menu other than Visual Workbench, you display a contents screen of a reference Help window (see Figure 6.8).



**Figure 6.8** A Reference Help Contents Screen

Reference Help windows work similarly to Visual Workbench Help windows with a couple of small differences. First, when you select a category from the reference Help's contents screen, you get a list of subcategory links. (In Visual Workbench Help, these subcategories are in a different format and are annotated with brief descriptions.) Second, to open the secondary Help window, you choose the Browser button at the top of the Help window. The secondary Help window, called the Browser window here, simply displays the list of subcategory topic links—the same list that appears in the primary Help window when you choose a category from the contents screen.

The Browser Help window is especially useful when searching through reference Help for a particular subject, since you can use it to quickly flip through topics much as you might flip through a reference manual to find the section you're interested in.

## Getting Help on Compiler and Linker Options

The C/C++ Compiler Options and Linker Options dialog boxes, accessed from the Project Options dialog box, use an enhanced Help system because of the large amount of information available when setting compiler and linker options.

### F1 Help for Compiler and Linker Options

Help for compiler and linker options is context sensitive when you use the F1 key. You can get either a short pop-up description of an option in the options string, or a complete description of the option.



If you highlight an option in the options string (by double-clicking the option) and press F1, a pop-up window appears with a brief description of the option you selected. The pop-up description also tells you what dialog-box control is used to set or clear the option.

You can also get a complete description of any option by placing the dialog-box focus on the option's control (text box, list box, check box, or option button) and pressing F1. The resulting Build Tools Help window contains descriptions of all options in the current category, but with the text scrolled to the proper location to display the description of the option you selected.

### **Getting Help from Within the Help Window**

After you use F1 to open a Build Tools Help window on an option control, or choose Build Tools from the Help menu, you can navigate through the Build Tools reference Help the same as any reference Help in the Visual C++ Help system. The Build Tools contents screen gives you access to two types of Help:

- Options and Statements
- Alphabetical Quick Reference

Options and Statements lets you find information on any of the option categories in either the C/C++ Compiler Options or Linker Options dialog boxes, or any module-definition file statement.

Alphabetical Quick Reference gives you a fast means of getting both a short description of any CL or LINK option and finding the dialog-box category that contains the control for the option.

► **To access Help on any option category or module-definition file statement:**

1. Open the Build Tools Help window by one of the methods described previously.
2. If you used F1 to open the Help window, choose the Contents button.

The contents screen shows three Options and Statements categories:

- Compiler Options
- Linker Options
- Module-Definition File Statements

3. Choose one of the Options and Statements categories.

If you choose either Compiler Options or Linker Options, the Help window displays a list of option categories in the dialog box. If you choose Module-Definition File Statements, the Help window displays a list of statement links.

4. Choose the option category or module-definition file statement you want.

If you prefer to open a secondary window for browsing categories in either the C/C++ Compiler Options or Linker Options dialog boxes, or for browsing module-definition file statements, simply choose the Browser button. (If the Browser button is dimmed, you are at the contents screen and must choose one of the categories first.)

► **To look up a command-line option alphabetically:**

1. From the Help window, choose the Contents button.

The contents screen shows two Alphabetical Quick Reference categories:

- Compiler Options
- Linker Options

2. Choose one of the Alphabetical Quick Reference categories.

The Help window displays a list of all options for the category you chose, sorted alphabetically. Use the scroll bars to access all options.

3. Choose the name of an option.

A short description of the option appears, including the category in the dialog box that contains the control to enable or disable the option.



# Using the Editor

The integrated text editor is an important part of Visual Workbench. You use it to:

- Create, open, and save source files.
- Edit source files.
- Print source files.

Writing text with Visual Workbench is similar to using most other editors. For example, to start a new line, press ENTER. To leave a blank line between lines, press ENTER twice. If you make a mistake while typing, press BACKSPACE to delete the error.

This chapter provides information on how to manage, edit, and print source files. Most of the procedures involved in using the editor, such as file and text handling and moving around in a file, should seem familiar if you have used other Windows-based text editors.

## Managing Source Files

This section explains how to create, save, open, and close a file. To familiarize yourself with the steps, you can use the simple C++ program below, QWINTTEST.CPP, which can be built and run in Windows as a QuickWin application (see Chapter 3 for a demonstration of building a QuickWin program). You can also use this program to try the other editing features described in this chapter.

```
// QWINTTEST.CPP - A sample QuickWin program

#include <iostream.h>

void main()
{
    cout << "Hello C++ world \n";
}
```

## Creating and Saving Source Files

Visual Workbench lets you create source files and manage multiple source files. Each source window associated with a source file can retain its own fonts, sizing, and other window attributes. When you make changes to a source window, the Save button on the toolbar becomes available (not dimmed) to indicate that the information in the source window and source file differs.

### Creating a Source File

Use the New command on the File menu to create new source windows for entering text. To save the file on your hard disk, use the Save command on the File menu or the Save button on the toolbar.

► **To create a new source file:**

1. From the File menu, choose New.
2. Type your program in the new window.  
If you are following the example, type in QWINTTEST.CPP, the example program listed earlier.
3. Save your program as described in the procedures in the next section, “Saving Source Files.”

New files are labeled UNTITLED.*n* in the source-window title bar, where *n* is a sequential number, until they are saved.

---

**Note** Before you can save or close a window, it must be active. To make a window active, either switch to the window (by clicking anywhere in it) or choose the window name or number from the Window menu.

---

### Saving Source Files

In Visual Workbench, you can save programs using the Save and Save As commands on the File menu or the Save button on the toolbar. You can set save preferences—such as whether to be prompted before saving a file—in the Editor dialog box, accessed from the Options menu.

► **To save a source file:**

1. Switch to the source window.
  2. From the File menu, choose Save (CTRL+S).  
Or click the Save button on the toolbar.
  3. If your file is unnamed, Visual Workbench displays the Save As dialog box (see Figure 7.1). In the File Name box, type the filename.  
If you are using the sample program, name the file QWINTTEST.CPP.
  4. In the Drives and Directories list boxes, select the drive and directory in which to save the file.
  5. Choose OK.
- If QWINTTEST.CPP already exists in the directory you choose, you are presented with a message box asking if you want to replace the existing file. Choose Yes.

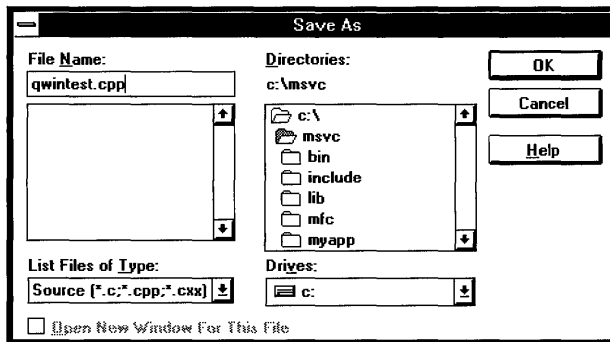


Figure 7.1 The Save As Dialog Box

If the file has already been named, the Save command saves any changes without displaying the Save As dialog box.

The Save button on the toolbar becomes available when you make any modification in the source file.

► **To save all open source files that are new or have been changed:**

- From the File menu, choose Save All.  
For every new source file that has not already been saved once, the Save As dialog box appears to let you type the filename and select the drive and directory in which to save the file.

► **To save a source file under a different name:**

1. Switch to the source window.
2. From the File menu, choose Save As.  
The Save As dialog box appears.
3. Type a new name for the file.
4. Select or clear the Open New Window For This File check box, depending on whether or not you want a new window created for the renamed file.

This check box is used to specify whether to open a new window for the renamed file and leave the original file's window open, or to use the original file's window for the renamed file. The status of this check box persists between invocations of this dialog box during the current Visual Workbench session.

5. Choose OK.

Renaming a file is useful for maintaining revised copies of a source file while keeping the original unchanged. Selecting the Open New Window For This File check box can be useful for duplicating one file in your project to be used as a template for a new file, when you want the original file left open. You can also use the Save As command to name and save a newly created file.

The Editor dialog box, accessed from the Options menu, has two options that relate to saving files. If you select the Save Before Running Tools check box, all open files are automatically saved whenever you run any tool installed on the Tools menu. This lets you integrate Visual Workbench with any tool that opens a source file also opened by Visual Workbench. Selecting the Prompt Before Saving Files check box causes Visual Workbench to prompt you before saving files whenever you perform a build or run a program from the Tools menu.

► **To set save options:**

1. From the Options menu, choose Editor.  
The Editor dialog box appears (see Figure 7.2).
2. Select Save Before Running Tools to automatically save source files before any tool on the Tool menu is run.
3. Select Prompt Before Saving Files to cause Visual Workbench to ask you whether the file should be saved each time you start a build (or run a tool if Save Before Running Tools is checked).
4. Choose OK.

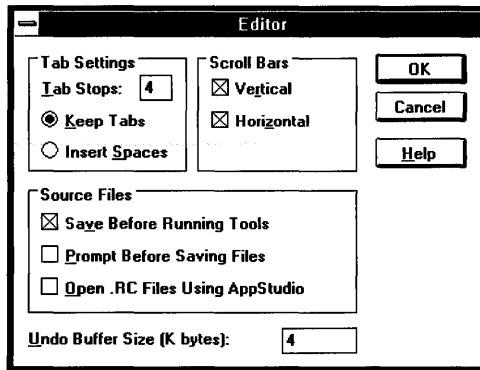


Figure 7.2 The Editor Dialog Box

## Opening and Closing Source Files

Because the editor works with any ASCII file, you can open and edit text files created with Visual Workbench or many other editors.

### Opening a Source File

When you open a source file, its name is added to the Window menu. Use the Duplicate command if you want to see more than one view of a source file, for instance to view different sections of a file at the same time. Do not use the File menu's Open command to attempt to open more than one view of a source file.

If you use the Duplicate command to display more than one copy of a file, each window on the file is titled *filename:n*, where *n* is a unique window number.

If no filename extension is given, Visual Workbench adds the extension (.C or .CPP) of the last file opened that had either a .C or .CPP extension.

► **To open a source file:**

1. From the File menu, choose Open (CTRL+O).
2. Select the drive and directory where the file is stored.

The default is the current drive and directory.



3. In the List Files of Type box, set the types of files to display.

Or type your own filename filter in the File Name text box. Separate filename extensions with semicolons (see “Using File Filters” on page 81). For example:

```
*.CPP ; *.H ; *.RC
```

The List Files of Type list box serves as a filter in displaying files to open. For example, C/C++ Header (\*.h) displays all files with the .H extension. The drop-down box lists commonly used filename extensions.

4. Choose OK to see a list of files in the selected directory.
5. Select the file from the list or type its name in the File Name box.
6. Choose OK.

---

**Tip** You can also open a file by using the Windows File Manager to display the file icon and then dragging and dropping the file icon into Visual Workbench. See Chapter 2 in the Microsoft Windows *User's Guide* for information on dragging icons.

---

## Opening Resource Files

Resource files (files with an extension of .RC) are source files that define resources such as menus and dialog-box controls and are compiled by the resource compiler as part of building an application for Windows. Although these are ASCII-based source files, they are also the primary information files used by App Studio.

When you open a resource file from either the Open File dialog box or the Project Files button on the toolbar, Visual Workbench does one of two things, based on the state of the Open .RC File Using App Studio check box in the Editor dialog box (accessed from the Options menu):

- If this check box is disabled (the default), opening a resource file is like opening any text file as a source window in Visual Workbench.
- If this check box is enabled, opening a resource file from Visual Workbench automatically invokes App Studio on that resource file.

When the option is enabled, Visual Workbench invokes App Studio and passes it the filename you selected. App Studio can open any resource file that the resource compiler can compile without errors, so you can use it with any existing resource files simply by opening the resource file from Visual Workbench or from App Studio.

Note that you can use the Windows File Manager to display a list of resource files and then drag and drop any resource file icon into Visual Workbench. The state of the Open .RC File Using App Studio check box in the Editor dialog box determines whether Visual Workbench or App Studio opens the file.

App Studio is installed on the Tools menu with the `$SRC` macro argument, which automatically opens the first resource file in the project list. (For more information on macro arguments, see “Using Argument Macros” on page 202.) Since you can open App Studio with your resource file from the Tools menu, the default action in the Open File dialog box (or Project Files toolbar button) is not to invoke App Studio on resource files. However, when you learn to use App Studio, you may decide you no longer need to hand-edit resource files and want the advantage of opening App Studio directly on files you select.

► **To open App Studio on a resource file:**

1. From the Options menu, choose Editor.
2. Select the Open .RC Files Using App Studio check box so that it is checked.
3. Choose OK to close the dialog box.
4. Open the resource file as described in “Opening a Source File” on page 79 or “Opening Project Source Files” on this page.

## Using File Filters

Visual Workbench provides the handy feature of letting you use your own special filename filters in the Open File and Save As dialog boxes. In Visual Workbench, these dialog boxes recall the filename filter that was last used. So you can use this feature to specify special file types that you always want to see. For example, the first time you open the Open File dialog box during a Visual Workbench session you can enter `*.CPP ; *.H` to look for only C++ source and header files. This filter appears each subsequent time you open that dialog box during the session.

## Opening Recently Used Files

A convenient way to open recently used files is to use the list at the bottom of the File menu, where the last four files that you opened and then closed appear. You can open one of these files by choosing its name from the list.

## Opening Project Source Files

Visual Workbench lets you quickly access source files associated with the active project by using the Project Files button at the left end of the toolbar. All editable files in the project list appear in the drop-down list that appears when you click the Project Files button (or press `CTRL+P`). Also, all include files that are added to the project as a result of include dependency scanning can be accessed here. For more information on include dependency scanning, see page 101.

► **To open any source file in the active project:**

1. Click the Project Files button on the toolbar.
2. Choose the source file from the drop-down list.

## Closing Source Files

All files are automatically closed when you quit Visual Workbench (you are prompted to save any altered files). You can also close any individual source file without exiting. If you have more than one window open on a source file, the file is not closed until all windows associated with it are closed.

► **To close a source file:**

1. Switch to the source window.
2. From the File menu, choose Close.  
Or double-click the document Control-menu box.  
Or press CTRL+F4.

If you are using the sample program, use one of these techniques to close the QWINTTEST.CPP source window.

If you create a new source file and try to close it before saving, a message appears asking if you want to save the changes before closing.

## Moving Around in Files

This section discusses some special techniques for moving around in source files in Visual Workbench.

► **To move to a specific line within the source file:**

1. From the View menu, choose Line.  
The Line dialog box appears.
2. Type the line number you want to move to.
3. Choose OK.

► **To set a bookmark in the source window:**

1. Move the insertion point to the line where you want to set a bookmark.
2. From the View menu, choose Toggle Bookmark, or press CTRL+F2.  
The line containing the bookmark is highlighted.

You can mark frequently accessed lines in your source file. Once a line is marked, use the View menu Next Bookmark and Previous Bookmark commands to move quickly to that line. Bookmarks can be cleared when you no longer need them.

► **To clear a bookmark:**

1. Move the insertion point to the marked line.
2. From the View menu, choose Toggle Bookmark.
  - The bookmark and the highlight are removed.

To remove all bookmarks, choose Clear All Bookmarks from the View menu.

► **To move to the next bookmark after the insertion point:**

- From the View menu, choose Next Bookmark, or press F2.

► **To move to the previous bookmark before the insertion point:**

- From the View menu, choose Previous Bookmark, or press SHIFT+F2.

► **To switch to a source window:**

- Click anywhere in the window (use the right mouse button to keep the insertion point from jumping to where you click).
  - Or–
- Choose the window from the Window menu.
  - Or–
- Use the shortcut key listed on the title bar (ALT+ <window number>).
  - Or–
- Use CTRL+F6 to cycle through all active document windows.
  - Or–
- Use F6 to cycle through all active windows (includes document windows and all other windows).
  - Or–
- Use CTRL+TAB to switch to the last active window (release the CTRL key to switch to the window).

► **To move from one brace to the matching brace:**

1. Place the insertion point immediately in front of a brace.
2. From the Edit menu, choose Find Matching Brace, or press CTRL+].

You can move from the opening brace to the closing brace or from the closing brace to the opening brace. This also works for parentheses and brackets.

## Using the Keyboard Commands

The editor includes a number of special keystrokes for editing and moving around in a source file. These are in addition to the familiar arrow keys, the SPACEBAR, and the ENTER key. For a complete reference to all keyboard commands, see pages 47 through 51 in Chapter 5, “Fast Track to Visual Workbench.”

To	Press
Move one word to the left	CTRL+LEFT ARROW
Move one word to the right	CTRL+RIGHT ARROW
Move to the first indentation of the current line	HOME
Move to the beginning of the current line	HOME, then HOME
Move to the first indentation of the next line	CTRL+ENTER
Move to the end of the current line	END
Move to the beginning of the file	CTRL+HOME
Move to the end of the file	CTRL+END
Undo the last edit	CTRL+Z ALT+BACKSPACE
Redo the last edit	CTRL+A
Delete to the end of the word	CTRL+T
Copy text to the Clipboard	CTRL+C CTRL+INS
Cut text to the Clipboard	CTRL+X SHIFT+DEL
Paste text from the Clipboard	CTRL+V SHIFT+INS
Move to the matching brace	CTRL+]
Insert a tab	TAB
Toggle display of tab symbols	CTRL+ALT+T

## Controlling the Source Window

The editor features a number of options that affect the source window. These include the ability to set tabs, highlight language syntax within the window, and make the window read-only.

## Setting Tabs

The editor supports tab stops in a source file. You can set the number of spaces a tab consists of, and then either save them as tabs or spaces when you save the file. You can also toggle the display of tab symbols in a source file.

► **To change tab settings:**

1. From the Options menu, choose Editor.

The Editor dialog box appears (see Figure 7.3).

2. In the Tab Stops box, type the number of spaces to be used as a tab stop.
3. Under Tab Settings, select the Keep Tabs option to treat tabs as a single tab character when the source file is saved.

Or select Insert Spaces to convert tabs to the number of spaces shown in the Tab Stops box.

4. Choose OK.

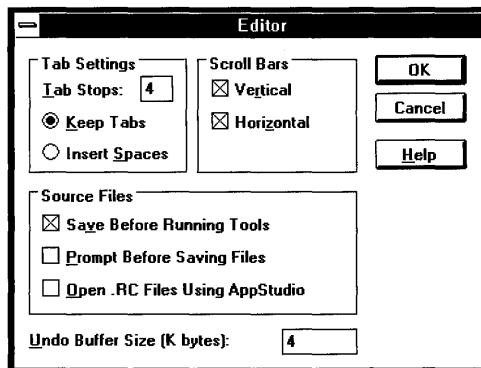


Figure 7.3 The Editor Dialog Box

► **To display or hide tab symbols:**

- Press CTRL+ALT+T.

Tab symbols are displayed as » wherever there is a tab in a source file. The CTRL+ALT+T keyboard shortcut is the only means of toggling the display of tab symbols.

## Highlighting Language Syntax

Visual Workbench highlights language keywords, identifiers, comments, and strings in different colors. This feature is useful when learning a language or when viewing lengthy and complex source files. For example, if you are editing a file with a .CPP extension, all C++ keywords are highlighted.

You can select the colors to use in highlighting language syntax items with the Color command on the Options menu. See Chapter 12, “Customizing Visual Workbench,” for additional information.

► **To highlight language syntax in the source window:**

1. From the Options menu, choose Color.  
The Color dialog box appears.
2. Select the Syntax Coloring check box.
3. Choose OK.

## Making a File Read-Only

The Read Only command on the Edit menu makes the active source file read-only. When you choose this command, the file cannot be edited.

This command is useful when you are viewing a program and don't want to accidentally make any changes to the file.

► **To make a source file read-only:**

1. Switch to the source window containing the file.
2. From the Edit menu, choose Read Only.

To cancel the command, choose it again.

A check mark next to the menu command and the word “READ” on the status bar at the bottom of the screen indicate that the source file is read-only.

## Finding and Replacing

Visual Workbench offers advanced find and replace capabilities. You can search for literal text or use regular expressions to find words or characters in the source window. You can search for text using three different methods:

- Select any word and press CTRL+F3 (or SHIFT+CTRL+F3 to search backwards)
- Use the Find dialog box
- Use the Find box and Find Next button on the toolbar

The Replace command on the Edit menu opens a dialog box that lets you find text and replace it with other text.

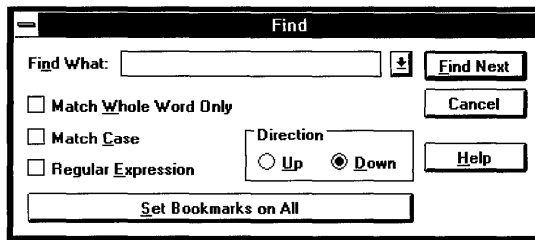
## Finding Text Using the Find Command

► **To find a character or group of characters in the active source window:**

1. Position the insertion point where you want to start the search.
2. From the Edit menu, choose Find.

The Find dialog box appears (see Figure 7.4).

3. In the Find What box, type the text you want to find, or select from the 16 previous instances of text searched for, which are listed in the drop-down box.
4. Select any of the Find options, as described below.
5. Choose Find Next.



**Figure 7.4** The Find Dialog Box

When you choose the Find command, the editor uses the location of the insertion point to select a default search string. If the insertion point is inside a word, that word is displayed as the search text in the Find What box.

If the insertion point is between words, the word to the right is displayed. If there is no word to the right, the word to the left is displayed. If that is not possible, nothing is displayed.

The Find dialog box has these choices for locating text:

### Match Whole Word Only

Locates separate occurrences of the search text. If you don't select this option, the editor finds embedded occurrences, for example, "main" in remainder.

### Match Case

Performs a case-sensitive search.

### Regular Expression

Finds text using regular expressions. You can find more information about using regular expressions in the Using the Editor section of Help.

### Direction

The Up option searches from the insertion point to the beginning of the file.

The Down option searches from the insertion point to the end of the file.



After selecting the search options, you can choose either the Find Next or the Set Bookmarks on All button. If you choose Find Next and the text is located, the source window jumps to the text and highlights it. Choose Find Next again to move to the next occurrence of the text.

If you choose Set Bookmarks on All, bookmarks will be set at all occurrences of located search text. See “Moving Around in Files” on page 82 for more information on using bookmarks.

## Finding Text Using the Toolbar

The Find box and Find Next button on the toolbar let you quickly look for text without using a menu dialog box.

► **To use the toolbar Find box and Find Next button to search for text:**

1. Position the insertion point where you want to start the search.
2. Move the insertion point to the Find box on the toolbar (CTRL+F) and type the text you want to search for.  
Or select from the 16 previous instances of text searched for, which are listed in the drop-down box.
3. To search forward in the file, press ENTER.  
Or double-click the selection.  
Or click the Find Next button.
4. To search backward in the file, press SHIFT+ENTER.  
Or press SHIFT and double-click the selection.  
Or press SHIFT and click the Find Next button.
5. To repeat the search, click the Find Next button (hold down SHIFT for a backward search).  
Or, with the insertion point in the Find box, press ENTER or SHIFT+ENTER to search forward or backward, respectively.

If the text you entered can be found, the active source window jumps to it and highlights the text.

► **To quickly find the next or previous occurrence of a text string:**

1. Select the text or place the insertion point anywhere in a single word.
2. Press CTRL+F3 to find the next occurrence of the selected text or word.  
Or press SHIFT+CTRL+F3 to find the previous occurrence of the selected text or word.

If you use the CTRL+F3 keys with the selected text or word, the text is automatically entered in the Find list box. You can then continue to press CTRL+F3 to jump to each occurrence of the selected text or word.

The current state of the following search options in the Find dialog box applies to searches using the toolbar's Find box:

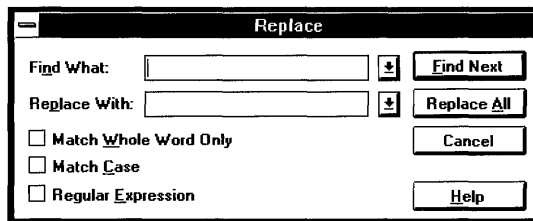
- Match Whole Word Only
- Match Case
- Regular Expression

## Replacing Text

### ► To find and replace text:

1. From the Edit menu, choose Replace.

The Replace dialog box appears (see Figure 7.5).



**Figure 7.5** The Replace Dialog Box

2. In the Find What box, type the search text or regular expression.  
Or choose from the 16 previous instances of text searched for, which are listed in the drop-down box.
3. In the Replace With box, type the replacement text.  
Or choose from the 16 previous instances of replacement text, which are listed in the drop-down box.
4. Select any search options you need.  
These options are the same as those in the Find dialog box (see “Finding Text Using the Find Command” on page 87).

5. Choose either Find Next to find the next occurrence of the text or Replace All to find and replace all occurrences of the text.

If you choose Find Next, the next occurrence of the text is found if it exists, and the Replace dialog box is replaced by an abbreviated dialog box containing only Replace control buttons. You can replace the found text, find the next occurrence of the text, replace all occurrences of the text, or cancel the operation using this dialog box.

For information on default search strings, see “Finding Text Using the Find Command.”

► **To repeat the last find or find and replace:**

- Choose Find Next in either the Find or Replace dialog box.

An abbreviated portion of the Find or Replace dialog box remains active until the search reaches the starting location (after wrapping) or until you choose Cancel.

## Printing

If you have installed a printer driver and connected to a printer port within Windows, you can print text from the active window.

If you have more than one printer, you can change the default selection in the Print dialog box, which you open by choosing Print on the File menu.

For information on installing printers, see Chapter 5, “Control Panel,” of the *Windows User's Guide*.

► **To print the entire contents of the active window:**

1. From the File menu, choose Print.

The Print dialog box appears (see Figure 7.6).

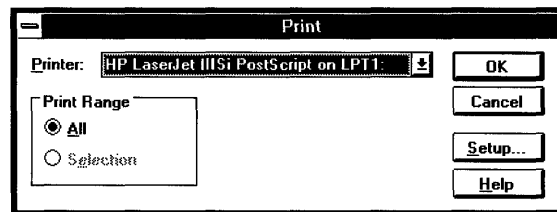


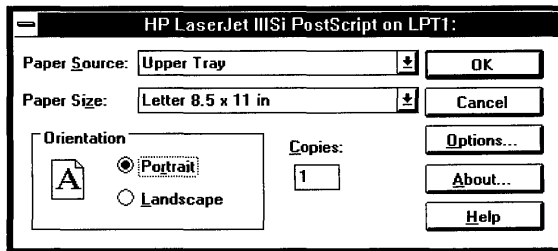
Figure 7.6 The Print Dialog Box

2. Under Print Range, select the All option.
3. Choose OK.

► **To print selected text in the active window:**

1. Select the text to be printed.
2. From the File menu, choose Print.
3. Under Print Range, select the Selection option.
4. Choose OK.

Choose the Setup button to set printer-specific information. The Setup button opens a printer-supplied dialog box (see Figure 7.7), in which you can change the paper size, the orientation of the printing, the number of copies, and various other printing options.



**Figure 7.7** A Printer Setup Dialog Box

► **To change the default printer:**

1. From the File menu, choose Print.
2. If your computer is connected to more than one printer, choose a printer from the drop-down list.
3. Choose OK.



# Using Projects

Projects are the cornerstone of Visual Workbench. A project keeps track of the various files and libraries that are needed to build a program or a library. It also contains information about compiler and linker options. By creating a project and selecting a project type, you can specify the kind of target file you want to generate when you build the project. Projects also contain information saved in workspaces, such as window sizes and positions.

Projects are stored on disk as two files: a makefile (.MAK extension) and a status file (.VCW extension). The makefile, which is compatible with the NMAKE utility, specifies the rules for the compilers and linker to build the target file. Visual Workbench calls this file the project file. When you open the project file from within Visual Workbench, the status file is automatically loaded.

To create a project, you add the filenames associated with the project to the project list, which Visual Workbench then uses to construct the project file. When you open a project, all source filenames in the project list are made available in the toolbar's Project Files drop-down list. This gives you quick access to the main source files.

Projects also speed development time by recompiling only files that have changed since the program's last compilation or build. For example, if your project has six source files and you edit only one of the files between builds, only that file is recompiled before linking. You also have the option to rebuild all files from the start if you want.

This chapter introduces Visual Workbench projects. It describes:

- Each of the project types
- How to create, open, and close a project
- How to build a project
- How Visual Workbench scans include files for dependencies
- Workspaces and how to use them
- How to use external projects and makefiles created by other editors

## Project Types

Every project must contain a project type, which specifies the kind of target file to be generated by a build. You choose a project type when you first create the project, using the New Project dialog box (accessed from the Project menu). To change the project type of an existing project, you use the Project Options dialog box (accessed from the Options menu). For step-by-step instructions on how to create a project and choose a project type, see “Creating a Project” on page 97.

When you select a project type, the appropriate compiler and linker options are automatically set to generate the target file. These options are sufficient for most needs, although you can use the C/C++ Compiler Options and Linker Options dialog boxes to fine-tune them.

Visual C++ Standard Edition provides a basic set of five Windows-based project types plus static libraries. Visual C++ Professional Edition includes these project types plus several more. This chapter describes the project types for both editions.

### Visual C++ Core Project Types

The following project types are available with both Visual C++ Standard Edition and Visual C++ Professional Edition:

- Windows application
- Windows dynamic-link library
- QuickWin application
- Visual Basic custom control (custom VBX control)
- Static library

You can use the Microsoft Foundation Class Library with any of these types if you select the Use Microsoft Foundation Classes check box in either the New Project or Project Options dialog box. When you select Use Microsoft Foundation Classes, an extra library is linked during the build. Selecting this check box provides that library as a default linker option.

### Windows Application

Windows-based applications have a full graphical interface and run only with Windows. They are developed using standard Windows API functions, or using the Microsoft Foundation Class Library. A Windows-based application filename has an .EXE extension.

Windows-based applications generally use the following files in the project list:

- Source files (.C, .CPP, or .CXX)
- Module-definition file (.DEF)
- Resource file (.RC)

### Windows Dynamic-Link Library

Dynamic-link libraries contain functions that are called at run time by Windows-based programs. All Windows APIs, for example, are kept in dynamic-link libraries. Creating a dynamic-link library is a good way to write code that can be shared by different programs running in Windows at the same time. A Windows-based DLL application filename has a .DLL extension.

Dynamic-link libraries generally use the following files in the project list:

- Source files (.C, .CPP, or .CXX)
- Module-definition file (.DEF)
- Resource file (.RC)

### QuickWin Application

A QuickWin application is a standard input/output MS-DOS program with a Windows shell. It runs only with the Windows operating system. Building an MS-DOS program as a QuickWin program is a quick way to create a Windows-style program, or adapt an existing MS-DOS program to Windows, without having to learn the basics of programming for Windows. When you run a QuickWin application, the program output appears in a QuickWin multiple document interface (MDI) child window as it would on your display monitor in MS-DOS. The program can use standard MS-DOS functions such as **printf** and **scanf** or C++ iostream operators to process input and output. A QuickWin application filename has an .EXE extension.

---

**Note** QuickWin programs can only run in Windows standard or 386 enhanced mode. Do not run a QuickWin program in real mode.

---

QuickWin applications generally require the following files in the project list:

- Source file (.C, .CPP, or .CXX)
- Module-definition file (.DEF)

If you do not include the module-definition file, Visual Workbench automatically includes one for you (for QuickWin applications only).



## Visual Basic Custom Control

Visual Basic custom controls (also called custom VBX controls) are essentially dynamic-link libraries with a .VBX extension that can be used as controls from within a Visual Basic program. They can also be used as resources in App Studio for Visual C++ applications. You can learn how to write Visual Basic custom controls from the *Control Development Guide*, which is included with Microsoft Visual Basic for Windows, Professional Edition. For more information on using Visual Basic custom controls in Visual C++, see Chapter 3, "Using the Dialog Editor," in the *App Studio User's Guide*.

Visual Basic custom controls generally require the following files in the project list:

- Source files (.C, .CPP, or .CXX)
- Module-definition file (.DEF)
- Resource file (.RC) (optional)

## Static Library

Static libraries are standard libraries that can be created directly from the build using object files that belong to the project. To create a static library (.LIB), create a project with normal project files and/or other object (.OBJ) files in the project file list, then build the project.

The generated library file is composed of all the object files in the project list and/or all object files generated by the build. This is a quick way to develop your own libraries from within Visual Workbench, without having to run a library-manager utility from an MS-DOS session, or outside of Windows.

## Visual C++ Professional Edition Project Types

Visual C++ Professional Edition provides the same project types as Visual C++ Standard Edition plus these additional project types:

- Windows P-code application
- MS-DOS application
- MS-DOS P-code application
- MS-DOS Overlaid application
- MS-DOS COM application (.COM)

## Windows P-Code Application

P-code is a special type of executable code that is smaller than machine code and uses an interpreter, incorporated in the executable file, to translate the p-code into machine code at run time. P-code programs are smaller but slower than normal programs that compile to machine code. In a Windows-based p-code application, it is typical to compile user-interface sections of code as p-code (where speed is not as

important) and computational sections as native code. The default compiler options for this project type include `/Oq`, which compiles the entire program using p-code. Use the directive `#pragma optimize("q", off)` in your source code for functions you want to be in native code.

### MS-DOS Application

MS-DOS applications are traditional character-based programs that run with either MS-DOS or Windows. MS-DOS programs are appropriate when the program doesn't need the Windows interface to execute.

### MS-DOS P-Code Application

P-code is a special type of executable code that is smaller than machine code and uses an interpreter, incorporated in the executable file, to translate the p-code into machine code at run time. P-code programs are smaller but slower than normal programs that compile to machine code. The default compiler options for this project type include `/Oq`, which compiles the entire program using p-code. Use the directive `#pragma optimize("q", off)` in your source code for functions you want to be in native code.

### MS-DOS Overlaid Application

This type uses the Microsoft Overlaid Virtual Environment (MOVE) to create programs that would otherwise be too large to run in conventional memory. Overlays are sections of a program that are loaded into memory from the executable file only as needed. To learn how to create an overlaid program, see Chapter 3, "Creating Overlaid MS-DOS Programs," in the *Command-Line Tools User's Guide*.

### MS-DOS COM Application (.COM)

This project type produces an executable file with a file extension of `.COM`. It uses the tiny (`/AT`) memory model compiler option and the `/TINY` linker option to place both code and data in a single physical memory segment. The restrictions on using this project type are that the program cannot use far references and must keep its data and code requirements less than 64K bytes.

## Creating a Project

A program is built from source files and libraries. The source files are compiled into object files and then linked with the libraries to create a program. In Windows-based programs, you need to link additional files such as resource files (files that contain resources like icons, menus, and dialog boxes) and module-definition files (files that contain information about the program).

Before you create a program, you need to create a project for it. A project can consist of only one source file or of many source, resource, and library files.

Everything that applies to projects for programs also applies to projects for libraries.

---

**Note** If your program consists of only one file, you can build and run it without creating a project. (For QuickWin programs, the module-definition file is automatically included.) However, it is a good idea to create a project if you have unique compiler or linker settings, since these will be lost if another project is loaded or Visual Workbench is restarted.

---

To familiarize yourself with the steps in building a project, you can use the files listed below to create a multifile C++ project. Although there is already a project called HELLO.MAK in the directory, you can experiment by creating a second project with a different name in that directory. This example assumes that you have installed the Microsoft Foundation Class Library sample programs. If you haven't installed the MFC sample programs, but have installed the SDK sample programs, you can use one of the programs in the \MSVC\SAMPLES directory, such as GENERIC, for this example.

File	Directory
HELLO.CPP	\MSVC\MFC\SAMPLES\HELLO
HELLO.DEF	\MSVC\MFC\SAMPLES\HELLO
HELLO.RC	\MSVC\MFC\SAMPLES\HELLO

The directory listed above is the default directory created by the Setup program. If you changed the default directory names during installation, the location of the required files will be different.

► **To create a project:**

1. From the Project menu, choose New.

The New Project dialog box appears.

If you know the directory for the project, you can simply type the project name, including the complete path, in the Project Name box and skip to step 6. For example, if you installed Visual C++ on drive C, enter the following in the Project Name box:

```
C:\MSVC\MFC\SAMPLES\HELLO\MYHELLO.MAK.
```

If you would rather use the graphical directory browser, continue with step 2.

2. Choose the Browse button to browse and change directories.

The Browse dialog box appears. This dialog box allows you to change directories and look at filenames. The purpose of browsing your directories is to make sure you use a unique name for the project and to change to the correct directory for the project.

- Use the Directories and Drives list boxes to change to the drive and directory containing the example files.

You should see the name of the existing project file (HELLO.MAK) already there.

- In the File Name box, type the new project filename.

For example, type MYHELLO.

All project files have a .MAK extension. You don't need to type any more than the first eight characters of the filename. The .MAK extension is automatically added when you choose OK.

- Choose OK.

The Browse dialog box disappears, and the project name you specified appears in the Project Name box preceded by its path. You must now specify a project type.

- From the drop-down list in the Project Type box, select a project type.

For this example, select Windows application (.EXE). Be sure the Use Microsoft Foundation Classes check box is selected if your application uses the foundation classes, as does the HELLO sample.

- Choose OK.

The Edit – *Projectname* dialog box appears (see Figure 8.1).

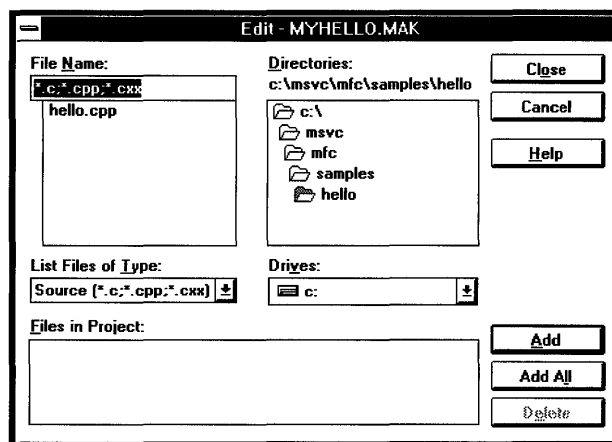


Figure 8.1 The Edit – *Projectname* Dialog Box

- Add the necessary filenames in the project to the project list.

In this example, you need to add HELLO.CPP, HELLO.DEF, and HELLO.RC. (HELLO.H cannot be added now since all include files are automatically found and added when the dialog box is closed.) If you need help using this dialog box, see the procedure titled “To add a file to the project” on page 101.

9. Choose the Close button to save the project and scan all the source files for include files.

Whenever you close (not cancel) the Edit – *Projectname* dialog box, Visual Workbench recursively scans all files in the project list for **#include** directives and automatically adds any include files it finds to the project list. For more information on include file dependencies, see “Using Include Dependencies” on the next page.

## Opening and Closing Projects

Although you can maintain many projects on your disk, Visual Workbench can work with only one project at a time. You can recall any previously created project by opening it. When you open an existing project, the current project is automatically closed, or you can close it manually.

► **To open an existing project:**

1. From the Project menu, choose Open.
2. From the file list, select a project file (.MAK extension).
3. Choose OK.

---

**Tip** You can also open a project by using the Windows File Manager to display the project file icon and then dragging and dropping the project file icon into Visual Workbench. See Chapter 2 in the Microsoft Windows *User's Guide* for information about dragging icons.

---

► **To close a project:**

- From the Project menu, choose Close.
- Or–
- Open another project.
- Or–
- Create another project.

To learn how to open a makefile created outside the Visual Workbench environment, see “Using External Projects” on page 107.

## Adding and Deleting Project Files

When you create a project, you can add files to the project. As projects grow and change, however, you may need to add files to or delete files from the project.

When you delete a file from the project, only the reference to that file is removed. The actual file is not deleted from the hard disk.

Choose the Edit command from the Project menu to open a dialog box where you can add or delete project files. This dialog box is identical to the dialog box used to add files to a new project (see Figure 8.1).

► **To add a file to the project:**

1. From the Project menu, choose Edit.

The Edit – *Projectname* dialog box appears.

2. Use the List Files of Type drop-down list box to set the type of files that appear in the File Name box.
3. From the file list in the File Name box, select the file to add to the project and choose Add.

Or double-click the file in the file list.

Or you can choose Add All to add every file that appears in the file list to the project list. This can save steps if you have several source files you want to add.

4. Repeat steps 2 and 3 for each file (or set of files) you want to add to the project.
5. Choose Close when you finish editing the list of project files.

An include dependency scan now takes place. For more information, see the next section, “Using Include Dependencies.”

► **To delete a file from the project:**

1. From the Project menu, choose Edit.

The Edit – *Projectname* dialog box appears.

2. Select the file from the Files in Project box and choose Delete.

Or double-click the file in the Files in Project box.

3. Repeat step 2 for each file you want to delete from the project.
4. Choose Close when you finish editing the list of project files.

An include dependency scan now takes place. For more information, see the next section, “Using Include Dependencies.”

## Using Include Dependencies

Visual Workbench creates include dependency lists whenever you create or edit a project. It does this by recursively scanning all the project files for **#include** directives and adding each included filename to the project list. Both source files (.C, .CPP, or .CXX) and resource files (.RC) are scanned, so the include files in the dependency list can have extensions of .H, .HXX, .HPP, .INC, .FON, .CUR, .BMP, .ICO, or .DLG. Since these file types are automatically added to the project list, you

cannot add them manually. These files are part of the project list and are available in the Project Files drop-down list, accessed from the toolbar's Project Files button.

An include dependency list associates each included file with the files that include it and are therefore dependent upon it. This allows Visual Workbench to generate the correct build dependencies. For example, when you edit a header file and perform a build, the build compiles only the files that include the modified header file.

An include dependency scan takes place automatically when you close the Edit – *Projectname* dialog box, after you have either created or edited the project list. If you subsequently include a new file in any file in the project list, you should regenerate the include dependencies for the project.

- ▶ **To regenerate include file dependencies for the entire project:**
  - From the Project menu, choose Scan All Dependencies.
  
- ▶ **To regenerate include file dependencies for the active file:**
  - From the Project menu, choose Scan Dependencies *Filename* (*Filename* appears on the menu as the name of the active source file).

Whenever a dependency scan occurs, a status window appears showing the status of the include dependency scan as it searches files. You can cancel this scan at any time by choosing the Cancel button, in which case the previous include dependency list remains in effect.

Visual Workbench scans both bracketed include files (*<includefile.h>*) and user include files ("*includefile.h*"). There are a large number of system include files shipped with Visual C++ that are unlikely to be changed. Since it would be inefficient to scan these include files when they are included by a project file, Visual Workbench uses an exclusion file called SYSINCL.DAT that contains a list of these include files. Whenever Visual Workbench scans for include dependencies, it uses this list to exclude any filenames it finds that match those in this list.

The exclusion file is an editable file, and you can use it to fine-tune your build dependencies. For example, if you have a large header file that you never change, you could add it to the list in the exclusion file. Then, whenever you generate include dependencies, that file will not be scanned. As another example, you may want a Microsoft system include file scanned because you are altering it, in which case you could remove that include filename from the exclusion file. Visual Workbench never overwrites this file, but it does build a new file containing all system include files shipped with Visual C++ whenever it doesn't find a SYSINCL.DAT file at startup. So you can easily regenerate the original file at any time simply by deleting or renaming the current exclusion file.

# Project Compiler and Linker Options

Besides file references, projects also contain information about compiler and linker settings to use. When you select a project type, the project configures the compiler and linker options to build that type of program or library. These options, and any compiler and linker options you set yourself, are automatically saved with the project information when you perform a build.

To customize the compiler and linker options generated by the project type, use the Customize Build Options group in the Project Options dialog box. This group contains three command buttons that open dialog boxes to let you set compiler, linker, or resource compiler options. To learn more about these dialog boxes, see Chapter 9, “Customizing Build Options.”

## Building a Project

You create a project so that you can eventually build a program or a library. Visual Workbench lets you either build or rebuild a project. When you build a project, only the files in the project list that have changed since the last build are included in the build. When you rebuild, all files are included in the build.

Before you build a project, you need to determine whether you want to produce a debug or release version of the program. The debug version contains information that can be used by Visual Workbench’s integrated debugger or by the Microsoft CodeView debugger. The release version contains no debug information and therefore is smaller and faster. To learn how to use the debugger on a debug version of a program for Windows, see Chapter 11, “Debugging Programs.”

► **To select debug or release build options:**

1. From the Options menu, choose Project.  
The Project Options dialog box appears.
2. Under Build Mode, select either Debug or Release.
3. Choose OK.

► **To build a project:**

- From the Project menu, choose Build *Targetname* (*Targetname* represents the name of the target file displayed on the menu).  
–Or–
- Click the Build button on the toolbar.



► **To rebuild a project:**

- From the Project menu, choose Rebuild All *Targetname* (*Targetname* represents the name of the target file displayed on the menu).  
–Or–
- Click the Rebuild All button on the toolbar.

After you choose Build or Rebuild All, the Output window provides information from the tools used by the build. The Output window is also where errors and warnings are reported. Since the build occurs in the background, you can continue to use Visual Workbench. Some menu commands and toolbar buttons are disabled during a build (for example, you are not allowed to exit Visual Workbench during a build without first stopping the build).

An audible message notifies you when the build has completed. The audible message corresponds to one of three standard system events in Windows:

System Event	Indicates
Asterisk	Build has completed without errors or warnings
Question	Build has completed with warnings
Exclamation	Build has completed with errors

If you have a sound driver installed, you can use the Sound application in the Windows Control Panel to assign these system events to different sounds. Otherwise, all audible messages simply issue a beep.

To abort a build at any time, choose the Stop Build command on the Project menu. The build will abort as soon as the currently executing build tool finishes.

If no errors are reported, the build is successful. If the project type is a program, you can run it by choosing Execute *Targetname* from the Build menu or debug it by choosing Go from the Debug menu or by clicking the toolbar's Run button. Choose Exit from the program's File menu to quit the program and return to Visual Workbench.

## Using a Workspace

A workspace is a convenient way of saving and restoring an arrangement of windows associated with a project. Loading a workspace also restores font settings and the status of the toolbar and status bar (displayed or hidden).

You can load a workspace to quickly re-create a work environment. For example, suppose you like to debug with a narrow Registers window to the right of your source window and Watch and Locals windows below your source. If you save this arrangement as a workspace, you can load it at the start of each debugging

session instead of positioning and sizing the windows. Workspaces also keep track of the insertion points in all files, so you can immediately continue working where you left off.

Three named workspaces, plus the last workspace used, are available for each project you create and for Visual Workbench's default state (that is, when no projects are loaded). You can load any of the named workspaces from the Project menu. Also, you can specify any of the named workspaces or the last workspace used to be initially loaded when you open a project.

## Using the Last Workspace Used

It is often useful to maintain your Visual Workbench setup between sessions. Whenever you exit Visual Workbench or close a project, the current workspace (Visual Workbench's configuration) is automatically saved. When a project is closed, either directly or by exiting Visual Workbench, the current workspace is saved in the project's workspace file. If no project is open when you exit Visual Workbench, the current workspace is saved in Visual Workbench's workspace file.

If you want the workspace you were last using to be automatically loaded the next time you open Visual Workbench, use the Workspace dialog box, accessed from the Options menu, to set the initial workspace to "Last Workspace Used" (see "Loading an Initial Workspace" on page 106). Then, since Visual Workbench always loads the most recently used project when it starts, the last workspace associated with that project is also loaded. If no project was active at the end of the last session, the last workspace used in Visual Workbench's default state is loaded.

## Saving a Workspace

You can save up to three workspaces for each project, besides the current workspace, which is automatically saved (and becomes the "Last Workspace Used"). You can also save up to three workspaces for Visual Workbench itself, before any project is assigned to it.

To save a workspace for a project, first open the project, then take the following steps. To save a workspace for Visual Workbench itself, save the workspace when no project is open.

► **To save a workspace:**

1. Arrange and size the appropriate windows.
2. From the Project menu, choose Save Workspace.
3. From the cascading menu, choose Edit, Debug, or Custom.

Edit, Debug, and Custom are default workspace names, and are simply useful labels. You can change the default names by choosing the Workspace command from the Options menu.

The workspace information is stored in a file with the project name and a .WSP extension.

## Loading a Workspace

When a project is open, you can only load the workspaces defined for that project. If no projects are open, you can load the workspaces defined for Visual Workbench.

► **To load a workspace:**

1. From the Project menu, choose Load Workspace.
2. From the cascading menu, choose Edit, Debug, or Custom.

Any files already open that do not belong in the current workspace are either minimized to an icon or are closed depending on the setting of the Close Non-Workspace Windows on Load check box in the Workspace dialog box.

## Loading an Initial Workspace

When you open a project, the project can automatically open a workspace. When you start Visual Workbench, it can also load an initial workspace.

Use the Workspace command on the Options menu to define an initial workspace. If a project is open when you use the command, you define an initial workspace for the project. If no project is open, you define an initial workspace for Visual Workbench.

► **To define an initial workspace to load:**

1. From the Options menu, choose Workspace.  
The Workspace dialog box appears (see Figure 8.2).
2. From the Initial Workspace box, select a workspace name.
3. Choose OK.

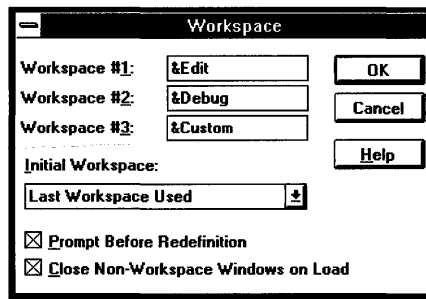


Figure 8.2 The Workspace Dialog Box

## Using External Projects

If you have existing makefiles that you want to use in place of Visual Workbench's project makefiles, you can run them from within Visual Workbench. These makefiles are called "external projects" because they are external to the normal mode of selecting project options from within Visual Workbench.

When you use an external makefile, you are responsible for the contents and actions of that makefile, since Visual Workbench does not read or alter the file. You can, however, build an external project and run any resulting program from within Visual Workbench. If your external makefile generates a CodeView-compatible debug version of the executable file, you can also debug the target from within Visual Workbench.

To use an external project, open it as you would any Visual Workbench project but select the Use as an External Makefile check box in the Open Project dialog box, accessed from the Open command on the Project menu.

► **To open an external project:**

1. From the Project menu, choose Open.  
The Open Project dialog box appears.
2. In the File Name box, type the name of the external makefile.
3. Select the Use as an External Makefile check box.

If you miss this step, Visual Workbench prompts you when it attempts to open the project as a Visual Workbench project and cannot. You can specify at that time that it is an external project.

4. Choose OK.

Once the external makefile is loaded, you can set build options by choosing the Project command from the Options menu. This opens the External Project Options dialog box (see Figure 8.3), where you can set build options.

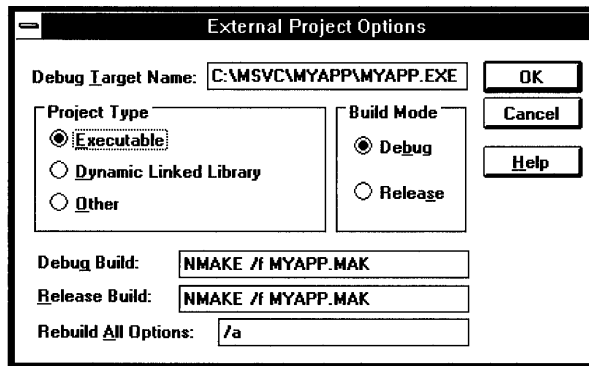


Figure 8.3 The External Project Options Dialog Box

Here are the build options you can set in the External Project Options dialog box:

#### Debug Target Name

Visual Workbench uses the name of the executable target file in this box to determine what file to use when you use the debugger. This name also appears on the Project menu following Build, Rebuild All, and Execute. Visual Workbench assumes the name of the executable file to have the same base name as the project. If you change the name in the Debug Target Name box, you are responsible for making sure your external makefile generates an executable file with this same name.

#### Project Type

This is the target file type (Executable, DLL, or other). Depending on the project type specified here, the Debug dialog box, accessed from the Options menu, presents slightly different options. If the project type is specified as Executable, the upper text box in the Debug dialog box is labeled “Program Arguments” and lets you enter any parameters to pass to the program. If the project type is specified here as DLL, the upper text box in the Debug dialog box is labeled “Calling Program” and lets you enter the executable filename that calls the DLL.

#### Debug Build

This is the program maintenance utility, such as NMAKE, along with any command-line options it requires to build a debug version of the project type. The command in this box is invoked when you choose Build *Targetname* from the Project menu or click the Build button on the toolbar if you have Debug selected under Build Mode in the External Project Options dialog box.

### Release Build

This is the program maintenance utility, such as NMAKE, along with any command-line options it requires to build a release version of the project type. The command in this box is invoked when you choose Build *Targetname* from the Project menu or click the Build button on the toolbar if you have Release selected under Build Mode in the External Project Options dialog box.

### Rebuild All Options

This is the command-line option required by the program maintenance utility to generate a complete rebuild. The command line specified in either the Debug Build or Release Build text box (depending on which mode you are building) and the command-line option in this text box are combined and invoked when you choose Rebuild All *Targetname* from the Project menu or click the Rebuild All button on the toolbar.

Importing Microsoft Programmer's Workbench (PWB) makefiles into Visual Workbench is straightforward since PWB makefiles use the default NMAKE command line in the Debug Build, Release Build, and Rebuild All Options text boxes. If you want the ability to build both Release and Debug versions, you can use the same makefile for both and add `DEBUG=1` as a command-line option in the Debug Build text box and `DEBUG=0` as a command-line option in the Release Build text box.

To build an external project, click the Build button on the toolbar or choose Build *Targetname* from the Project menu. To rebuild an external project, click the Rebuild All button on the toolbar or choose Rebuild All *Targetname* from the Project menu.

To run an external project target file, choose the Execute *Targetname* command from the Project menu. Or to debug an external project target file that contains debug information, choose Go from the Debug menu or click the Run button on the toolbar, or use any of the trace commands, Step Into, Step Over, or Step to Cursor, from the Debug menu.



# Customizing Build Options

When you choose a project type, Visual Workbench sets up the compiler and linker options necessary to build that project type. You can customize these options using dialog boxes accessed from the Project Options dialog box, which contains a Customize Build Options group with three buttons:

- Compiler
- Linker
- Resources

A typical use of customization, especially for Visual C++ Professional Edition users, is to use the C/C++ Compiler Options dialog box to optimize your program for size or speed after all development and debugging have been done. Another use of these options might be to add more Windows import libraries to your link options from the Linker Options dialog box. Or you may want to add a custom resource compiler option using the Resource Compiler Options dialog box.

This chapter describes how to set options in each of these dialog boxes. It also describes how to use the specialized Help system employed by the C/C++ Compiler Options and Linker Options dialog boxes.

## The Compiler and Linker Options Dialog Boxes

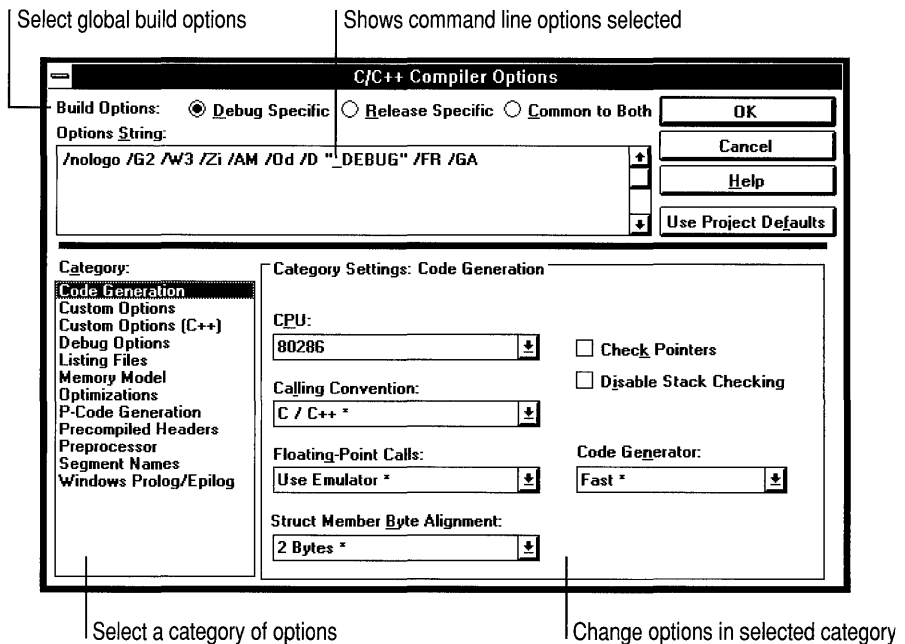
The C/C++ Compiler Options dialog box and the Linker Options dialog box are similar in operation (see Figure 9.1). Each is divided into two sections. The upper section displays the options string, which is a representation of the options given to the compiler and linker when you perform a build. There are also three Build Options buttons:

- Debug Specific
- Release Specific
- Common to Both



Choose the Debug Specific or Release Specific button before selecting options to have your options apply to a Debug or Release build, respectively. You can choose Common to Both to have the options you select apply to both Debug and Release builds.

**Note** The options string is a representation and not the actual command string. In most cases, the options string provides an exact replica of the command-line string. There are two exceptions to this. First, in precompiled headers for mixed languages, where a single option string is not passed to all modules, the options string indicates /Yu (followed by a header filename) for each language. Second, a special mnemonic that is not a LINK command-line option (/LIB) is used to indicate which libraries are passed to the linker.



**Figure 9.1** The C/C++ Compiler Options Dialog Box

The lower section of the C/C++ Compiler Options and Linker Options dialog boxes is divided in two. On the left, a list box contains categories of options. When you select a category, a new Category Settings group appears on the lower right of the dialog box to allow you to set options in that category.

Within the Category Settings group, you use various controls (list boxes, text boxes, check boxes, and option buttons) to select compiler or linker options, which are then reflected by their corresponding command-line mnemonics in the options string at the top of the dialog box.

When the control is a list box, a “default” is indicated by an asterisk. The default is the option that the compiler or linker provides when no command-line argument is provided. When you select a default option, the mnemonic for that option does not appear in the options string.

---

**Note** In list boxes in the C/C++ Compiler Options and Linker Options dialog boxes, the item with the asterisk indicates the compiler or linker default operation in the absence of a command-line argument for the option. It does *not* indicate the default option for your selected project type. For example, the default compiler option for memory model—that is, when no command-line option is present—is Small (/AS). However, most project types have a default of Medium (/AM), which is automatically added to the options string when you select the project type.

---

## Getting Help on Options

Because there are so many compiler and linker options available to you in the C/C++ Compiler Options and Linker Options dialog boxes, a specialized Help facility is provided to let you get quick definitions of options in the options string. Also the normal Visual Workbench Help facility for dialog boxes is enhanced to provide quick access to help on the check boxes, text boxes, and list boxes that set those options.

The Resource Compiler Options dialog box uses the same Help interface as standard Visual Workbench dialog boxes since resource compiler options are not as complex as compiler and linker options.

### The Help Command Button

As with any Visual Workbench dialog box, you can get a general overview of the compiler, linker, and resource compiler dialog boxes by choosing the Help command button.

### F1 Help for Compiler and Linker Options

Help for compiler and linker options is context sensitive when you use the F1 key. You can get either a short pop-up description of an option in the options string, or a complete description of an option, depending on the location of the insertion point when you press F1.

- ▶ **To get a short description of any option in the options string:**
  1. Highlight the option in the options string (double-click the option).  
Or just place the insertion point on the option.
  2. Press F1.

A pop-up window appears with a brief description of the compiler or linker option (see Figure 9.2).

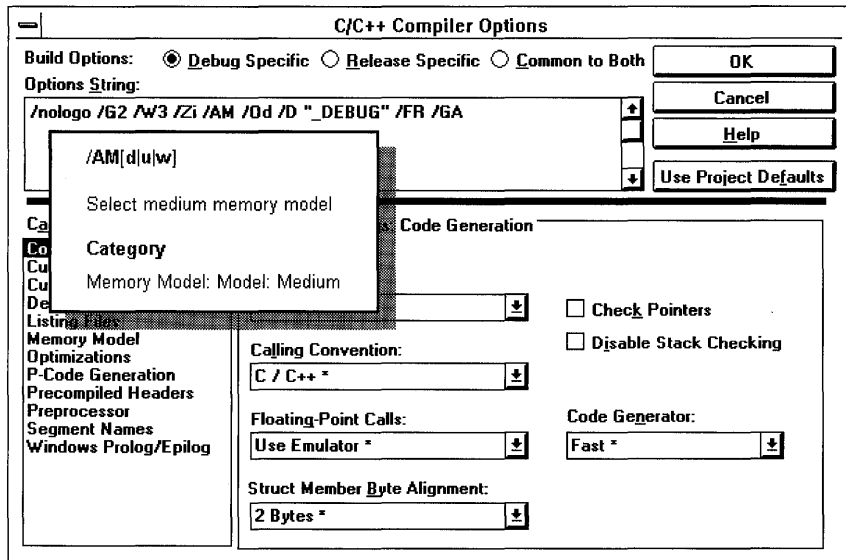


Figure 9.2 Help on the Options String

► **To get complete help on any option in the Category Settings group:**

1. Move the dialog focus to the option control.

For example, place the dialog focus in a list box, on a check box, or in a text box.

2. Press F1.

A Help window appears with definitions and descriptions of all the options associated with the control (see Figure 9.3). For example, if it is a list box, all list box entries are described.

Once a Help window is open, you can get help on any option, either by category or alphabetically, by going to the Contents screen. See page 71 for more information on using the Help system for compiler and linker options.

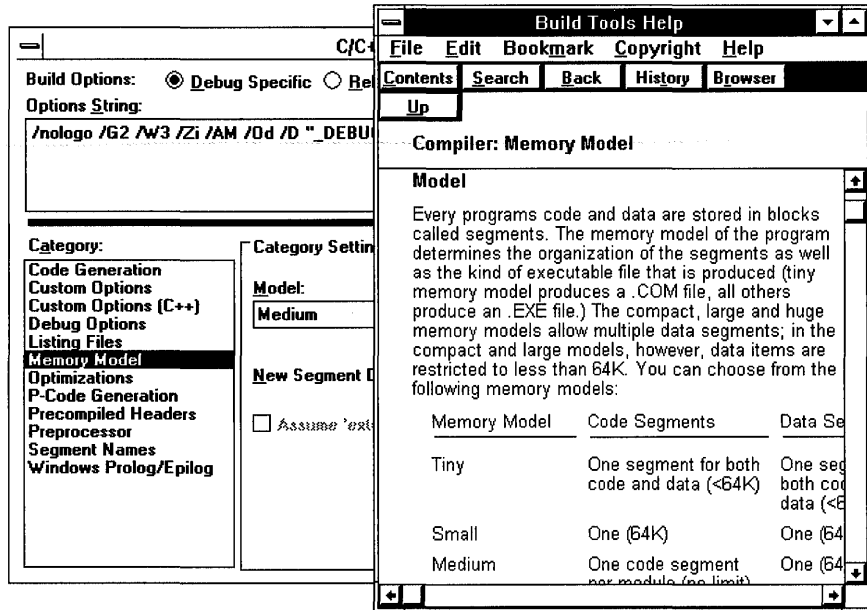


Figure 9.3 Help on Option Controls

## Default Compiler and Linker Options

When you choose a project type, Visual Workbench provides a set of default compiler and linker options for that project type for both debug and release builds. Tables 9.1 through 9.4 show these default compiler and linker options for each project type.

### ► To return all options to their project-specified default settings:

- Choose the Use Project Defaults button.

This sets all options in the current build group (debug or release) to the state originally set by Visual Workbench for the chosen project type.

Default compiler and linker options, as described in this chapter, are written to the MSVC.INI file at installation or whenever this file is re-created (as happens if it is accidentally deleted). Options stored in this file are used whenever a new project is created or the Use Project Defaults button is chosen.

If you are porting existing projects into Visual Workbench (that is, creating a new Visual C++ project that matches the existing project's build options), you need to be sure that the default build options set by the Visual Workbench project type do not conflict with the build options of your existing project. You can do this by comparing the build options in your existing makefile with the defaults in the

following tables specified for your chosen project type. For an alphabetic cross-reference of CL and LINK options to their respective controls in the C/C++ Compiler Options and Linker Options dialog boxes, see pages 52 and 55, respectively.

**Table 9.1 Default Compiler Options for Windows Project Types**

Project Type	Common to Both	Debug Specific	Release Specific
Windows application (.EXE)	/nologo /W3 /AM /FR /GA	/G2 /Zi /Od /D "_DEBUG"	(f-)* /O1 /D "NDEBUG"
Windows dynamic-link library (.DLL)	/nologo /W3 /AM /FR /GD	/G2 /Zi /Od /D "_DEBUG"	(f-)* /O1 /D "NDEBUG"
Visual Basic Custom Control (.VBX)	/nologo /G2 /Gc /Zp1 /W3 /AM /FR /GD	/Zi /Od /D "_DEBUG"	(f-)* /Gs /Ox /D "NDEBUG"
QuickWin application (.EXE)	/nologo /G2 /Mq /W3 /AM /FR	/Zi /Od /D "_DEBUG"	(f-)* /Gs /Ox /D "NDEBUG"
Static library (.LIB)	/nologo /G2 /W3 /AM /FR /GA	/Z7 /Od /D "_DEBUG"	(f-)* /Gs /Ox /D "NDEBUG"
Windows P-code application (.EXE)	/nologo /G2 /Oq /W3 /AM /FR /GA	/Zi /Od /D "_DEBUG"	/O1 /D "NDEBUG"

\* Visual C++ Professional Edition only

**Table 9.2 Default Compiler Options for MS-DOS Project Types**

Project Type	Common to Both	Debug Specific	Release Specific
MS-DOS application (.EXE)	/nologo /G2 /W3 /AM /D "_DOS" /FR	/Zi /Od /D "_DEBUG"	(f-)* /Gs /Ox /D "NDEBUG"
MS-DOS P-code application (.EXE)	/nologo /G2 /W3 /AM /Oq /D "_DOS" /FR	/Zi /Od /D "_DEBUG"	/Gs /Ox /D "NDEBUG"
MS-DOS Overlaid application (.EXE)	/nologo /G2 /Gy /W3 /AM /D "_DOS" /FR	/Zi /Od /D "_DEBUG"	(f-)* /Gs /Ox /D "NDEBUG"
MS-DOS COM application (.COM)	/nologo /G2 /W3 /AT /D "_DOS" /FR	/Zi /Od /D "_DEBUG"	(f-)* /Gs /Ox /D "NDEBUG"

\* Visual C++ Professional Edition only

**Table 9.3 Default Linker Options for Windows Project Types**

<b>Project Type</b>	<b>Common to Both</b>	<b>Debug Specific</b>	<b>Release Specific</b>
Windows application (.EXE)	/NOLOGO /NOD /STACK:5120 /ALIGN:16 /ONERROR:NOEXE /PACKC:61440	/CO	None
Windows dynamic-link library (.DLL)	/NOLOGO /NOD /ALIGN:16 /ONERROR:NOEXE /PACKC:61440	/CO /MAP:FULL	None
Visual Basic Custom Control (.VBX)	/NOLOGO /NOD /ALIGN:16 /ONERROR:NOEXE /PACKC:61440	/CO	None
QuickWin application (.EXE)	/NOLOGO /NOD /STACK:5120 /ALIGN:16 /ONERROR:NOEXE /PACKC:61440	/CO	None
Windows P-code application (.EXE)	/NOLOGO /NOD /ONERROR:NOEXE /PCODE /PACKC:61440	/CO	None

**Table 9.4 Default Linker Options for MS-DOS Project Types**

<b>Project Type</b>	<b>Common to Both</b>	<b>Debug Specific</b>	<b>Release Specific</b>
MS-DOS application (.EXE)	/NOLOGO /NOD /NOI /STACK:5120 /ONERROR:NOEXE	/CO	None
MS-DOS P-code application (.EXE)	/NOLOGO /NOD /NOI /PCODE /ONERROR:NOEXE	/CO	None
MS-DOS Overlaid application (.EXE)	/NOLOGO /NOD /NOI /ONERROR:NOEXE	/CO	None
MS-DOS COM application (.COM)	/NOLOGO /NOD /NOI /TINY	/CO	None

## Compiler Options

To set compiler options, choose Compiler in the Project Options dialog box Customize Build Options group. Each option category is described below, with a short description of each individual option that can be changed in that category's Category Settings group.

### Code Generation

The Code Generation category (see Figure 9.4) includes options for compiling for different CPU types, for using the Pascal or C/C++ calling convention, for selecting how floating-point numbers are handled, and for setting alignment boundaries for structures. It also lets you turn on and off options for checking for out-of-range pointers and, if you have Visual C++ Professional Edition, lets you turn on and off fast compiling. The stack-checking option is also enabled or disabled here.

Category Settings: Code Generation

CPU:  
80286

Calling Convention:  
C / C++ \*

Floating-Point Calls:  
Use Emulator \*

Struct Member Byte Alignment:  
1 Byte

Code Generator:  
Fast \*

Check Pointers  
 Disable Stack Checking

Figure 9.4 Compiler Options: Code Generation

### CPU

If you are writing programs for a computer with an 8086/8088, 80186/80188, 80286/80288, or 80386/80388 processor, you can use one of the CPU options. All options produce 16-bit instructions. Programs compiled for higher-numbered processors do not run on lower-numbered processors.

The /G3 option is only available with Visual C++ Professional Edition and requires the optimizing compiler (/f-). It generates smaller and faster code than the /G2 option. The resulting programs run only on computers with 80386 or higher

processors. /G3 implies /G2 and thus generates both the `_M_I286` and the `M_I286` preprocessor identifiers. The /G3 option does not support 80386/80387 inline-assembler instructions.

### CPU List Box

List Entry	Option	Comment
8086/8088 *	/G0	Generate 8086 instructions
80186/80188	/G1	Generate 80816 instructions
80286/80288	/G2	Generate 80286 instructions
80386/80388	/G3	Generate 80386 instructions (optimizing compiler)

\* This is a default CL option and does not appear in the options string when selected.

## Calling Convention

The calling-convention options determine which way arguments passed to functions are pushed on the stack, and whether the calling or called function removes the arguments from the stack. C functions can have a variable number of arguments, and in the C/C++ calling convention, arguments are pushed on the stack from right to left (so that the first argument in the list is the last one pushed on the stack). Pascal and FORTRAN programs use the Pascal calling convention, where the number of arguments is fixed and arguments are pushed on the stack from left to right.

Also in C, the calling function must remove the arguments, whereas in Pascal and FORTRAN, the called function does this. The Pascal calling convention optimizes size somewhat, while the C/C++ calling convention allows the flexibility of using variable-number parameter lists.

### Calling Convention List Box

List Entry	Option	Comment
Pascal	/Gc	Use Pascal calling convention
C/C++ *	/Gd	Use C/C++ calling convention

\* This is a default CL option and does not appear in the options string when selected.

## Floating-Point Calls

The floating-point calls options specify how your program handles floating-point-math operations. The options /Fpa, /FPc87, and /FPc are only available with Visual C++ Professional Edition and require the optimizing compiler (/f-). For complete information on using floating-point calls, see Chapter 1, “CL Command Reference,” in the *Command-Line Utilities User’s Guide*.



### Floating-Point Calls List Box

List Entry	Option	Comment
Inline 80x87 Instructions	/FPi87	Use Inline 80x87 instructions
Use Emulator *	/FPi	Use 80x87 inline emulation
Alternate Math	/FPa	Use fast alternate math (optimizing compiler)
80x87 Calls	/FPc87	Use 80x87 calls (optimizing compiler)
Coprocessor Calls	/FPc	Use 80x87 emulation calls (optimizing compiler)

\* This is a default CL option and does not appear in the options string when selected.

## Struct Member Byte Alignment

The /Zp option controls how the members of a structure are packed into memory and specifies the same packing for all structures in a module. When you specify the /Zpn option, where *n* is 1, 2, 4, 8, or 16, each structure member after the first is stored on *n*-byte boundaries.

### Struct Member Byte Alignment List Box

List Entry	Option	Comment
1 Byte	/Zp1 (/Zp)	Pack structures on 1-byte boundaries
2 Bytes *	/Zp2	Pack structures on 2-byte boundaries
4 Bytes	/Zp4	Pack structures on 4-byte boundaries
8 Bytes	/Zp8	Pack structures on 8-byte boundaries
16 Bytes	/Zp16	Pack structures on 16-byte boundaries

\* This is a default CL option and does not appear in the options string when selected.

## Code Generator

The Code Generator list box lets you select whether to use the fast compiler or the optimizing compiler or whether to let the compiler determine which to use based on other options. For Visual C++ Standard Edition, the fast compiler option is automatically selected and cannot be changed. For Visual C++ Professional Edition, you can choose from two options in the list box: Fast and Optimizing.

The Fast option invokes the fast compiler (the CL default). The fast compiler results in faster compilations but can produce larger, slower programs. Fast compiler (/f) is useful during the development process. Although most optimizations are accepted by the fast compiler, many optimizations have different implementations in the fast compiler than in the optimizing compiler.

Selecting the Optimizing option (/f-) turns off the fast compile option and invokes the C/C++ optimizing compiler.

**Code Generator List Box**

List Entry	Option	Comment
Fast *	/f	Faster compilation but larger, slower program
Optimizing	/f-	Slower compilation but smaller, faster program

\* This is a default CL option and does not appear in the options string when selected.

**Code Generation Check-Box Summary**

The check boxes in the Code Generation category are listed together here for summary information. Each specific option is described in the sections that follow.

**Code Generation Check Boxes**

Check Box	Option	Comment
Check Pointers	/Zr	Check null pointers (fast compile only)
Disable Stack Checking	/Gs	Remove stack-check calls

**Check Pointers**

The /Zr option checks for null or out-of-range pointers, which cause run-time errors in your program. The /Zr option is only available with the fast compile option.

**Disable Stack Checking**

Stack checking is a means by which the compiler inserts “stack probe” routines that are called on entry to each function to verify that the program stack has enough room to allocate the local variables required by the function. When stack checking is turned off, a stack overflow can occur without being diagnosed (no stack-overflow message is printed). Disabling stack checking can make your programs smaller and faster, but at the expense of diagnostic capability.

**Custom Options**

Several miscellaneous options are controlled in the Custom Options category (see Figure 9.5). These include options for turning off Microsoft language extensions, enabling QuickWin (a default of the QuickWin project type), enabling function-level linking and string pooling, and selecting the warning level. The Other Options text box is included here to allow you access to CL options that do not have corresponding controls in the C/C++ Compiler Options dialog box.

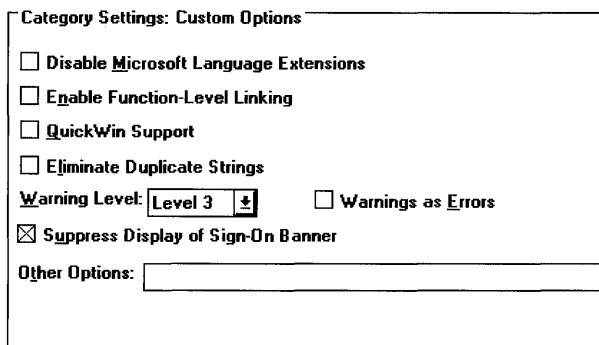


Figure 9.5 Compiler Options: Custom Options

## Custom Options Check-Box Summary

The check boxes in the Custom Options category are listed together here for summary information. Each specific option is described in the sections that follow.

### Custom Options Check Boxes

Check Box	Option	Comment
Disable Microsoft Language Extensions	/Za	Disables the default (/Ze)
Enable Function-Level Linking	/Gy	Package functions
QuickWin Support	/Mq	Windows interface for MS-DOS programs
Eliminate Duplicate Strings	/Gf	Implement string pooling
Warnings as Errors	/WX	Treat all warnings as errors
Suppress Display of Sign-On Banner	/nologo	Banner not sent to Output window

## Disable Microsoft Language Extensions

Visual C++ supports the ANSI C standard. In addition, it offers a number of features beyond those specified in the ANSI standard. These features are enabled by the /Ze option (the default) and disabled by the /Za option. For a complete list of these extensions, see Help or Chapter 1, “CL Command Reference,” in the *Command-Line Utilities User's Guide*.

## Enable Function-Level Linking

The /Gy option enables linking on a function-by-function basis by creating packaged functions. Packaged functions are used in conjunction with the **FUNCTIONS** statement in a module-definition file to order functions in an executable file, or to

assign functions to a segment or an overlay. You can also exclude unreferenced packaged functions from an executable file using the linker's `/PACKF` option. C++ member functions are automatically packaged.

## QuickWin Support

MS-DOS programs compiled with the `/Mq` compiler option have a limited Windows user interface, including a standard menu bar, standard Help (for QuickWin features), and a client (or application) window with a child (document) window for input and output streams (`stdin`, `stdout`, and `stderr` for C). The `/Mq` compiler option defines the `_WINDOWS` constant, declared in the Windows version of `STDIO.H`. For information on writing QuickWin programs, see Chapter 7 of *Programming Techniques*.

## Eliminate Duplicate Strings

The `/Gf` option enables the compiler to place a single copy of identical strings into the executable file. Because identical strings are copied into a single memory location, programs compiled with the `/Gf` option can be smaller than those compiled without `/Gf`. This space optimization is sometimes called “string pooling.” Use of `/Gf` does not guarantee string pooling in all cases. String pooling does, however, occur in most cases.

## Warnings as Errors

The `/WX` option instructs the compiler to consider any warning message it produces as an error. If there are any warning messages, an error message is emitted and compilation continues.

## Suppress Display of Sign-On Banner

The `/nologo` option suppresses the copyright message displayed when `CL` is invoked. This speeds the compilation slightly since the Output window is not updated with this information.

## Warning Level

You can control the number of warning messages produced by the compiler by setting the warning level option. Warning level `/W1` displays only severe warnings. `/W2` displays intermediate warnings, such as the use of functions with no declared return type, failure to put `return` statements in functions that aren't void, and data conversions that would cause loss of data or precision. `/W3` displays less severe warnings, such as warnings about function calls that precede their function prototypes. `/W4` displays warnings such as non-ANSI features and extended keywords.

### Warning Level List Box

List Entry	Option	Comment
None	/W0	Display no warnings
Level 1 *	/W1	Display only the most severe warnings
Level 2	/W2	Display intermediate level of warnings
Level 3	/W3	Display most warnings
Level 4	/W4	Display all warnings

\* This is a default CL option and does not appear in the options string when selected.

## Other Options

The Other Options text box lets you type in CL options independent of the dialog-box controls used by Visual Workbench to set them. This can be useful if you want to use a specific set of command-line options from some other makefile and you don't want to set these options from the dialog boxes. For a complete description of all CL options, see Chapter 1, "CL Command Reference," in the *Command-Line Utilities User's Guide* (provided with Visual C++ Professional Edition).

**Note** You are responsible for the accuracy of any option you enter in this text box. If Visual Workbench recognizes the option as one that can be set using a dialog-box control, it changes the dialog-box control to reflect the option and removes the option from the Other Options box. However, if the option is not recognized, it is left on the options string as is and passed to the compiler.

## Custom Options (C++)

The Custom Options (C++) category (see Figure 9.6) lets you specify how C++ pointers to class members are represented.

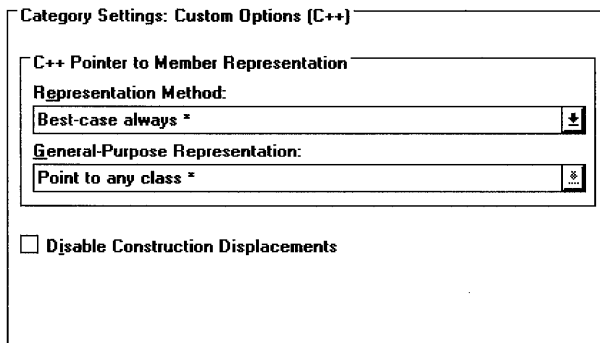


Figure 9.6 Compiler Options: Custom Options (C++)

## C++ Pointer to Member Representation

Visual C++ supports pointers to members of any class. The number of bytes required to represent a pointer to a member of a class and the code required to interpret the representation varies considerably, depending upon whether the class is defined with no, single, multiple, or virtual inheritance (no inheritance being smallest and virtual inheritance largest).

If you always declare a pointer to a member of a class after defining the class, you can use the default option (/vmb), which causes the compiler to generate an error when a pointer to a member of a class is declared before the class. This lets the compiler create the best-case representation, since it knows the inheritance model of the class before encountering a pointer to a class member.

If you need to declare a pointer to a member of a class prior to defining the class, you must select the general representation model (/vmg) and then specify the inheritance-model that is to be assumed for both the pointer representation and the code required to interpret the pointer representation.

## Representation Method

Use the /vmb option if you always define a class before you declare a pointer to a member of the class.

When the compiler encounters the declaration of a pointer to a member of a class, it already has knowledge of the kind of inheritance used by the class. Thus, the compiler can use the smallest possible representation of a pointer and generate the smallest amount of code required to operate on the pointer for each kind of inheritance.

Use the /vmg option if you need to declare a pointer to a member of a class before defining the class. This need can arise if you define members in two different classes that reference each other. For such mutually referencing classes, one class must be referenced before it is defined. You must then select an inheritance model from the General-Purpose Representation list box.

### Representation Method List Box

List Entry	Option	Comment
Best-case always *	/vmb	Always declare class before declaring pointers to class members
General-purpose always	/vmg	Can declare pointers to class members before defining class

\* This is a default CL option and does not appear in the options string when selected.

## General-Purpose Representation

When the representation method is General-purpose always (/vmg), you must also specify an option to indicate the inheritance model of the not-yet-encountered class definition. This can be one of three options: /vms (single inheritance), /vmm (multiple inheritance), or /vmv (virtual inheritance). Each of these options specifies the most general representation of a pointer to a member of a class—that is, it allows the pointer to point to members of classes with all inheritance models of its specified type or more restrictive types. Each makes a trade-off between flexibility and code size.

When you specify an inheritance model, that model is used for all pointers to member classes, regardless of their inheritance type or whether or not the pointer is declared before or after the class. Therefore, if you know that you always use single-inheritance classes, you can reduce code size by selecting that inheritance model; however, if you want to be safest (at the expense of the largest data representation), you can choose the virtual-inheritance model, which allows pointers to all classes.

The /vms option specifies that all pointers to not-yet-declared class members will only point to a member of a class that uses either no inheritance or single inheritance. This requires the smallest pointer size and least amount of code to interpret the pointer size; however, it causes the compiler to generate an error if the class member pointed to turns out to be from a class with a multiple or virtual inheritance model.

The /vmm option allows the pointer to point to members of classes with no inheritance, single inheritance, or multiple inheritance, but produces a larger code than the /vms option. It causes the compiler to generate an error if the pointer points to a virtual class member.

The /vmv option specifies the most general representation of a pointer to a member of a class to be one that uses virtual inheritance. In terms of pointer size and the code required to interpret the pointer, this is the most expensive option. This option, however, never causes an error and is the default.

### General-Purpose Representation List Box

List Entry	Option	Comment
Point to any class *	/vmv	Declare virtual inheritance
Point to single inheritance classes	/vms	Declare single inheritance
Point to multiple inheritance classes	/vmm	Declare multiple inheritance

\* This is a default CL option and does not appear in the options string when selected.

## Disable Construction Displacements

Microsoft Visual C++ implements C++ construction displacement support. Construction displacements solve the problem created when a virtual function, declared in a virtual base and overridden in a derived class, is called from a constructor during construction of a further derived class. The problem is that the virtual function may be passed an incorrect **this** pointer. This is caused by discrepancies between the displacements to virtual bases of a class and the displacements to its derived classes. The solution provides a single construction displacement adjustment, called a vtordisp field, for each virtual base of a class.

By default, vtordisp fields are introduced whenever the code both defines user-defined constructors and destructors and also overrides virtual functions of virtual bases. The Disable Construction Displacements option (/vd0) lets you disable the construction displacements generation to help optimize program size. Select Disable Construction Displacements only if you are certain that all class constructors and destructors call virtual functions virtually.

### Disable Construction Displacements Check Box

Option	Comment
/vd0	May reduce program size

## Debug Options

The Debug Options category (see Figure 9.7) lets you select parameters used when compiling a file for use with the Visual Workbench or CodeView debugger. The /Zi option has changed from previous versions of the compiler to include all type information in a program database.

Support for the older style debug information storage is provided with the new option /Z7 (which emulates the behavior of /Zi in the Microsoft C/C++ version 7.0 compiler). The /Zd option is an abbreviated form of the /Z7 option.

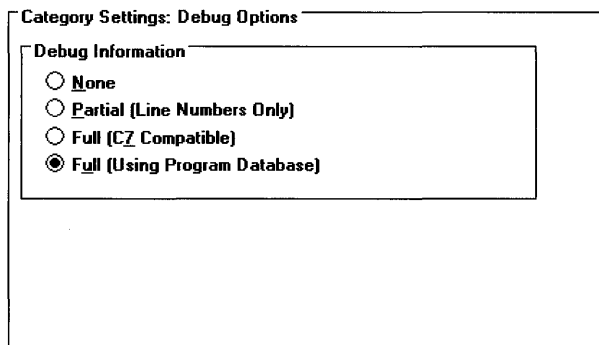


Figure 9.7 Compiler Options: Debug Options



## Debug Options Option-Button Summary

The option buttons in the Debug Options category are listed together here for summary information. Each specific option is described in the sections that follow.

### Debug Options Option Buttons

Option Button	Option	Comment
None	none	No debug information is created
Partial (Line Numbers Only)	<code>/Zd</code>	Only public symbols and line numbers
Full (C7 Compatible)	<code>/Z7</code>	Put all debug information in object files
Full, Use Program Database	<code>/Zi</code>	Enabled by the <code>/Od</code> (Disable (Debug)) option button in the Optimizations category

### None

Choose None to disable the creation of symbolic debugging information. This is the default for release mode builds.

### Partial (Line Numbers Only)

The `/Zd` option produces an object file containing only global and external symbol information and line number information. This reduces the size of the debuggable executable file. You can use `/Zd` if you do not use the expression evaluator during debugging. `/Zd` uses the C7-compatible method of information storage.

### Full (C7 Compatible)

The `/Z7` option produces an executable file containing line numbers and full symbolic-debugging information for use with the Visual Workbench debugger or with CodeView. This symbolic information is a map of your source code and includes such things as the names and types of variables and functions, as well as the names and numbers of all program segments. The executable file also includes full symbol-table information and line numbers. The `/Z7` option was called `/Zi` in previous versions of CL. `/Zi` now uses a database to maintain type information.

### Full, Use Program Database

Visual C++ provides an improved method of storing some of its debug information. When you use the `/Zi` option, all type information in your source files is now stored in a special database file called MSVC.PDB. In previous versions of the Microsoft C and C++ compilers, all debug type information was written into each object file.

Now each object file contains a reference to the type information found in the program database. The program database (.PDB) file is necessary to correctly run the Microsoft Debugging Information Compactor (CVPACK) on a file compiled with /Zi, which is done automatically when you build an application in debug mode. If for some reason the .PDB file is deleted, you must rebuild all source files with /Zi to re-create it.

## Listing Files

Use the Listing Files category (see Figure 9.8) to generate output files for assembly listing files and source browser files.

**Category Settings: Listing Files**

- Browser Information**
  - Include Local Variables
  - Don't Pack Information
- Assembly**
  - include Machine Code
  - include Source and Machine Code

Figure 9.8 Compiler Options: Listing Files

## Browser Information

Source browser files (.SBR) contain symbolic information used by the Microsoft Browser Database Maintenance Utility (BSCMAKE) to produce a browser database file (.BSC) that is used by the Visual Workbench browser. You can force the compiler to generate .SBR files with or without symbolic information on local variables. You can also force the compiler to skip the step of compacting the .SBR file by removing unreferenced definitions.

### Browser Information Check Boxes

Check Box	Option	Comment
Browser Information	/Fr	Generate .SBR files (ignore local variables)
Include Local Variables	/FR	Generate .SBR files
Don't Pack Information	/Zn	Generate .SBR files but don't compact them

## Assembly

Use the Assembly check boxes to generate files containing various combinations of source code, assembly code, and machine code. Each of the options (/Fa, /Fl, and /Fc) produces a file with the base name of the source file. /Fa produces a file with the extension .ASM, and the /Fl and /Fc options produce files with the extension .COD.

### Assembly Check Boxes

Check Box	Option	Comment
Assembly	/Fa	Default assembly-code listing
Include Machine Code	/Fl	Generate combined assembly- and machine-code listing
Include Source and Machine Code	/Fc	Generate combined source-, assembly-, and machine-code listing

## Memory Model

Options in the Memory Model category (see Figure 9.9) determine the memory model used by the compiler and the allocation of stack segments and data segments for functions.

**Category Settings: Memory Model**

<b>Model:</b>	<b>Segment Setup:</b>
Medium <span style="float: right;">⌵</span>	SS == DS * <span style="float: right;">⌵</span>

**New Segment Data Size Threshold:**

Assume 'extern' and Uninitialized Data 'as'

Figure 9.9 Compiler Options: Memory Model

## Model

Every program's code and data are stored in blocks called "segments." The memory model of the program determines the organization of the segments as well as the kind of executable file that is produced (tiny memory model produces a .COM file, all others produce an .EXE file). The compact, large, and huge memory models allow multiple data segments; in the compact and large models, however, data items are restricted to less than 64K.

**Model List Box**

List Entry	Option	Comment
Tiny	/AT	One data segment for both code and data
Small *	/AS	One data segment; one code segment
Medium	/AM	One data segment; one code segment per module
Compact	/AC	Multiple data segments; one code segment
Large	/AL	Multiple data segments; one code segment per module
Huge	/AH	Multiple data segments (arrays can be > 64K) ; one code segment per module

\* This is a default CL option and does not appear in the options string when selected.

**Segment Setup**

The segment setup options are added as extensions to the existing memory-model option to modify the stack segment. For example, small memory model with SS==DS is coded as /ASd. The primary use of modified segment setups is for DLLs and Windows callback functions.

**Segment Setup List Box**

List Entry	Option	Comment
SS==DS *	d	Stack segment equals data segment
SS!=DS, DS loaded on function entry	u	Stack segment does not equal the data segment; the data segment is loaded for each function entry
SS!=DS, DS NOT loaded on function entry	w	Stack segment does not equal the data segment; the data segment is not loaded at function entry

\* This is a default CL option and does not appear in the options string when selected.

**New Segment Data Size Threshold**

The /Gt option causes all data items (other than constant data) either assumed to be far (uninitialized or marked **extern**), or of a determined size to be allocated in a new data segment. Type in a value (*n*) to tell the compiler to allocate all items whose size is greater than or equal to *n* in a new data segment. This option requires a memory model that allows multiple data segments (compact, large, or huge).

**New Segment Data Size Threshold Text Box**

Entry	Option	Comment
<i>n</i>	/Gtn	Range of <i>n</i> = 0 to 65534

## Assume 'extern' and Uninitialized Data 'far'

Under the compact, large, or huge memory model, the 16-bit compiler allocates all initialized and uninitialized data and data marked as **extern** as near if the data items are smaller than or equal in size to the threshold value set by the /Gt option. This is important for Windows-based applications since you can only achieve multiple instances of an application when all data is near. It is also creates more efficient code. The /Gx- option turns off this default behavior.

With the /Gx- option, the 16-bit compiler makes no assumptions about where the linker places uninitialized or external data. All references to those data items are done with far addressing, in case they are placed in a far segment.

### Assume 'extern' and Uninitialized Data 'far' Check Box

Option	Comment
/Gx-	Use with /AC, /AL, or /AH to keep all data far (/Gx is default)

## Optimizations

The Optimizations category (see Figure 9.10) lets you determine how the compiler will tune the performance of your program. Four of the five option buttons (Default, Disable (Debug), Maximize Speed, and Minimize Size) at the left of the Category Settings section require no further optimization on your part. If you select the Customize option button, you can set specific customizations using other option controls in this dialog box.

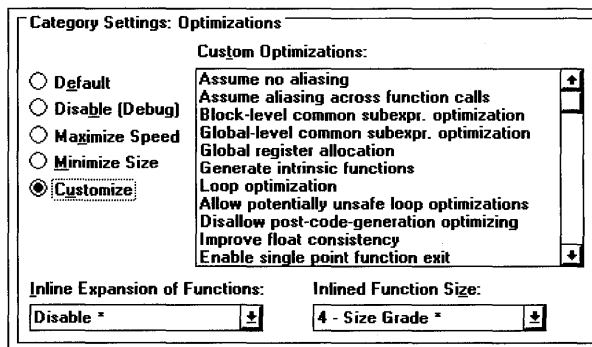


Figure 9.10 Compiler Options: Optimizations

### Optimizations Option Buttons

Option Button	Option(s)	Comment
Default	none	This is the same as Disable (Debug) without /Od
Disable (Debug)	/Od	Turn off all optimization and generate debug information
Maximize Speed	/O2	Apply optimizations for fastest program
Minimize Size	/O1	Apply optimizations for smallest program
Customize	(see Custom Optimizations)	Select options using the Custom Optimizations list box

## Disable (Debug)

Select the Disable (Debug) option button to create an executable file that contains debug information. This sets the option /Od, which enables debugging if any of the debugging options /Zi, /Z7, or /Zd are set (see the Debug Options category on page 127). /Od turns off optimization, which speeds compilation and also makes the program easier to debug by suppressing code reorganization. /Zi, /Z7, and /Zd create the debug information used by both the Visual Workbench debugger and CodeView.

## Maximize Speed

Select the Maximize Speed option button (/O2) to generate the fastest possible program. Its effect is the same as using the following options on the same command line:

```
/f- /Ow /Ox /Ob2 /OV4 /G2 /Gs /Gf /Gy
```

Visual C++ Professional Edition recognizes and uses all these options. Visual C++ Standard Edition ignores /f- and /Ow, and has a limited /Ox functionality.

Further increases in speed can be realized with other optimizations, such as the use of /G3 (use 80386 instructions).

## Minimize Size

Select the Minimize Size option button (/O1) to generate the smallest possible program. Its effect is the same as using the following options on the same command line:

```
/f- /Ow /Oe /Os /Ob1 /OV4 /G2 /Gs /Gf /Gy
```

Visual C++ Professional Edition recognizes and uses all these options. Visual C++ Standard Edition ignores /f-, /Ow, and /Os.

Further reduction in size can be realized with other optimizations, such as the use of p-code options and pragmas.

## Customize

Select the Customize option button to set Custom Optimizations options. You then select optimizations using the Custom Optimizations multiple-selection list box.

## Custom Optimizations

Options in this list box are enabled when the Maximize Speed, Minimize Size, or Customize option button is selected. Visual C++ Professional Edition uses all the options in this list box. Visual C++ Standard Edition uses only /Oe, /Ol, /Op, /Ox, and /Ot.

For detailed information on each of these optimizations, see Help or Chapter 1, "CL Command Reference," in the *Command-Line Utilities User's Guide*.

### Custom Optimizations List Box

List Box Entry	Option	Comment
Assume no aliasing	/Oa	Enable use of registers
Assume aliasing across function calls	/Ow	Reload variables after function call
Global-level common subexpr. optimization	/Og	Search functions for common subexpressions
Global register allocation	/Oe	Allocate registers according to frequency of variable use
Generate intrinsic functions	/Oi	Replace intrinsic functions with their inline code
Loop optimization	/Ol	Perform loop optimizations
Allow potentially unsafe loop optimizations	/Oz	Optimize loops aggressively
Improve float consistency	/Op	Use coprocessor registers to improve speed and size
Enable single point function exit	/Or	Use for debugging functions with several returns
Favor small code	/Os	Optimize for space
Favor fast code	/Ot	Optimize for time
Full optimization	/Ox	Maximize for speed. Same as: /Ob1 /Oc /Oe /Og /Oi /Ol /On /Oo /Ot /Gs (Std. Ed. ignores /Og /Oi /On)

## Inline Expansion of Functions

Inline expansion options control which functions become expanded. Expanding a function inline makes the program faster because it does not incur the overhead of calling the function. When /Ob1 is used, the compiler only expands functions marked as **inline** or **\_\_inline**, or C++ member functions defined within a class declaration. When /Ob2 is used, the compiler expands functions marked as **inline** or **\_\_inline**, as well as any other function that the compiler chooses.

### Inline Expansion of Functions List Box

List Entry	Option	Comment
Disable *	/Ob0	Disable inline expansion
Only __inline	/Ob1	Inline expansion is user-defined
Any suitable	/Ob2	Inline expansion at compiler discretion plus user-defined

\* This is a default CL option and does not appear in the options string when selected.

## Inline Function Size

If you choose “Any suitable” in the Inline Expansion of Functions list box (/Ob2), you let the compiler determine which functions to inline. You can then use the Inline Function Size list box to give the compiler general rules for determining which functions to inline, based on their size. You do this by choosing one of 10 size grades, from /OV0 to /OV9. For example, /OV limits inlining to only the smallest functions, whereas /OV9 inlines all functions up to a very large size. The size gradients are linear between the two extremes but do not map directly to specific precompiled source sizes.

### Inline Function Size List Box

List Entry	Option	Comment
Very Small	/OV0	Only inlines very small functions
1-Size Grade	/OV1	Size of function inlined relative to size grade
2-Size Grade	/OV2	Size of function inlined relative to size grade
3-Size Grade	/OV3	Size of function inlined relative to size grade
4-Size Grade *	/OV4	Size of function inlined relative to size grade
5-Size Grade	/OV5	Size of function inlined relative to size grade
6-Size Grade	/OV6	Size of function inlined relative to size grade
7-Size Grade	/OV7	Size of function inlined relative to size grade
8-Size Grade	/OV8	Size of function inlined relative to size grade
Fairly Large	/OV9	Inlines fairly large functions

\* This is a default CL option and does not appear in the options string when selected.



## P-Code Generation

If you have Visual C++ Professional Edition, and you want to optimize for the smallest-possible code size, options in the P-Code Generation category (see Figure 9.11) let you produce an alternate form of code called “p-code.”

P-code produces much smaller programs than machine code, but your machine cannot execute them directly. Instead, they are executed by a small run-time interpreter incorporated in the executable file. Because of this, programs run slower than when compiled into machine code. For more information on using p-code, see Chapter 7, “Reducing Program Size with P-Code,” in the *Command-Line Utilities User's Guide*.

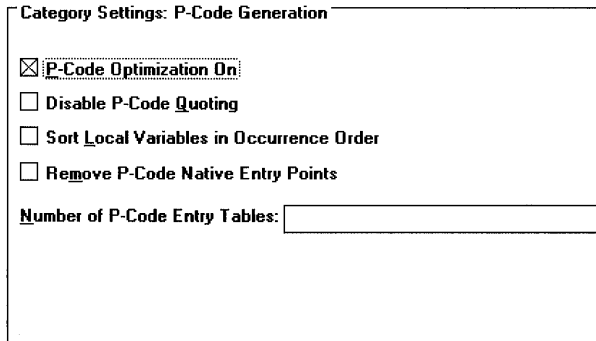


Figure 9.11 Compiler Options: P-Code Generation

### P-Code Generation Check-Box Summary

The check boxes in the P-Code Generation category are listed together here for summary information. Each specific option is described in the sections that follow.

#### P-Code Generation Check Boxes

Check Box	Option	Comment
P-Code Optimization On	/Oq	Turn on p-code optimization
Disable P-Code Quoting	/Of-	Quoting is on by default
Sort Local Variables in Occurrence Order	/Ov-	/Ov (default) sorts by frequency of use
Remove P-Code Native Entry Points	/Gn	Remove the native-code function entry point from each p-code function

## P-Code Optimization On

The `/Oq` option optimizes your code for size by compiling the program into p-code. You can also use the **optimize** pragma to turn p-code optimizing off and on in your source file, to exclude time-critical portions of code from p-code optimization, for example.

The `/Oq` option is not compatible with any of the following options: `/FPa`, `/FPc`, `/FPc87`, `/Fs`, `/Gr`, `/Sl`, `/Sp`, `/Ss`, or `/St`. Nor is it compatible with any of the optimization options.

## Disable P-Code Quoting

P-code quoting (`/Of`) enables the compiler to find duplicate sections of code and then create a single p-code function to implement them. This is the default setting for customization options but should be disabled (`/Of-`) for debugging, as it makes code difficult to read. Select this check box to disable p-code quoting.

## Sort Local Variables in Occurrence Order

The compiler reduces the size of p-code programs by using 1-byte opcodes to reference local variables. These opcodes are frame-relative addresses, and only a limited number are available for each function. The optimizer uses one of two algorithms to determine which variables receive the available opcodes:

- `/Ov` sorts the local variables by frequency of use (the default)
- `/Ov-` sorts the local variables in the order in which they occur

## Remove P-Code Native Entry Points

Native-code entry points are a short series of machine-code instructions placed at the beginning of a function compiled into p-code so that the function can be called by a machine-code function. You can save about 4 bytes per function by removing these, but you must make sure that no p-code functions are called by machine-code functions.

## Number of P-Code Entry Tables

Use the `/Gp` option to specify the maximum number of entry tables for your program. An entry table is needed for every segment that contains a p-code function or a function called by a p-code function. One entry table can describe up to 256 such

functions. If a segment contains more functions, the Make P-Code utility (MPC) creates additional entry tables. Use this option to cause MPC to generate an error if it requires more entry tables than the number specified by this option.

#### Number of P-Code Entry Tables Text Box

Entry	Option	Comment
<i>n</i>	<i>/Gpn</i>	Defaults to 255 if <i>/Gp</i> is not entered

## Precompiled Headers

Precompiled headers (PCH) are a means of greatly speeding compile time by compiling header files only once into a precompiled header file (.PCH) and thereafter using the precompiled header for each build. Although the Visual C++ compiler allows several ways to create and use precompiled header files, the Precompiled Headers category in the C/C++ Compiler Options dialog box (see Figure 9.12) presents two simplified approaches to implementing this feature.

You can use the new simplified method (*/YX*), which requires very little knowledge of precompiled headers, or you can use the more flexible method (*/Yc* and */Yu*), for more control and for mixed-language (C and C++) source files.

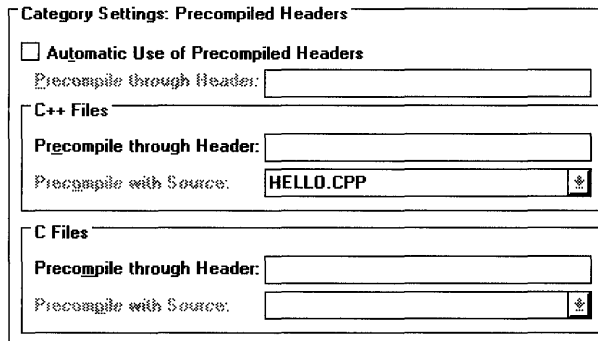


Figure 9.12 Compiler Options: Precompiled Headers

### Automatic Use of Precompiled Headers

The */YX* option instructs the compiler to use a precompiled header file with the default name MSVC.PCH if it exists, or to create one if it does not exist. The file is always created in the current directory (the */Fp* option, which specifies a precompiled header filename and directory, is not supported in Visual Workbench).

---

### Automatic Use of Precompiled Headers Check Box

---

Option	Comment
/YX	Automatically creates and uses the MSVC.PCH precompiled header file

---

Using /YX is the easiest way to implement precompiled headers. If your project contains only one source file, all you need to do is enable the Automatic Use of Precompiled Headers check box. When the source file is compiled, all preprocessor directives (such as **#include**, **#define**, or **#if**, for instance) from the beginning of the file up to where the C or C++ code starts are compiled into MSVC.PCH. Thereafter, whenever you build your project, the precompiled code from MSVC.PCH is used in place of the preprocessor directives, assuming you haven't altered the preprocessor directives in any way. Where this really benefits is in precompiling large header files, such as WINDOWS.H, that appear in **#include** statements.

If you have more than one source file in your project, you need to take a little care in order to use /YX effectively. There are basically three ways to use /YX with multiple files. In all cases, preprocessor directives must appear contiguously, in the same order, and at the beginning of each file. (The order within a group of **#define** statements is an exception and does not have to be identical.)

- Method 1: Place identical preprocessor code in every file. Make sure that the beginning of every source file in your project, up to where source code begins, contains identical preprocessor directives. This way, the first file compiled creates the MSVC.PCH file and all other source files use it. In subsequent builds, all source files use the MSVC.PCH file.
- Method 2: Place a subset of the preprocessor code at the beginning of every file. For example, if you have some headers used by every file, put the **#include** statements first and in the same order in each source file. The compiler then determines if the subset of contiguous preprocessor directives is large enough to warrant a precompiled header file. If so, it creates an MSVC.PCH file with common preprocessor elements. As an example, suppose that AFILE.C is compiled first and contains the following code at the start of the file:

```
#include <windows.h>  
#include "myapp.h"  
#include "afile.h"
```

BFILE.C is compiled next and contains:

```
#include <windows.h>  
#include "myapp.h"  
#include "bfile.h"
```

When it compiles `AFILE.C`, the compiler creates an `MSVC.PCH` file that contains `WINDOWS.H`, `MYAPP.H`, and `AFILE.H`. When it compiles `BFILE.C`, since the preprocessor directives don't match exactly, the compiler creates a new `MSVC.PCH` file; however, it includes only `WINDOWS.H` and `MYAPP.H` in it since these are common to both files. This has the advantage that both source files can now use the `MSVC.PCH` file for the majority of included code. The disadvantage is that `MSVC.PCH` had to be compiled twice.

- **Method 3:** Identify in advance which header files (and other preprocessor directives) you want precompiled by instructing the compiler where to stop precompiling, or using precompiled, preprocessor directives. To use this method, specify a header filename in the Precompile through Header text box that marks the last **#include** statement in the precompiled header data.

As an example, suppose you would be content if you could just precompile `WINDOWS.H`. To do so, simply place the following statement at the beginning of every source file in your project:

```
#include <windows.h>
```

Then select the Automatic Use of Precompiled Headers check box and type `windows.h` in the Precompile through Headers text box. `WINDOWS.H` is precompiled when the first source file is compiled and all source files thereafter use the precompiled code. This method can also be used by projects with only one source file to limit what goes into the precompiled header file.

Note that you can achieve the same results by inserting the directive **#pragma hdrstop** at a common point in all source files before which you want all preprocessor directives precompiled. For example, in the previous example, you could place the following code at the beginning of every source file:

```
#include <windows.h>  
#pragma hdrstop
```

When used with `/YX`, **#pragma hdrstop** does not require or use a filename argument and should be placed prior to source code in every file.

If you want to observe the behavior of the `/YX` option during the build, use the `/W4` option (Warning Level in the Custom Options category). Note that `/YX` is slower than `/Yu` when generating browser database information.

The compiler uses the following rules when comparing an existing `MSVC.PCH` file for consistency:

- The current compiler options must match those specified when the precompiled header was created.
- The current working directory must match that specified when the precompiled header was created.

- The order of all preprocessor directives, with the exception of **#define** directives, must match those specified when the precompiled header was created. The values of **#define** directives must match. The **#pragma** directives must be nearly identical—multiple spaces outside of strings are treated as a single space.
- The value and order of include paths specified using the `/I` option must match those used when the precompiled header was created.
- The timestamps of all the header files used to build the precompiled header must match those that existed when the precompiled header was created.

## C++ PCH and C PCH Information

The following rules apply for building precompiled header files from Visual Workbench using the `/Yc` and `/Yu` model:

- There can be one precompiled header (`.PCH`) file for each source language in the project (C and C++).
- For a given language, all files use the same precompiled header.
- Each source file in a given language must include the same header files, in the same order, up to the header file that you specify in the header text box.
- The include file must be specified in each source file with the same path.

To implement precompiled headers in modules of a specific language, type the name in the Precompile through Header text box (in the C++ Files or C Files group) of the last header file you want precompiled. Then, from the corresponding language's Precompile with Source list box, select a file that includes the header file in the Precompile through Header text box.

For example, assume you have a stub header file that is included by all of your source files and contains the following lines:

```
#include "first.h"  
#include "second.h"  
#include "last.h"
```

Also, assume one of your source files is called `MYPROG.CPP`. To include `FIRST.H`, `SECOND.H`, and `LAST.H` in the precompiled header file, type `LAST.H` in the

Precompile through Header text box in the C++ Files group. Then select MYPROG.CPP from the Precompile with Source list box in the C++ Files group.

---

**Note** You cannot use the `/Fp` command-line option from within Visual Workbench. Nor can you use the `#pragma hdrstop` declaration in any source file in your project when using `/Yc` and `/Yu`. You can, however, access these capabilities through an external makefile. See *Programming Techniques* and the *Command-Line Utilities User's Guide* for more information.

---

In the following tables, the Option column describes what you will see in the options string when you implement precompiled headers. If you are using only one language, you will see one `/Yu` option followed by the header filename. If you are using mixed-language precompiled headers, you will see one `/Yu` option for both header files specified. Note that the `/Yc` option does not appear in the options string.

Also note that the Option column does not indicate the actual CL command line that is generated. In the makefile, a separate `/Yu` option and `/Yc` option is generated for each language, on a per-module basis, accompanied by the respective header and source filenames you specify.

#### Precompile through Header: Text Boxes (for both C Files and C++ Files)

Entry	Option	Comment
<i>headerfilename</i>	<code>/Yu headerfilename</code>	Name of the last header file used in the precompiled header

#### Precompile with Source: List Boxes (for both C Files and C++ Files)

Entry	Option	Comment
<i>sourcefilename</i>	<code>/Yu sourcefilename</code>	<i>sourcefilename</i> is the name of the source file that includes the last header file used in the precompiled header

## Preprocessor

The Preprocessor category (see Figure 9.13) lets you control elements used by the C/C++ preprocessor, such as symbols, macros, and include paths. For a list of ANSI and Microsoft-specific predefined macros, see Chapter 7 in the *C Language Reference*, Chapter 13 in the *C++ Language Reference*, or Help.

**Category Settings: Preprocessor**

**Symbols and Macros to Define:**

**Individual Symbols to Undefine:**

**Undefine All Symbols**

**Include Path:**

**Ignore Standard Places of Include Files**

**Figure 9.13 Compiler Options: Preprocessor**

## Symbols and Macros to Define

Use the `/D` option to define constants and macros for your source file. This has the same effect as using a `#define` preprocessor directive at the beginning of your source file. You cannot use any identifier that contains spaces. You can assign a number or a string to the identifier using the equal sign (`=`) with no intervening spaces. If you omit the equal sign, the identifier is assumed to be defined (value of 1).

Type all the constants and macros you want to define separated by spaces or commas. Note that all project types define `_DEBUG` by default for debug build mode. This is used by the Microsoft Foundation Class Library and may be disregarded or removed if your project does not use the Microsoft Foundation classes.

### Symbols and Macros to Define Text Box

Entry	Option	Comment
<i>identifier</i>	<code>/D "identifier"</code>	Maximum of 30 identifiers

## Individual Symbols to Undefine

To turn off individual symbols that you have defined or that are predefined, type the names of all symbols separated by spaces, commas, or semicolons. A `/U` option is then created for each symbol name.

### Individual Symbols to Undefine Text Box

Entry	Option	Comment
<i>name</i>	<code>/U "name"</code>	Undefines existing symbols



## Undefine All Symbols

To turn off all symbols (both user-defined and predefined), select the Undefine All Symbols check box.

### Undefine All Symbols Check Box

Option	Comment
/u	Undefines all defined symbols

## Include Path

You can add to the list of directories searched for include files by typing the path in the Include Path text box. To indicate more than one path to search, separate the paths by spaces, commas, or semicolons. When an **#include** directive is encountered in a program (and a complete path is not provided), the compiler searches the current directory first, then the path or paths specified with the /I option, and finally the paths specified in the Directories dialog box, accessed from the Options menu.

### Include Path Text Box

Entry	Option	Comment
<i>directory</i>	/I " <i>directory</i> "	Searches directory for include files

## Ignore Standard Places of Include Files

You can prevent the compiler from searching the standard places for include files by using the /X (for "exclude") option. This excludes the current directory and any directories listed in the Directories dialog box, accessed from the Options menu. You can use this option with the /I option to define the location of include files with the same name (but using different code) as those in the standard places.

### Ignore Standard Places of Include Files Check Box

Option	Comment
/X	Ignore the current directory and the Visual Workbench include directories

## Segment Names

The compiler places code and data into separate segments in the object file. Every segment has a name that is used by the linker to determine which segments to combine and how segments are ultimately grouped in the executable file. The Segment Names category (see Figure 9.14) lets you set names for the data, code, p-code, module, and v-table segments.

Category Settings: Segment Names	
Data Segment:	<input type="text"/>
Code Segment:	<input type="text"/>
P-Code Segment:	<input type="text"/>
Module Name:	<input type="text"/>
V-Table Segment:	<input type="text"/>

Figure 9.14 Compiler Options: Segment Names

## Data Segment

The `/ND` option renames the default data segment of your code. This is mainly useful for shared data segments. There are some important restrictions involving the use of this option. See Chapter 1, “CL Command Reference,” in the *Command-Line Utilities User’s Guide* for more information.

### Data Segment Text Box

Entry	Option	Comment
<i>datasegment</i>	<code>/ND “datasegment”</code>	Renames the default data segment

## Code Segment

The `/NT` option renames the default code segment for medium-, large- and huge-model programs. It appends `_TEXT` to the specified name. It should not be used with tiny or small model.

### Code Segment Text Box

Entry	Option	Comment
<i>codesegment</i>	<code>/NT “codesegment”</code>	Renames the default code segment

## P-Code Segment

The `/NQ` option sets the name of a temporary segment for the p-code compiler; the temporary segment is removed before the program is run. This option can only be used with the `/Oq` (p-code optimization) option. During its operation, the p-code

compiler generates several temporary segments. If you encounter LINK error 1049 (“too many segments”), use `/NQ` to combine these temporary segments into one temporary segment.

#### P-Code Segment Text Box

Entry	Option	Comment
<i>pcodesegment</i>	<code>/NQ "pcodesegment"</code>	Names a temporary p-code segment

## Module Name

The `/NM` option names a module segment. It is recommended that you use the `/NT` option to rename code segments. `/NM` is maintained for compatibility with earlier versions of Microsoft C.

#### Module Name Text Box

Entry	Option	Comment
<i>modulename</i>	<code>/NM "modulename"</code>	Sets module in code segments

## V-Table Segment

The `/NV` option sets the name of a segment for far v-tables. All far v-tables in a C++ program are grouped in the specified segment.

#### V-Table Segment Text Box

Entry	Option	Comment
<i>vtablesegment</i>	<code>/NV "vtablesegment"</code>	Sets the segment name for far v-tables

## Windows Prolog/Epilog

The Windows Prolog/Epilog category options (see Figure 9.15) create prolog and epilog code for far functions in both protected mode and real mode. Select an option in the Generate Prolog/Epilog For group for either a real-mode or protected-mode application. For protected-mode applications, you can then use the Protected Mode Options check boxes.

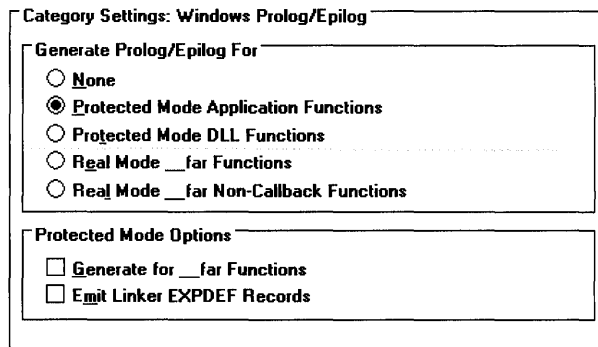


Figure 9.15 Compiler Options: Windows Prolog/Epilog

## Generate Prolog/Epilog For

In real mode, use the `/Gw` option to generate Windows prolog/epilog code for Windows callback functions. In protected mode, this code is optimized for each function using the `/GA` option for applications and the `/GD` option for DLLs. In real-mode programs that contain far functions but no callback functions, you can optimize the prolog/epilog code using the `/GW` option, which is an improved version of the `/Gq` option in previous compilers.

### Generate Prolog/Epilog For Options Buttons

Option Button	Option	Comment
None	None	Do not optimize Windows prolog/epilog code
Protected Mode Application Functions	<code>/GA</code>	Optimize Windows prolog/epilog code for protected-mode callback functions
Protected Mode DLL Functions	<code>/GD</code>	Optimize Windows prolog/epilog code for protected-mode callback functions
Real Mode __far Functions	<code>/Gw</code>	Generate Windows prolog/epilog code for real-mode far callback functions
Real Mode __far Non-Callback Functions	<code>/GW</code>	Optimize Windows prolog/epilog code for real-mode far non-callback functions

## Protected Mode Options Check-Box Summary

When you select either of the protected-mode options (/GA or /GD), the Protected Mode Options check boxes are enabled. This lets you enable the /GEe and /GEf options.

### Protected Mode Options Check Boxes

Check Box	Option	Comment
Generate for __far functions	/GEf	Treat all far functions as if they were marked as <b>__export</b>
Emit Linker EXPDEF Records	/GEe	Force emission of linker EXPDEF records

## Linker Options

To set linker options, choose Linker in the Project Options dialog box Customize Build Options group. This opens the Linker Options dialog box (see Figure 9.16). Each option category is described below, with a short description of each individual option that can be changed in that category's Category Settings group. For more information on linker options, see Help or Chapter 2, "Linking Object Files with LINK," in the *Command-Line Utilities User's Guide*.

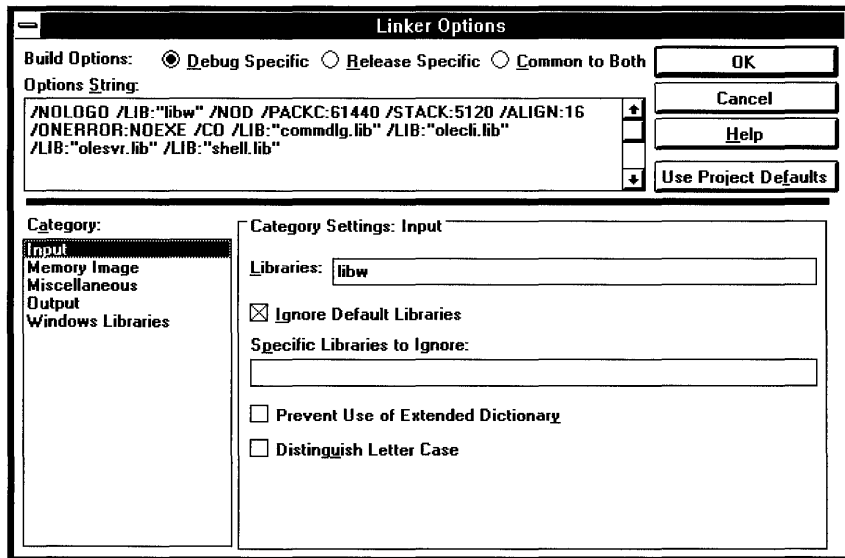


Figure 9.16 The Linker Options Dialog Box

# Input

At the beginning of the linking process, the linker attempts to resolve external references by searching libraries and retrieving the appropriate object code. The Input category (see Figure 9.17) lets you specify special library options to the linker.

Category Settings: Input

Libraries:

Ignore Default Libraries

Specific Libraries to Ignore:

Prevent Use of Extended Dictionary

Distinguish Letter Case

Figure 9.17 Linker Options: Input

## Libraries

To include libraries for the linker to search, enter the names, separated by spaces or commas, in this text box. You do not need to enter the .LIB extension, as this is assumed.

To use any of the standard Windows API libraries, type the name in the Libraries text box or use the Windows Libraries category, which contains a list box of all supported Windows libraries. If you type the name of a Windows API library in the Libraries text box, the name you typed is immediately removed from the Libraries text box and the corresponding Library name is selected in the Import Libraries and DLLs list box in the Windows Libraries category. Duplicate library names typed in the Libraries text box are also removed automatically.

### Libraries Text Box

Entry	Option	Comment
<i>libraryname</i>	<code>/LIB:"<i>libraryname</i>"</code>	Causes <i>libraryname</i> .LIB to be linked

Note that the “/LIB:” mnemonic is not a valid command-line option for LINK. It is used here to represent libraries as command-line arguments so that you can view them in one place along with other linker options. The “/LIB:” mnemonic does not appear as such in the makefile.

## Specific Libraries to Ignore

The /NOD option tells LINK not to search default libraries named in object files. If you want the linker to ignore all libraries, select the Ignore Default Libraries check box. If you want LINK to ignore specific libraries, type the name or names of those libraries in this text box, separated by spaces, commas, or semicolons. If one of the libraries you specify to ignore contains OLDNAMES.LIB, you must specify OLDNAMES.LIB in the Libraries text box.

### Specific Libraries to Ignore Text Box

Entry	Option	Comment
<i>libraryname</i>	/NOD:“ <i>libraryname</i> ”	Do not search default library <i>libraryname</i>

## Input Check-Box Summary

The check boxes in the Input category are listed together here for summary information. Each specific option is described in the sections that follow.

### Input Check Boxes

Check Box	Option	Comment
Ignore Default Libraries	/NOD	Ignore all default libraries
Prevent Use of Extended Dictionary	/NOE	Do not use extended dictionary
Distinguish Letter Case	/NOI	Do not ignore case differences

## Ignore Default Libraries

This check box sets the /NOD option, which tells LINK not to search any default libraries named in object files. Note that if you use this option, you must also specify OLDNAMES.LIB in the Libraries text box to resolve external references.

## Prevent Use of Extended Dictionary

The /NOE option prevents the linker from searching extended dictionaries when resolving references. An extended dictionary is a list of symbol locations in a library created with the Microsoft Library Manager (LIB). The linker consults extended dictionaries to speed up library searches. When LINK uses extended dictionaries, it gives an error when a duplicate definition is found. Use this option when you redefine a symbol and an error occurs.

## Distinguish Letter Case

This check box preserves case in identifiers. Since C and C++ distinguish between uppercase and lowercase, it is a good idea to use the `/NOI` linker option.

## Memory Image

Once the linker has resolved all external references, it determines how the executable file will use memory when it is loaded. The Memory Image category (see Figure 9.18) has several options for optimizing data and code segmentation, setting stack size, and other memory-related parameters.

Category Settings: Memory Image

Don't Remove Unreferenced Packaged Functions

Pack EXE File

Pack Code: 61440

Pack Data:

Stack Size: 5120

Translate Far Calls

Max. Number of Segments:

Figure 9.18 Linker Options: Memory Image

## Memory Image Check-Box Summary

The check boxes in the Memory Image category are listed together here for summary information. Each specific option is described in the sections that follow.

### Memory Image Check Boxes

Check Box	Option	Comment
Don't Remove Unreferenced Packaged Functions	<code>/NOPACKF</code>	<code>/PACKF</code> is provided by default
Pack EXE File	<code>/EXEPACK</code>	Optimize executable file for relocation
Translate Far Calls	<code>/FARCALL</code>	Optimize far calls to functions in the same segment

## Don't Remove Unreferenced Packaged Functions

By default, the linker removes unreferenced packaged functions, which may be created with the `/Gy` compiler option and are always created by C++ member functions. The `/NOPACKF` option disables this action. For a description of packaged functions, see “Enable Function Level Linking” on page 122.



## Pack EXE File

This check box produces the `/EXEPACK` option, which directs LINK to remove sequences of repeated bytes (usually null characters) and to optimize the load-time relocation table before creating the executable file. This may decrease the executable file size if it contains at least 500 load-time relocations and large streams of repeated characters.

## Translate Far Calls

The `/FARCALL` option directs the linker to optimize far calls to procedures that lie in the same segment as the caller. This can result in slightly faster code; the gain in speed is most apparent on 80286-based and later computers.

## Pack Code

The `/PACKC` option turns on code-segment packing. This is on by default for Windows-based programs and DLLs. The linker packs physical code segments by grouping neighboring logical code segments that have the same attributes. `/PACKC` changes the segment and offset addresses so that all items in a group share the same segment. This affects only programs with multiple code segments and can result in slightly faster and more compact code. The number you enter in the Pack Code text box specifies the maximum size of groups formed by `/PACKC` (the default segment size without `/PACKC` is 65,500 bytes).

The `/PACKC` option is not recommended when linking Windows-based applications with `/FARCALL`.

### Pack Code Text Box

Entry	Option	Comment
<i>n</i>	<code>/PACKC:n</code>	Pack adjacent code segment definitions

## Pack Data

The `/PACKD` option turns on data-segment packing for programs with multiple data segments. Adjacent data-segment definitions are combined into the same physical segment. The linker considers any segment definition with a class name that doesn't end in `CODE` as a data segment. The linker stops adding segments to a group when it cannot add another segment without exceeding the number (*n*) you enter in this text box (the default segment size without `/PACKD` is 65,536 bytes). Packing data segments can result in slightly faster and more compact code. You can use `/PACKD` to get around the limit of 254 physical data segments per executable file imposed by an operating system.

**Pack Data Text Box**

Entry	Option	Comment
<i>n</i>	/PACKD: <i>n</i>	Pack adjacent data segment definitions

**Stack Size**

The /STACK option lets you change the stack size from its default value of 2048 bytes. You can enter any even number up to 65,534. Do not specify /STACK for a DLL.

**Stack Size Text Box**

Entry	Option	Comment
<i>n</i>	/STACK: <i>n</i>	Set stack size to <i>n</i> bytes

**Max. Number of Segments**

The /SEG option sets the maximum number of program segments. You can use this text box to set the maximum number of segments from 1 to 16,384. Since LINK must allocate memory to keep track of each segment, the higher this number is, the less space LINK has to run in. If LINK runs out of memory, try setting this number to the actual number of segments in your program and relinking. The default is 128 segments when /SEG is not specified.

**Max. Number of Segments Text Box**

Entry	Option	Comment
<i>n</i>	/SEG: <i>n</i>	Set maximum number of program segments from 1 to 16,384

**Miscellaneous**

The Miscellaneous category (see Figure 9.19) contains just two controls—a check box to suppress banner information from the Output window, and a text box that lets you type in command-line options to the LINK utility directly without using the dialog-box controls.

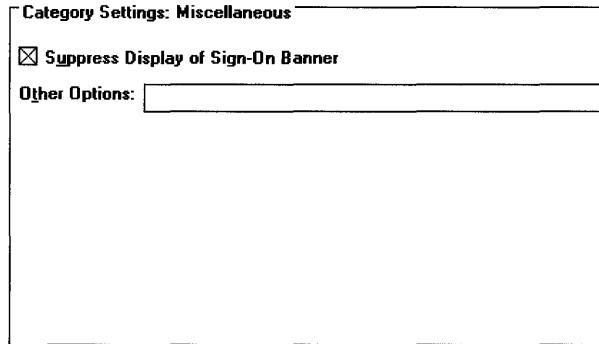


Figure 9.19 Linker Options: Miscellaneous

## Suppress Display of Sign-On Banner

The `/NOLOGO` option suppresses the copyright message displayed when LINK is invoked. This speeds the build slightly since the Output window is not updated with this information.

### Suppress Display of Sign-On Banner Check Box

Option	Comment
<code>/NOLOGO</code>	Banner not sent to Output window

## Other Options

The Other Options text box allows you to directly type in any LINK command-line option to be included in the project makefile. This lets you copy linker options from another makefile and use command-line linker options that are not available from dialog-box controls. For a complete description of all LINK command-line options, see Chapter 2, "Linking Object Files with LINK," in the *Command-Line Utilities User's Guide* (available with Visual C++ Professional Edition).

**Important** You are responsible for the accuracy of any option you enter in this text box. If Visual Workbench recognizes the option as one that can be set using a dialog-box control, it changes the dialog-box control to reflect the option and removes the option from the Other Options box. However, if the option is not recognized, it is left on the options string as is and passed to the linker.

# Output

The linker is capable of producing many different types of files besides the target file type. Options in the Output category (see Figure 9.20) give you control over which files the linker generates.

**Category Settings: Output**

**Generate Debugging Information**

**Create MAP File**

**Include Line Numbers/Addresses in MAP**

**Prevent Creation of EXE on Linker Error**

**Produce More Detailed Output**

**Produce COM File**

**Segment Alignment:**

**Figure 9.20 Linker Options: Output**

## Output Check-Box Summary

The check boxes in the Output category are listed together here for summary information. Each specific option is described in the sections that follow.

### Output Check Boxes

Check Box	Option	Comment
Generate Debugging Information	/CO	Use with /Zi, /Z7, or /Zd compiler options
Create MAP File	/MAP	Generate list of segments and public symbols
Include Line Numbers/Addresses in MAP	/LINE	Generate list of segments and line numbers/addresses
Prevent Creation of EXE on Linker Error	/ONERROR:NOEXE	Do not write executable file to disk when a linker error occurs
Produce More Detailed Output	/INFO	Display link status information
Produce COM File	/TINY	Produce tiny memory-module-executable file

## Generate Debugging Information

The `/CO` option adds Microsoft symbolic debugging information to the executable file, which allows you to use either the Visual Workbench debugger or CodeView to debug it. If the object files were not compiled with either the `/Zi`, `/Z7`, or `/Zd` option, this option places only public symbols in the executable file.

## Create Map File

The `/MAP` option creates a map file that contains a list of segments as well as public symbols sorted by name and by address. Symbols in C++ appear in the form of decorated names. The map file contains the same name as the executable file with an extension of `.MAP`.

## Include Line Numbers/Addresses in MAP

The `/LINE` option creates a map file that contains a list of segments and also the line numbers and associated addresses from source files to the map file. It does not add public symbols to the map file as the `/M` option does.

## Prevent Creation of EXE on Linker Error

By default, if certain errors occur, LINK writes an executable file to disk and overwrites any existing file having the same name. The resulting executable file has the error bit set in its header. Specify `/ONERROR:NOEXE` to prevent such a file from being written to disk and preserve any existing file having the same name.

## Produce More Detailed Output

The `/INFO` option displays to the Output window information about the linking process, including the phase of linking, the object files being linked, and the library modules used.

## Produce COM File

Select this check box to produce the `/TINY` option. This causes LINK to produce a `.COM` file instead of an `.EXE` file. When the `/CO` option (Generate Debugging Information) is used with `/TINY`, debug information is put in a separate file with the same base name as the `.COM` file and with the `.DBG` extension. A C or C++ program linked as a `.COM` file must consist of only one physical segment, must not use far references, and cannot be a Windows-based program.

## Segment Alignment

The `/ALIGN` option aligns segments in a segmented executable file at the boundaries specified by the number you enter in the Segment Alignment text box. The alignment size is in bytes and must be an integer power of two, or else is rounded up by LINK to the next higher power of two. The default alignment is 512 bytes when `/ALIGN` is not specified. This only affects Windows-based programs.

### Segment Alignment Text Box

Entry	Option	Comment
<i>n</i>	<code>/ALIGN:n</code>	Align segments at boundaries of <i>n</i> bytes

## Windows Libraries

The Windows Libraries category (see Figure 9.21) has a single list box to allow you to easily find and use the Windows API libraries that you want.

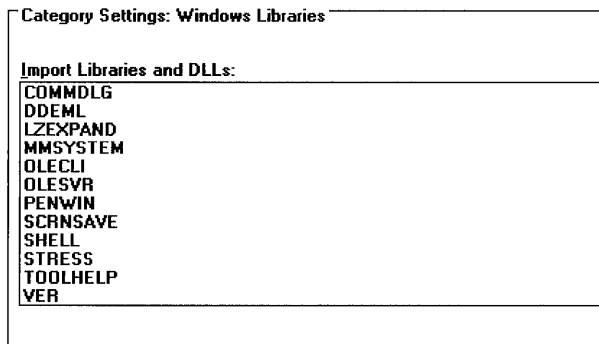


Figure 9.21 Linker Options: Windows Libraries

## Import Libraries and DLLs

Use this multiple-selection list box to instruct the linker to link any of the Windows version 3.1 API libraries that you select. For each Windows library you select, an entry of `/LIB:"libraryname"` (where *libraryname* represents the Windows library) appears in the options string.

**Note** The `/LIB:` mnemonic is not a valid command-line option for LINK. It is used here to represent libraries as command-line arguments so that you can view them in one place along with other linker options. The `/LIB:` mnemonic does not appear as such in the makefile.

### Import Libraries and DLLs List Box

List Entry	Option (See preceding note)	Comment
COMMDLG	/LIB:"COMMDLG"	Common dialog box templates and procedures
DDEML	/LIB:"DDELM"	Dynamic Data Exchange
LZEXPAND	/LIB:"LZEXPAND"	Lempel-Ziv data compression
MMSYSTEM	/LIB:"MMSYSTEM"	Windows Multimedia Extension
OLECLI	/LIB:"OLECLI"	Object linking and embedding client
OLESVR	/LIB:"OLESVR"	Object linking and embedding server
SCRSERVER	/LIB:"SCRSERVER"	Screen saver, WinMain, and other startup code
SHELL	/LIB:"SHELL"	Registration, Drag&drop, and file association
STRESS	/LIB:"STRESS"	Application stressing
TOOLHELP	/LIB:"TOOLHELP"	User heap, GDI heap, memory management
VER	/LIB:"VER"	File Installation

## Resource Compiler Options

The Resource Compiler Options dialog box lets you select specific options to the Microsoft Resource Compiler, which is run from the makefile whenever you include a resource file (.RC) in the project list.

You select resource compiler options by either selecting check boxes or by typing the options into the Custom Options or Defines text boxes (see Figure 9.22). If a conflict occurs between a check box option and a text box option, the one specified in the text box takes precedence.

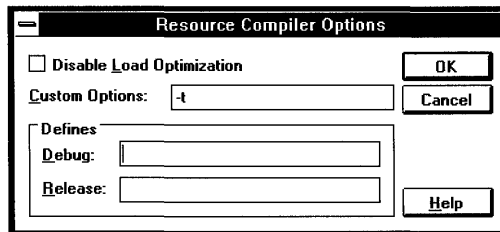


Figure 9.22 Resource Compiler Options Dialog Box

## Disable Load Optimization

This check box tells the resource compiler not to arrange preload information into contiguous segments. This is the same as the `/K` custom option.

## Custom Options

The Custom Options text box lets you set specific resource compiler options by simply typing them in. A complete description of Microsoft Resource Compiler options can be found in Help.

## Defines

The Debug and Release text boxes let you define symbols to be checked by the resource compiler dependent upon the current build mode. This lets you specify conditional branching in a resource script file based on whether a term is defined in the Defines text box that corresponds to your build option.

For example, you could define `DEBUG` in the Debug text box and use the directives `#ifdef DEBUG` and `#endif` in your resource script to surround a Debug menu statement. This menu would then appear when you perform a debug build but would not appear in a release build.

The Defines text boxes take the place of using the `/D` custom option.





# Using the Browser

Part of the process of program development involves understanding relations between program elements, such as between base classes and derived classes or between called and calling functions. It is also important to be able to move quickly between related program elements that may be located in several different files.

The Visual Workbench graphical browser, an integrated version of the Microsoft Source Browser, uses information generated by the compiler to help you find related symbols and display symbol relationships. The generated database contains information about where each symbol is defined and used, and about the relationships among modules, constants, macros, variables, functions, and classes.

This chapter introduces you to the capabilities of the browser, including creating, opening, and querying a browser database. The chapter also shows techniques for viewing class hierarchies, function call trees, and symbol definitions and references.

## Creating a Browser Database

To use the browser, you must first create a browser database. Visual Workbench does this for you automatically as part of the normal build process when the Browser Information check box in the C/C++ Compiler Options dialog box is enabled.

Note that the Browser Information compiler option is enabled by default for all project types. Since generating a browser database slows down the build and browser database files can be very large, you may want to turn off Browser Information when you do not specifically want to build a browser database.

The following procedure assumes that you have a project loaded in Visual Workbench. You may want to use the SCRIBBLE sample project located in the `\MSVCMFC\SAMPLES\SCRIBBLE` directory as an example to experiment with

the browser's capabilities. See Chapter 8, "Using Projects," to learn how to open and build a project.

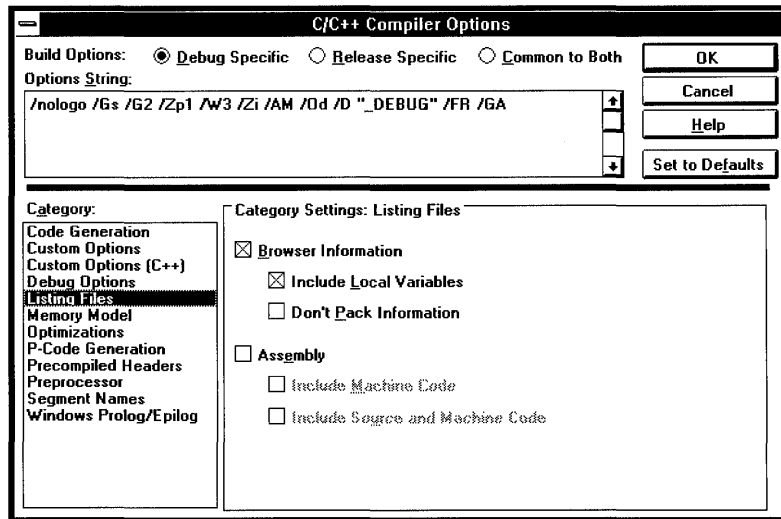
► **To create a browser database (.BSC) file:**

1. From the Options menu, choose Project.  
The Project Options dialog box appears.
2. Under Customize Build Options, choose Compiler.  
The C/C++ Compiler Options dialog box appears.
3. In the Category box, select Listing Files.

This opens the Category Settings: Listing Files group in the lower right-hand corner of the dialog box, which contains the Browser Information categories (see Figure 10.1).

4. Select the Browser Information check box to enable it, if it isn't already enabled.

Include Local Variables should be checked and Don't Pack Information should be cleared (not checked). (See page 129 for information on customizing the Browser Information options.)



**Figure 10.1 C/C++ Compiler Options: Listing Files**

5. Choose OK in the C/C++ Compiler Options dialog box.

6. Choose OK in the Project Options dialog box.
7. Click the Rebuild button on the toolbar or choose Rebuild All *Targetname* from the Project menu.

During the build, Visual Workbench creates a browser database file with the name of your project and the extension .BSC in your project directory.

## Opening a Browser Database

You can open a browser database by using one of four methods:

- Choosing the Open command on the Browse menu
- Choosing the Open command on the File menu
- Choosing the Next Definition or Next Reference command on the Browse menu
- Dragging a browser database file (.BSC) from the Windows File Manager and dropping it into Visual Workbench

The first method, choosing the Browse menu's Open command, opens the Browse window. The Open command is normally followed by a browser database filename, which is usually the name of the current project plus a .BSC extension. However, if you have opened a browser database using the Open File dialog box, the name of the most recently opened database appears after the Open command.

The Open command on the Browse menu appears with no name following it if there isn't an active project and you haven't opened a browser database since you started Visual Workbench. In this case, choosing the Open command opens the Open File dialog box with a file filter of \*.BSC.

### ► To open the Browse window for the current project:

- From the Browse menu, choose Open *Project*.BSC (where *Project* is the name of the current project).

If you have created a browser database for the current project, the Browse window appears. Otherwise, a message appears, indicating that the browser database file cannot be opened.

The second method of opening a browser database is to directly open any existing browser database by using the Open File dialog box. This can be a browser database generated by Visual Workbench or by Microsoft Programmer's WorkBench.

► **To open a browser database using the Open File dialog box:**

1. From the File menu, choose Open.

Or click the Open button on the toolbar.

Or choose Open from the Browse menu if no filename follows the Open command (that is, Open is followed by an ellipsis ...).

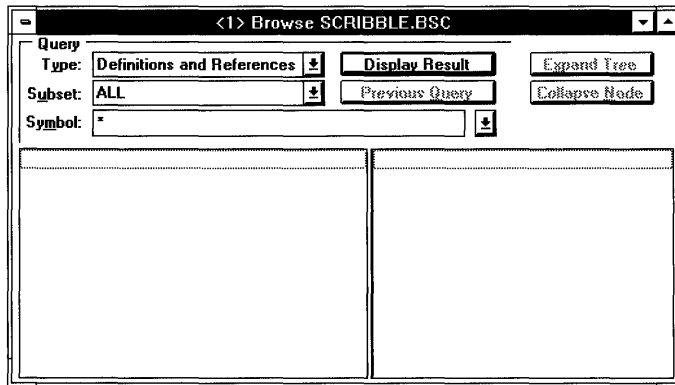
The Open File dialog box appears.

2. From the drop-down list in the List Files of Type box, select Browse Database (\*.bsc).
3. Use the Directories and Drive list boxes to locate the directory that contains the browser database.

The browser file appears in the file list.

4. Double-click the browser database filename.

The Browse window appears (see Figure 10.2).



**Figure 10.2** The Browse Window

When you open a browser database in this manner, the filename of the currently open database appears on the Browse menu following the Open command. When you close the Browse window, the Browse menu's Open command continues to display the name of the most recently used database.

The third method of opening a browser database is to use the Browse menu's Go to Definition or Go to Reference command when a browser database is associated (named on the Browse menu) but not open.

► **To open a browser database and jump to the first definition or reference:**

1. Select a symbol name in a source file (or place the insertion point at the beginning of a symbol name).  
Note that the browser database filename associated with the source file must appear on the Browse menu after the Open command.
2. Choose Go to Definition (F11) to jump to the first definition, or Go to Reference (SHIFT+F11) to jump to the first reference of the symbol.

The database named in the Browse menu's Open command is opened, the Browse window is minimized, and the source file containing the first definition (or first reference if you chose Go to Reference) appears with the insertion point at that definition or reference.

See "Browsing Definitions and References" on page 176 for information on using the Browse window for querying definitions and references.

Finally, as with source files and resource files, you can use the Windows File Manager to display a file icon and drag and drop the file icon into Visual Workbench. When the file icon is a browser database file (with an extension of .BSC), the Browse window opens with the browser database.

## Querying a Browser Database

You can search for information in a browser database by using one of two basic methods:

- Using the query controls in the Browse window
- Using the commands on the Browse menu

The advantage of the Browse window is that it lets you see graphical representations of interclass and interfunction relationships and lets you construct lists of symbols from which to query definitions and references.

The advantage of using the commands on the Browse menu is that you can jump immediately to definitions and references of a symbol while you are writing or editing code. The Browse menu's Go to Definition and Go to Reference commands work in conjunction with the Next and Previous commands to traverse all definitions or references of a particular symbol.

This section provides a quick summary of querying a browser database using both methods. For more detailed information, see the specific section devoted to each of the browser query types later in this chapter.

## Using the Browse Window

To perform a query in the Browse window, you first set the query parameters in the Query group at the top of the Browse window (see Figure 10.3) and then choose the Display Result button.

Query	
Type:	Definitions and References <input type="button" value="↓"/>
Subset:	ALL <input type="button" value="↓"/> <input type="button" value="Display Result"/>
Symbol:	* <input type="button" value="↓"/>

**Figure 10.3** Browse Window Query Group

The Query group contains three controls to set and display query parameters:

### Type

This determines the type of symbol relationship you want to view. The five choices are:

- Definitions and References
- Call Graph
- Caller Graph
- Derived Class Graph
- Base Class Graph

### Subset

This determines or displays the type of symbols you are browsing. For the Call Graph and Caller Graph query types, Subset is always Functions. For the Derived Class Graph and Base Class Graph query types, Subset is always Classes.

For Definitions and References queries, Subset lets you determine the subset of symbol types to be included in the search. This can be useful when you use a wildcard in the Symbol text box that matches several symbol names and a lengthy list of symbols appears in the left display panel. You can narrow the results list by choosing a subset of symbol types from the Subset drop-down list. Or you can choose ALL to have a list of all symbols placed in the left display panel.

### Symbol

This determines the specific function, variable, type, macro, or class you want to browse. You can type in a symbol name or choose from the previous eight symbol names you have queried.

You can also use a wildcard in the symbol name as described in Table 10.1.

**Table 10.1 Wildcard Types in Symbol Names**

Type	Represents
*	All symbols
<i>Sym</i> *	All symbols starting with some common characters (denoted here by <i>Sym</i> )
<i>ClassName</i> ::	All member functions and member variables of a class (denoted here by <i>ClassName</i> ) when the query type is Definitions and References

When a wildcard matches more than one symbol, all matching symbols appear either in the left display panel (if the query type is Definitions and References) or in a disambiguation dialog box (for all other query types). When a disambiguation dialog box appears, you must select just one of the symbols, since all graphs require a single root.

The following procedure assumes you have opened a Browse window:

► **To query the browser database from the Browse window:**

1. From the Type drop-down list box, select a query type.
2. In the Subset drop-down list box, if the query type is Definitions and References, select a symbol type for the query to search, or select ALL to search all symbol types.

For example, to display all member functions of the class **CCmdTarget** in the left panel, type **CCmdTarget :** in the Symbol text box and select Functions from the Subset list box.

3. In the Symbol text box, type the name of the symbol you want to browse.  
Or, using the drop-down list, select a symbol you have browsed recently.  
Or type a wildcard combination to match one or more symbol names.
4. Choose the Display Result button, or press ENTER, to start the query.

If the query type is Definitions and References, the left panel below the Query group displays the symbol, or lists of symbols. For all other query types, the left panel displays a graph.

Techniques for manipulating the graphs and lists once you have performed a query in the Browse window are described in “Browsing Classes and Functions” on page 169.

To recall the state of the previous browse operation, choose the Previous Query button. By choosing the Previous Query button repeatedly, you can recall all the browse operations you have performed since opening the Browse window.



## Using Menu Commands

You can use menus and shortcut keys to browse for definitions and references directly from any symbol in a source file associated with the open database file, or any symbol in the Browse window. To do this, you use the five menu commands on the Browse menu:

- Go to Definition (F11)
- Go to Reference (SHIFT+F11)
- Next (CTRL+NUMPAD+)
- Previous (CTRL+NUMPAD-)
- Pop Context (CTRL+NUMPAD\*)

► **To query the browser database from a source file:**

1. Select a symbol by either highlighting it or placing the insertion point to the left of it.
2. From the Browse menu, choose Go to Definition (F11) to jump to the first definition of that symbol.

Or choose Go to Reference (SHIFT+F11) to jump to the first reference of that symbol.

When you choose Go to Definition, a source window opens on the file containing the definition of the selected symbol, with the insertion point placed at the definition's location in the source file. If the source for the definition is not available, such as a definition for a library function like **printf()**, the insertion point jumps instead to the first reference, which is usually the prototype.

When you select Go to Reference, a source window opens on the file containing the first reference to the selected symbol, with the insertion point placed at the reference's location in the source file.

If the Browse window isn't already open, it is opened and minimized the first time you choose either Go to Definition or Go to Reference.

When you choose either Go to Definition or Go to Reference, a single list is formed and kept internally of all definitions or references of that symbol, depending on which menu command you chose. You can traverse this list forward or backward using the Next and Previous command on the Browse menu. To change the list type—for example, from definitions to references—you must use one of the “Go to” commands first.

► **To jump to the next or previous definition or reference in a source file:**

- From the Browse menu, choose Next or Previous.

If you last chose Go to Definition, a source window appears with the insertion point on the Next or Previous definition of the queried symbol.

If you last chose Go to Reference, a source window appears with the insertion point on the Next or Previous reference of the queried symbol.

► **To return to the last symbol queried:**

- From the Browse menu, choose Pop Context.

The source window containing the current query context (the symbol selected before the last Go to Definition or Go to Reference) is made active displaying the queried symbol.

---

**Note** The current position of the definition or reference in a list generated by a “Go to” command and traversed using the Next and Previous Browse menu commands is not related to the currently selected definition or reference in the Browse window.

---

## Browsing Classes and Functions

One of the most appealing features of the browser is the ability to graphically view C++ class hierarchies and relationships between called and calling functions.

Once you have queried the browser database for a graphical representation, you can manipulate the nodes of the resulting tree to view a small section or the fully expanded tree. As you click each node, the browser displays a list of definitions and references for the symbol at that node. And, for class trees, the browser also displays a list of member functions and variables for each node you select.

This section describes in detail how to access graphical data produced by class and function queries.

## Overview of Graphical Browser Query Types

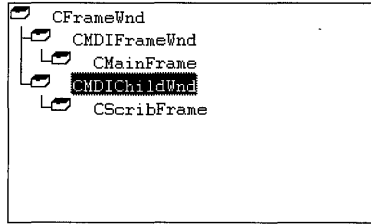
Graphical query types available in the Browse window include:

- Call Graph
- Caller Graph
- Derived Class Graph
- Base Class Graph

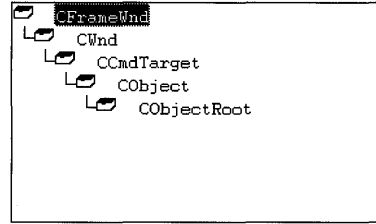
The browser displays relationships between base and derived C++ classes, and between called and calling functions, as graphical trees. This is similar to the way the Windows File Manager displays directories. In a class graph, each node of the tree represents a class; in a function graph, each node represents a function. Each node can be expanded if it contains further branches or collapsed to hide all its branches.

Figure 10.4 illustrates the graphs used to display C++ class-hierarchy information. A Derived Class Graph branches from left to right with the base class on the left and derived classes expanding to the right. A Base Class Graph also branches from left to right, but with the derived class on the left and the class or classes from which it is derived expanding to the right.

A Derived Class Graph



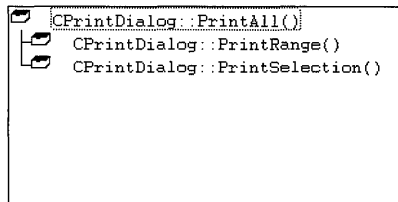
A Base Class Graph

**Figure 10.4** Class Graphs

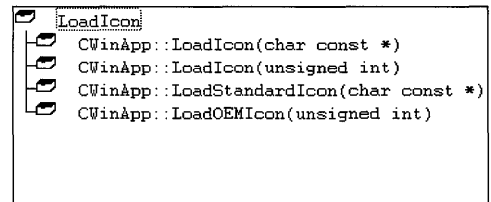
Derived Class Graphs and Base Class Graphs automatically provide a list of member functions and member variables in the top window panel to the right of the graph. You can click a member variable name to open the source window where it is defined, or click a member function name to open the source window where it is implemented.

Figure 10.5 illustrates the graphs used to display function-calling relationships. In a Call Graph, a node on the left represents a function that calls all functions labeled in nodes to its right. Conversely, in a Caller Graph, a node on the left represents a function that is called by all the functions labeled in nodes to its right.

A Call Graph



A Caller Graph

**Figure 10.5** Call and Caller Graphs

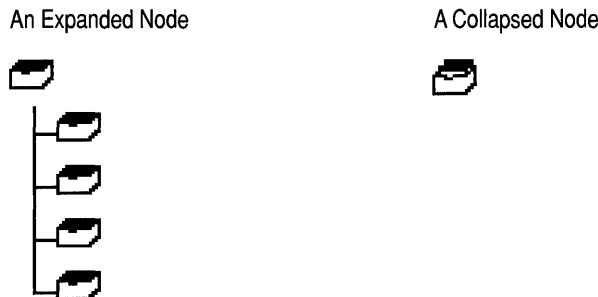
The browser also displays locations of symbol definitions and references. This is done automatically when you select any of the four query types just mentioned (Derived Class Graph, Base Class Graph, Call Graph, or Caller Graph). To see where any class or function in a graphical tree is defined and a list of all locations where it is referenced, simply click the node. The definitions and references appear to the right of the graph.

You can double-click any reference or definition to open the corresponding source file with the insertion point on the corresponding line.

You can also specifically request a query type of Definitions and References. This has the advantage that you can apply the search to all types of symbols, including variables, types, macros, and labels as well as classes and functions.

## Expanding and Collapsing Graphs

A node in a class graph or function graph can be collapsed or expanded to hide or display subordinate nodes. A node icon that looks like an empty file box indicates that a node has either been expanded or has no subordinate nodes. A node icon that looks like a file box containing files indicates that a node has subordinate nodes but is collapsed (see Figure 10.6).



**Figure 10.6** Expanded and Collapsed Nodes

You can expand a graph one level at a time, or all at once, using the mouse or various keyboard shortcuts. The “active node” refers to the selected symbol in the graph.

### ► To display different levels of a graph:

- To expand the active node one level, choose the Expand Node button, or double-click the node, or press PLUS SIGN (+) on the number pad.
- To collapse the active node one level, choose the Collapse Node button, or double-click the node, or press MINUS SIGN (–) on the number pad.
- To fully expand the active branch, press ASTERISK (\*) on the number pad.
- To expand the entire graph, choose the Expand Tree button or press ALT+X.

## Browsing Classes

When you select Derived Class Graph or Base Class Graph as a query type, the browser displays a graphical tree. You can view class hierarchical relationships by picking a class name and then deciding to view either a tree of all classes derived from the class (Derived Class Graph) or a tree of all the classes from which it is derived (Base Class Graph).

Once a class graph is displayed, you can view information about each class in the two window panels to the right of the graph. When you select any class in the graph, the location where it is defined and all locations where it is referenced are displayed in the bottom right panel. All member functions and variables are displayed in the top right panel.

### Derived Class Graph

The Derived Class Graph query type is used to display a graph of all classes derived from the class you select as the base class.

► **To display a tree of all classes that are derived from a class:**

1. From the Type drop-down list box, select Derived Class Graph.

Notice that Classes appears automatically in the Subset box and is the only choice.

2. In the Symbol box, type the class name, type a wildcard, or select a previously used symbol from the drop-down list.

For example, type CWnd.

If you type a wildcard, such as C\*, that matches several symbol names, a disambiguation dialog box appears with a list of all symbols matched by the wildcard. Select one name from the list and choose OK.

3. Choose the Display Result button or press ENTER.

The CWnd derived class graph is displayed in the Browse window (see Figure 10.7).

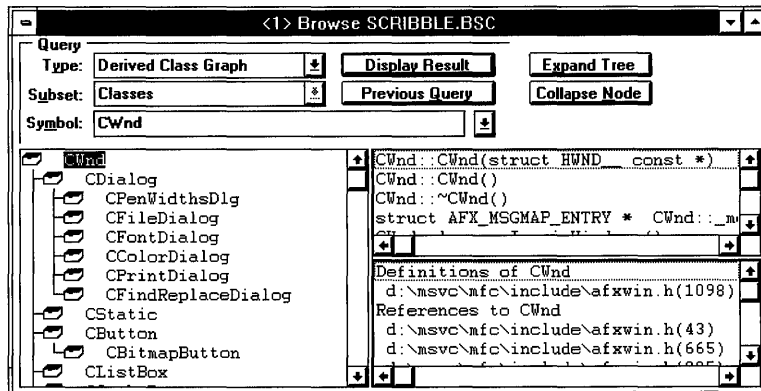


Figure 10.7 Derived Class Graph of the CWnd Class

## Base Class Graph

When you want to view the inheritance path of a particular class, select the Base Class Graph query type. This graph is a chain of single nodes unless the class uses multiple inheritance, in which case the multiple base classes appear as multiple branches.

► **To display a graph of base classes from which a class is derived:**

1. From the Type drop-down list box, select Base Class Graph.

Notice that Classes appears automatically in the Subset box and is the only choice.

2. In the Symbol box, type the class name, type a wildcard, or select a recently queried class from the drop-down list.

For example, select or type `CDialog`.

3. Choose the Display Result button or press ENTER.

The `CDialog` base class graph appears in the Browse window.

4. To see the complete inheritance path (Figure 10.8), choose Expand Tree.

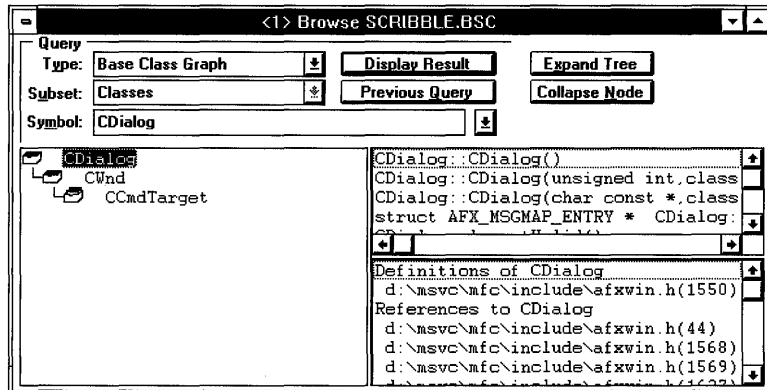


Figure 10.8 Base Class Graph of the CDialog Class

## Member Functions and Variables

Member functions and variables are displayed automatically as a result of any query using the Derived Class Graph or Base Class Graph query type.

- ▶ **To display a list of all member functions and variables belonging to a class:**
  1. Perform either a Derived Class Graph or Base Class Graph query.
    - Either specify the class in the Symbol box or make sure it is included in the resulting graph.
  2. Click the name of the class in the resulting graph to make it the active node.
    - The upper panel to the right of the class tree displays all member functions and variables belonging to the class (not inherited).

---

**Tip** You can also get a list of all member functions and variables by using the Definitions and References query type and specifying the wildcard *ClassName::*, where *ClassName* represents the name of the class whose member functions and variables you want to examine.

---

## Browsing Function Relationships

You can use the browser to graphically display relationships between calling and called functions by selecting the Caller Graph or Call Graph query type. These graphs are typically tree structures unless they display a recursive function node. Nodes representing recursive functions, or any functions that appear more than once, are followed by an ellipsis (...).

To get a listing of where a function is defined and all the places it is referenced, select the function in the Caller Graph or Call Graph tree. The panel to the right of the graph panel in the Browse window lists the locations of the function definition and all references to it. Double-click any location in this list to open a source window at that location in the specified file.

► **To display a graph of all functions that are called by a function:**

1. From the Type drop-down list box, select Call Graph.

The Subset box displays Functions.

2. In the Symbol box, type the function name, type a wildcard, or select a previously queried function from the drop-down list.
3. Choose the Display Result button or press ENTER.

The call graph of the selected function is displayed (see Figure 10.9).

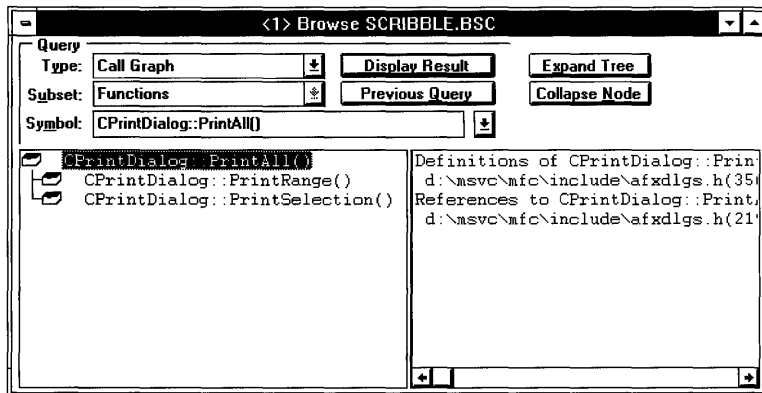


Figure 10.9 A Call Graph

► **To display a graph of all functions that call a function:**

1. From the Type drop-down list box, select Caller Graph.

The Subset box displays Functions.

2. In the Symbol box, type the function name, type a wildcard, or select a previously queried function from the drop-down list.

Note that if you type the name of a function with more than one definition (such as a C++ class constructor), a disambiguation dialog box appears to let you choose the function from a list box. This dialog box also appears when you use wildcards that match more than one symbol name.

3. Choose the Display Result button or press ENTER.

The caller tree of the selected function is displayed.



## Browsing Definitions and References

You can use the browser to quickly move between files and access common program elements. For example, you may want to examine or change a definition of a function but are not sure where the function is defined. Or, after changing a function's definition, you may want to find every place where it is referenced.

When you select a class or function graph query type, references and definitions are automatically displayed for the active node to the right of the graph. For references and definitions of all other types of symbols, or to get just reference or definition information on any class or function, use the following procedures.

---

**Note** The browser database is based on the state of source files at the time of the last build. If you edit source files and then perform browse operations, the locations of definitions and references may not be accurate.

---

If a definition is listed as <Unknown>, it is probably defined in a library for which you are not provided the source code. For example, the macro `_MSC_VER` is generated by the compiler and defined in code that isn't provided with Visual C++.

► **To display a list of locations where a symbol is defined or referenced:**

1. From the Type drop-down list box, select Definitions and References.
2. In the Subset drop-down list box, select the symbol type you are interested in.
3. In the Symbol box, type the symbol name.

Or, using the drop-down list, select a symbol you have browsed recently.

Or type a wildcard combination to match one or more symbol names (for example, type `*` to produce a list of all symbol names).

4. Choose the Display Result button or press ENTER.

A list of all symbols that match the query parameters appears in the left display panel. The list may contain a single symbol, if you typed an exact name of a symbol, or many symbols if, for example, you used a wildcard or typed the name of an overridden C++ class member function.

5. Click any symbol in the left panel to get a list of its definitions and references in the panel on the right.

The top symbol in the list is selected by default when there is more than one symbol.

---

Once you have a list of definitions and references, you can jump directly from the Browse window to a definition or reference using either of two methods.

► **To jump to any definition or reference from the Browse window:**

1. Select the symbol in the left display panel.
2. Double-click the definition or reference in the list in the right panel.

Or select the item and press ENTER.

The source file is opened (or is made the active file if it is already open), and the insertion point is placed on the line containing the definition or reference.

► **To jump to the first definition or reference from the Browse window:**

1. Select the symbol in the left display panel.
2. Choose Go to Definition (F11) to jump to the first definition, or choose Go to Reference (SHIFT+F11) to jump to the first reference of the symbol.

Once you have jumped to the first definition of, or reference to, a particular symbol using this method, you can jump to subsequent definitions or references using the Next (CTRL+NUMPAD+) and Previous (CTRL+NUMPAD-) commands on the Browse menu.

---

**Note** The current position of the definition or reference in a list generated by a “Go to” command (and traversed using the Next and Previous Browse menu commands) is not related to the currently selected definition or reference in the Browse window.

---



# Debugging Programs

Debugging a program is a two-phase process. The first phase involves correcting compiler and linker errors during the build process. These errors usually consist of incorrect language syntax, undeclared variables, or misspelled keywords.

The second debugging phase occurs after any syntax errors are corrected and the project is successfully built. If the program does not perform correctly, you need to analyze its internal workings. This means using a debugger to set breakpoints and examine variables, which allows you to locate the bug, correct it with the editor, and rebuild the program.

Visual Workbench offers features for performing both phases of debugging. Visual Workbench supports debugging of both EXE and DLL projects for Windows. This chapter covers debugging an EXE project. If you want to debug a DLL project, you need to first create an appropriate EXE shell program to call the DLL. Using the Debug dialog box, which is accessed from the Options menu, you specify the name of the program that calls the DLL project.

As an alternative to using the Visual Workbench debugger, you can use the Microsoft CodeView debugger. If you have Visual C++ Professional Edition, this tool is provided in both Window-hosted and MS-DOS–hosted versions. You need to use CodeView if you are debugging any application for MS-DOS or p-code application for Windows. Visual Workbench debug information is fully compatible with both versions.

If you are familiar with CodeView, you'll find the Visual Workbench debugger similar in operation and functionality. Both debuggers use similar breakpoint syntax; Watch, Locals, and Registers windows; and QuickWatch, Call Stack, and tracing capabilities. CodeView supports a command window for command-line syntax and a memory window, whereas the Visual Workbench debugger uses graphical controls entirely and has no memory window. However, only the Visual Workbench debugger is fully integrated with all the other Visual Workbench facilities, such as the browser, editor, and Class Wizard. For information on using CodeView, see the *CodeView Debugger User's Guide*.

In this chapter, you'll use the SORTDEMO sample program to learn debugging strategies and techniques within Visual Workbench.

## Using the Debugging Windows

Visual Workbench displays information in a series of windows that you can view as you debug a program. You activate these windows with commands from the Window menu.

Table 11.1 lists the windows and information they show.

**Table 11.1** Debugging Windows

Window	Information
Watch	Values of variables and expressions in the Watch window. Watch expressions are entered directly into the Watch window. Values are displayed only while using the debugger.
Locals	Values of local variables within the function currently being stepped through. Values are used only while using the debugger.
Registers	Current contents of the memory and status registers.
Output	Information about the build process, including any compiler errors. This window also displays output from <b>OutputDebugString</b> function calls or the class library <b>afxDump</b> dump context object during a debugging session.

---

You can size and minimize these windows during debugging so that you can see various types of information at one time.

Figure 11.1 shows the debugging windows.

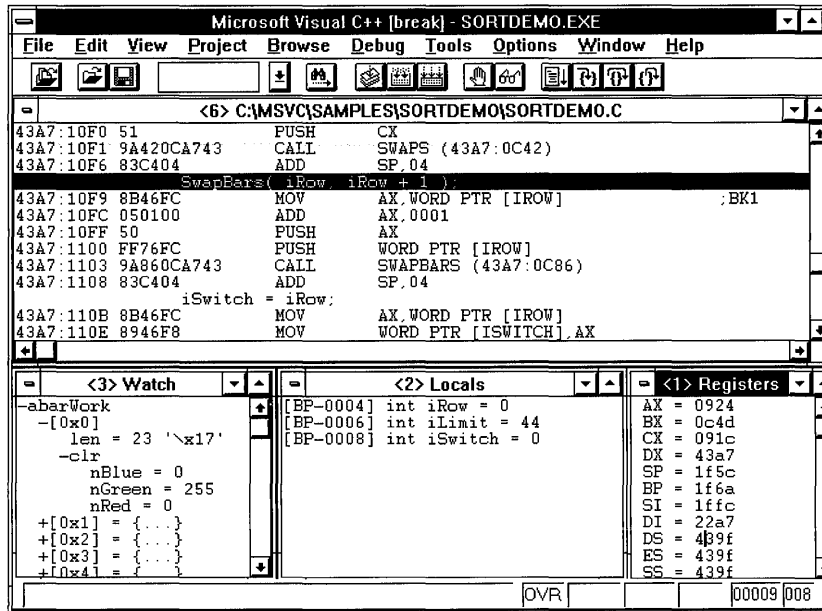


Figure 11.1 Debugging Windows

## Debugging During Building

Language syntax errors are the most common errors preventing you from successfully building a program. When a program is built, any compiler and linker errors are displayed in the Output window. If you need help on an error, move the insertion point to the error number and press F1 to display online information about the error.

### ► To move through the list of errors:

- From the View menu, choose Next Error (F4) to highlight the error following the current error.  
–Or–
- Choose Previous Error (SHIFT+F4) to highlight the error preceding the current error.  
–Or–
- Double-click or press ENTER on an error in the Output window.

As each error is highlighted in the Output window, the corresponding line containing the error is highlighted in the source window, where you can fix it.

To get help on any error or warning in the Output window, place the insertion point on the line containing the error or warning and press F1.

Note that the Output window behaves like a source window, allowing you to copy and print information from the window.

## Using the Visual Workbench Debugger

Phase two of debugging occurs after your program is built, but doesn't work correctly. In this phase, you use the Visual Workbench debugger to set breakpoints in the source code, view variables, and control program execution.

---

**Note** Debug programs are slower and larger than release programs.

---

## Preparing a Debug Version of a Program

Before a program can be debugged, you must include debugging information in the executable file with the project. You can use the SORTDEMO project as an example in the following procedure.

► **To prepare a project for debugging:**

1. Open the project.

To use the example project, open SORTDEMO.MAK, which is in the `\MSVC\SAMPLES\SORTDEMO` directory, if it is not already open.

2. From the Options menu, choose Project.

The Project Options dialog box appears (see Figure 11.2).

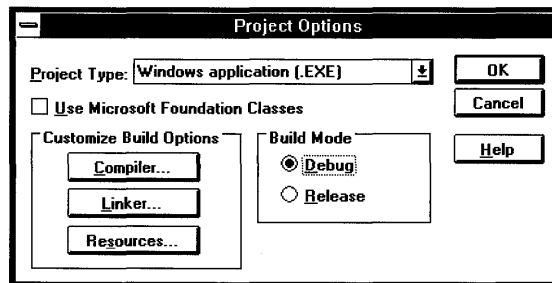


Figure 11.2 The Project Options Dialog Box

3. Under Build Mode, select the Debug option if it isn't already selected.
4. Choose OK to close the Project Options dialog box.

5. From the Project menu, choose Build.

The current project is compiled and linked and, if no errors occur, a version of the executable file containing debug information is created.

## Setting and Removing Breakpoints

Breakpoints are useful when you have a general idea of where a bug occurs in a program. The debugger runs until it reaches the breakpoint, then stops. At this point, you can step to the next line of code or trace through a function until you find the problem. While the program is paused at a breakpoint, you can also examine variable values using the QuickWatch dialog box or the Watch or Locals window, or examine register values using the Registers window.

You can set and clear breakpoints using either the Breakpoints dialog box or the Toggle Breakpoint button on the toolbar. The Toggle Breakpoint toolbar button simply sets or clears a breakpoint at the insertion point. The Breakpoints dialog box allows you to set more complex breakpoints, such as breaking if an expression is true or breaking on a window message.

All breakpoints that are active when a project is closed are saved as part of the project information and become active when the project is reopened. (Expressions and variables in the Watch window are also saved as part of the project information and restored when the project is reopened.) A project is closed when you choose Close from the Project menu, when you open or create another project, or when you quit Visual Workbench.

### Quick Access to Setting and Removing Breakpoints

The Toggle Breakpoint button on the toolbar is a quick way to set and clear breakpoints in a program. If you prefer to use shortcut keys, use F9 to toggle breakpoints.

► **To quickly set a breakpoint:**

1. Move the insertion point to the line where you want the program to break.
2. Click the Toggle Breakpoint button on the toolbar (or press F9).

Visual Workbench highlights the line, indicating that the breakpoint is set.

► **To quickly remove a breakpoint:**

1. Move the insertion point to the line containing the breakpoint.
2. Click the Toggle Breakpoint button on the toolbar (or press F9).



This is all you need to know to get started using breakpoints. If you want to learn more about the Breakpoints dialog box, read the next section. To try a sample breakpoint debugging session using the Toggle Breakpoint toolbar button and the SORTDEMO sample program, step through the following procedure:

► **To set and use a breakpoint in the SORTDEMO program:**

1. Open the SORTDEMO project if it is not already open.
2. Build the SORTDEMO project.

Follow the procedure in “Preparing a Debug Version of a Program” on page 182 if you have not already done this.

3. Open the SORTDEMO.C project file.
4. Move the insertion point to a line containing code inside the BubbleSort() function (line 741 for example).
5. Click the Toggle Breakpoint button on the toolbar.

Visual Workbench highlights the line, indicating that a breakpoint is set on this line (see Figure 11.3).

```

<3> C:\MSVC\SAMPLES\SORTDEMO\SORTDEMO.C
// BubbleSort: BubbleSort cycles through the elements, comparing
// adjacent elements and swapping pairs that are out of order. It
// continues to do this until no out-of-order pairs are found.
//
void BubbleSort()
{
    int iRow, iSwitch, iLimit = nBar-1;

    // Move the longest bar down to the bottom until all are in order.
    do
    {
        iSwitch = 0;
        for( iRow = 0; iRow < iLimit; iRow++ )
        {
            // If two adjacent elements are out of order, swap their values
            // and redraw those two bars.
            //
            iCompares++;
            if( abarWork[iRow].len > abarWork[iRow + 1].len )
            {
                Swaps( &abarWork[iRow], &abarWork[iRow + 1] );
                SwapBars( iRow, iRow + 1 );
                iSwitch = iRow;
            }
        }
    }
}

```

**Figure 11.3** Setting a Breakpoint in SORTDEMO.C

6. To run the program, click the Run button on the toolbar.

The SORTDEMO program loads and a window appears with the colored bars in random order (you can sort the bars using any of several sort procedures in SORTDEMO's Sort menu).

7. To cause the program to execute the program section containing the breakpoint, choose Bubble Sort from SORTDEMO's Sort menu.

The program runs until the breakpoint is reached and then returns the focus to Visual Workbench at the line in the program containing the breakpoint.

With the program paused, you are free to examine and modify its state. You can examine and change variable values using the Watch window, Locals window, or QuickWatch dialog box, or examine and change register values and status flags using the Registers window. You can also show the call stack that led to the current state.

8. To continue the program to its completion, click the Run button on the toolbar again (or press F5) and choose Exit from SORTDEMO's File menu.

There are several ways you can proceed through the program after a breakpoint has been reached. These are described in "Controlling Program Execution" on page 189.

## Using the Breakpoints Dialog Box

The Breakpoints dialog box (see Figure 11.4) keeps a list of all breakpoints assigned to your project. Breakpoints can be set in any of your project source files or in an executable file. To add a breakpoint, you first select the breakpoint type, fill in parameters if needed, and then choose the Add button. To remove a breakpoint, you select it in the list and choose Delete. To make a breakpoint inactive, you select it and choose Disable. For additional information on breakpoints, search for breakpoints in Help or press F1 in the Breakpoints dialog box.

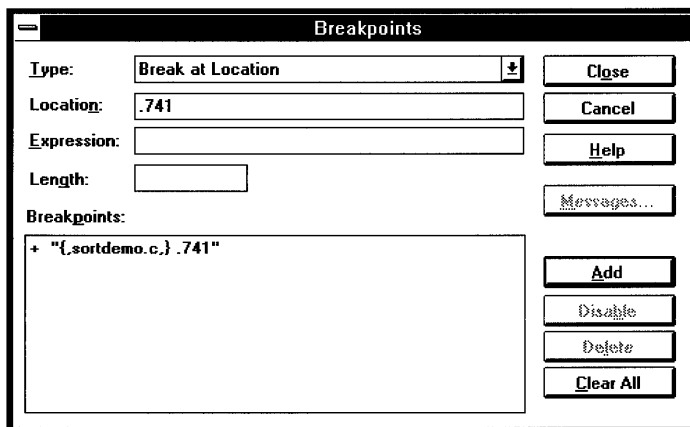


Figure 11.4 The Breakpoints Dialog Box

You can choose six types of breakpoints from the Type list box in the Breakpoints dialog box. These let you determine where and when the program will pause execution. Breakpoint types are:

#### Break at Location

This is the simplest type of breakpoint and is the default type used. If you have the insertion point on a line of code in a source file, this line automatically appears in the Location text box, making it easy to simply choose Add to add this breakpoint. When your program's execution reaches the breakpoint location, the program stops temporarily and you can use other debugging features. See the next section, "Breakpoints Involving Location," for information on proper location values.

#### Break at Location if Expression is True

You specify a location and an expression. Whenever execution reaches that location, the debugger checks the expression. If the expression is true (nonzero), the breakpoint is taken.

#### Break at Location if Expression has Changed

You specify a location and an expression that represents a variable or a portion of memory. Type a variable or memory address in the Expression text box. If the value of any byte has changed since the last time the debugger checked, the breakpoint is taken. See "Breakpoints Involving Expressions" on the next page for more information.

#### Break when Expression is True

This breakpoint is taken whenever the expression becomes true. The debugger evaluates the expression after every line or every instruction, instead of only at a certain location. As a result, this type of breakpoint can greatly slow your program's execution.

#### Break when Expression has Changed

The debugger checks the variable or range of memory as each line or each instruction is executed. You can also specify a range of memory by typing the starting address in the Expression text box and the length in the Length text box. This type of breakpoint can also slow your program's execution. See "Breakpoints Involving Expressions" on the next page for more information.

#### Break at WndProc if Message is Received

You specify a Windows callback function in the WndProc text box and use the Messages dialog box to select either a single message or one or more classes of messages on which to break. When a targeted message is received, the program's execution is paused at the specified WndProc. See "Breakpoints on Messages" on the next page for more information.

### Breakpoints Involving Location

You can enter a location directly into the Location text box by typing a location and choosing the Add button. The location can be entered either as a line number (with an optional filename) or as an address (in either hexadecimal or decimal notation).

Registers can also be used to form the address. Use a period (.) to indicate a line number and an exclamation point (!) to separate a filename from a line number. The following table shows the various forms of syntax for specifying location:

<b>Format</b>	<b>Example</b>	<b>Sets breakpoint at:</b>
<i>Filename!.linenumber</i>	MyApp.cpp!.35	Line 35 in MYAPP.CPP
<i>.linenumber</i>	.35	Line 35 in the active source file
<i>Segment:Offset</i>	0x2717:0x1222	Specified segment and offset
<i>Offset</i>	0x1222	Offset in code segment (CS)
<i>Segment:Offset</i>	CS:0x1222	Offset in CS
<i>Segment:Offset</i>	CS:IP	Instruction pointer offset to CS
<i>Offset</i>	IP	Instruction pointer offset to CS

### Breakpoints Involving Expressions

Four of the breakpoint types evaluate whether an expression is true or has changed. Depending on the breakpoint type, when the expression evaluates correctly, the program pauses either immediately or at the specified location. To use one of these types, you must first specify the desired expression in the Expression text box.

To detect when an expression has changed, you must enter a variable with a memory location (an l-value) in the Expression text box when you set the breakpoint. For variables that are not pointers, the value of the Length text box should normally be left as 1, since the length is calculated as the size of the variable multiplied by the number in the Length text box.

When you specify the name of a pointer in the Expression text box (such as `Ptr`, where `Ptr` is defined as `int *Ptr`), the breakpoint is set only when the address of the pointer changes. To set a breakpoint when the memory pointed to by the pointer changes, dereference the pointer in the Expression text box (for example, `*Ptr`), and then specify the number of bytes in memory to examine in the Length text box.

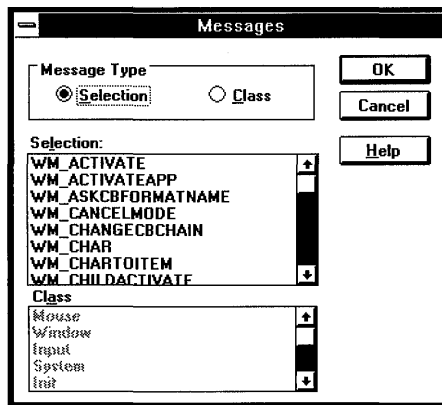
Memory ranges can be set by typing a starting address in hexadecimal or decimal notation in the Expression text box and the number of bytes in the Length text box. Registers can be used for addresses.

### Breakpoints on Messages

The breakpoint type “Break at WndProc if Message is Received” lets you set a breakpoint that tests messages received by any exported Windows callback function (window procedure). You can select whether to break on a specific message, or on any message from a class of messages or from several classes of messages.

► **To set a breakpoint on a message:**

1. From the Debug menu, select Breakpoints.  
The Breakpoints dialog box appears.
2. In the Type drop-down list box, select “Break at WndProc if Message is Received.”
3. In the WndProc text box, specify the name of the Windows callback function you want tested for messages.  
If you are setting a breakpoint during a debug session, the drop-down list contains all the exported functions in your project determined by the debugger to be Windows callback functions; otherwise, it is blank.
4. Choose the Messages button to open the Messages dialog box (see Figure 11.5).



**Figure 11.5 The Messages Dialog Box**

5. In the Message Type group, select the Selection option button.
6. In the Selection list box, select the message.
7. Choose OK to close the Messages dialog box.
8. Choose Close to close the Breakpoints dialog box.

You can follow the same basic procedure, with slight modification, to set a breakpoint on one or more classes of messages.

► **To set a breakpoint on any message in one or more classes:**

1. From the Debug menu, select Breakpoints.
2. In the Type drop-down list box, select “Break at WndProc if Message is Received.”
3. In the WndProc text box, specify the name of the exported Windows function.
4. Choose the Messages button to open the Messages dialog box.

5. In the Message Type group, select the Class option button.
6. In the Class list box, select one or more message classes.
7. Choose OK to close the Messages dialog box.
8. Choose Close to close the Breakpoints dialog box.

If you set a breakpoint using either of these two procedures, the program will pause execution at the specified exported Windows function (WndProc) when it receives a message matching the qualifications.

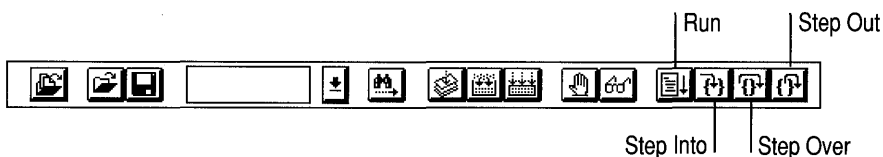
---

**Note** If you are debugging a Microsoft Foundation Class Library program, setting a breakpoint on a message may trace into the Microsoft Foundation Class Library source code. The debugger then requires the path to the Microsoft Foundation Class directory in order to open the file containing the breakpoint. If you do not have the MFC Files Path set correctly in the Directories dialog box (accessed from the Options menu), the debugger prompts you for a path. The debugger requires only the base path (for example, C:\MSVC\MFC).

---

## Controlling Program Execution

Once a breakpoint is reached and the program stops, you can control program execution with commands on the Debug menu. Most of these commands have equivalent toolbar buttons. Table 11.2 lists the Debug Menu commands and their actions.



**Table 11.2** Debug Menu Commands

Debug Menu Command	Action
Go	Executes code from the current statement until a breakpoint is reached, a watchpoint expression becomes true, or the end of the program is reached. (Equivalent to the Run button on the toolbar.)
Restart	Resets execution to the first line of the program. It reloads the program into memory and discards the current values of all variables (breakpoints and watch expressions still apply). It automatically halts at the <b>main()</b> or <b>WinMain()</b> function. (No toolbar equivalent.)
Stop Debugging	Terminates the debugging session and returns to a normal editing session. (No toolbar equivalent.)

**Table 11.2** Debug Menu Commands (*continued*)

Debug Menu Command	Action
Step Into	Steps into a function when it is called and steps through all of the instructions in the function.
Step Over	Single-steps through instructions in the program. If this command is used when you reach a function call, the function is executed without stepping through the function instructions.
Step Out	Executes the program out of a function call and stops on the instruction immediately following the call to the function. This allows you to step into a function without having to step all the way through it.
Step to Cursor	Executes the program as far as the line that currently has the cursor. This is equivalent to setting a temporary breakpoint at the cursor location. (No toolbar equivalent.)

## Using the Watch Window

The Watch window allows you to enter variables you want to view or expressions you want to see evaluated as the program progresses. You can enter values or expressions into the Watch window at any time, but they are only evaluated while the program is in a debug session. The Watch window displays an error on watch variables and expressions until a debug session begins.

A watch expression can be any valid C or C++ expression. For example, the following are all valid watch expressions in C and C++:

```
count
count + 1
count + 1 == 5
```

For relational expressions, the Watch window shows 0 if the expression is false and 1 if the expression is true:

```
count + 1 == 5 = 0
count + 1 == 5 = 1
```

Cast operators can be used in the Watch window, even on user-defined types (such as LPSTR).

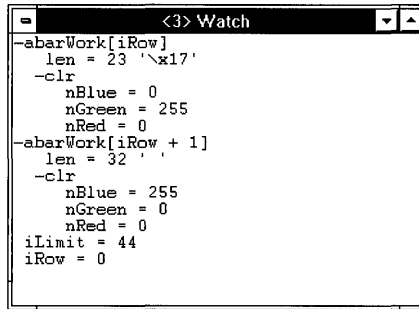
## Adding and Deleting Variables and Expressions

### ► To add a variable or expression to the Watch window:

1. While the program is paused between steps or at a breakpoint, move the insertion point to the first blank line in the Watch window (see Figure 11.6).

2. Type the variable name or expression.
3. Press ENTER.

The variable or expression is evaluated immediately. If the expression or variable cannot be evaluated, an error message appears in the window next to the variable or expression.



```
<3> Watch
-abarWork[iRow]
  len = 23 '\x17'
  -clr
    nBlue = 0
    nGreen = 255
    nRed = 0
-abarWork[iRow + 1]
  len = 32 ''
  +clr
    nBlue = 255
    nGreen = 0
    nRed = 0
iLimit = 44
iRow = 0
```

**Figure 11.6** The Watch Window

If you want, you can use the copy (CTRL+C) and paste (CTRL+V) commands to copy variable names from your program source files into the Watch window. You must always press ENTER to insert the variable or expression into the window.

You can also use the QuickWatch dialog box to add variables to the Watch window. This is described in “Using QuickWatch” on the next page.

- ▶ **To delete a variable or expression from the Watch window:**
  1. Move the insertion point to the line containing the variable or expression.
  2. Select the entire variable name or expression using either the mouse or SHIFT+LEFT ARROW and SHIFT+RIGHT ARROW.
  3. Press the DEL key.

To learn how to modify variables in the Watch window, see “Modifying a Variable” on page 193.

## Expanding and Collapsing Variables

In the Watch window, Locals window, and QuickWatch dialog box, variables that contain more than one element, such as arrays, structures, classes, or enumerated types, are displayed with either a + or – sign preceding them (see Figure 11.7).

The + symbol indicates that the variable contains elements and can be expanded. The – symbol indicates that the variable is fully expanded and can be collapsed.



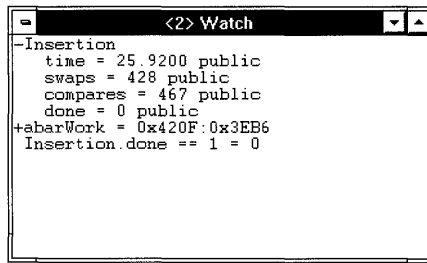


Figure 11.7 The Watch Window with Expanded and Collapsed Variables

► **To expand or collapse a variable:**

1. Move the insertion point to the line containing the variable.
2. Press ENTER.

Or double-click anywhere on the variable.

## Using QuickWatch

QuickWatch is a dialog box that gives you a fast way to view variables and expressions (see Figure 11.8). Unlike watch expressions, which remain in the Watch window, the values of QuickWatch variables and expressions appear only when you open the QuickWatch dialog box (although they can easily be added to the Watch window from the QuickWatch dialog box). QuickWatch is useful in exploratory debugging where you are checking a number of variables that are suspect.

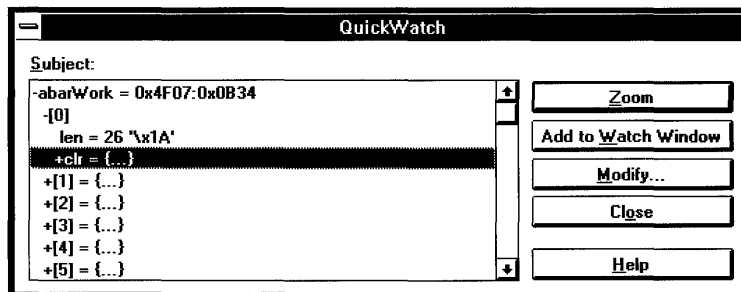


Figure 11.8 The QuickWatch Dialog Box

► **To view a variable's value or an expression's result using QuickWatch:**

1. Place the insertion point on a variable or expression in your code.
2. Press SHIFT+F9.

Or click the QuickWatch button on the toolbar.

► **To add a variable or expression to the Watch window from QuickWatch:**

1. Place the insertion point on a variable or expression in your code.
2. Press SHIFT+F9.

Or click the QuickWatch button on the toolbar.

3. Choose Add to Watch Window.

The Watch window opens if it is not already open.

The QuickWatch dialog box also provides controls that allow you to expand arrays or structures to see all their elements and to modify variables while the program is at a breakpoint.

The Zoom button expands or contracts an array or structure. The button is unavailable if the currently selected item cannot be expanded.

The Modify button opens the Modify Variable dialog box with the currently selected item as the default.

Note that the QuickWatch dialog box is active only when your program is running in the debugger.

## Modifying a Variable

While the program is paused at a breakpoint or between steps, you can change the value of any variable in your program. This gives you the flexibility to try out changes and see their results in real time or to recover from some logic error and continue. You can modify any variable in either the Watch window or the Locals window directly. Or you can modify any variable in your program by using the QuickWatch dialog box.

► **To modify the value of a variable in the Watch window or Locals window:**

1. In the Watch window or Locals window, place the insertion point at the end of the variable value and use the BACKSPACE key to delete the value.
2. Type the new value.
3. Press ENTER.

You can also modify expressions using this same procedure.

The Modify Variable dialog box (see Figure 11.9) is accessed from the QuickWatch dialog box. It allows you to modify the variable in the QuickWatch dialog box or any variable in the program.

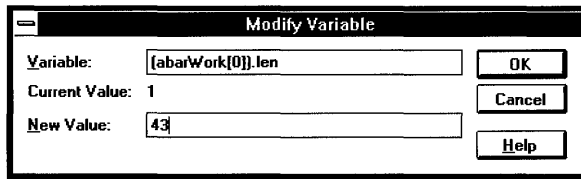


Figure 11.9 The Modify Variable Dialog Box

► **To modify the value of a variable using the Modify Variable dialog box:**

1. Place the insertion point on a variable in your code.
2. Click the QuickWatch button on the toolbar.

Or press SHIFT+F9.

The QuickWatch dialog box appears and displays the variable and its current value.

3. In the QuickWatch dialog box, choose the Modify button.

Or press CTRL+F9.

The Modify Variable dialog box appears with the variable name in the Variable text box.

4. Type a new value in the New Value text box.
5. Choose OK to close the Modify Variable dialog box.
6. Choose Close to close the QuickWatch dialog box.

To change the value of a structure or array, modify the individual fields or elements. You cannot change an entire array or structure all at once.

## Using Show Call Stack

During a debug session, you can view all the functions that have been called but have not returned. The Show Call Stack command on the Debug menu opens a dialog box that lists the function calls that led to the current statement (see Figure 11.10). If the Show Function Parameters check box is enabled, each call is shown with the arguments passed to it. The most recently executed function is listed first.

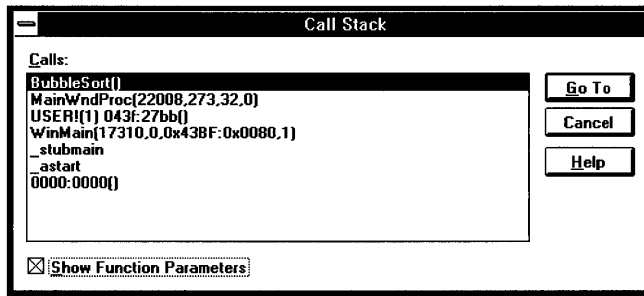


Figure 11.10 The Call Stack Dialog Box

► **To observe the behavior of a function call:**

1. Place the insertion point at the desired location in the function.
2. From the Debug menu, choose the Continue to Cursor command to execute your program to the location of the insertion point.

The Locals window is updated automatically to show the local variables for the function or procedure. Expressions and variables in the Watch window are also reevaluated in the call context.

3. From the Debug menu, choose Show Call Stack.

While the Call Stack dialog box is open, you can select any function shown in the function list, and the function's code will be displayed in the source window.

## Using the Registers Window

The Registers window (see Figure 11.11) displays the names and current values of the native CPU registers and flags. You can change the value of any register or flag directly in the Registers window while the program is being debugged.

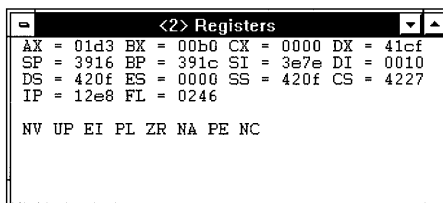


Figure 11.11 The Registers Window

► **To change the value of any register in the Registers window:**

1. Use the TAB key or mouse to move the insertion point to the beginning of a register value.
2. Enter the new value by overtyping the current value.

---

**Important** Changing register values may affect the next execution command. Be especially careful with IP, BP, and the segment registers.

---

The flag values displayed in the Registers window are:

Flag	Set Symbol	Clear Symbol
Overflow	OV	NV
Direction	DN	UP
Interrupt	EI	DI
Sign	NG	PL
Zero	ZR	NZ
Auxiliary carry	AC	NA
Parity	PE	PO
Carry	CY	NC

► **To set or clear a flag in the Registers window:**

1. Use the TAB key or mouse to move the insertion point to the flag.
2. Press the SPACEBAR to toggle the flag.

The Registers window does not show 32-bit registers or p-code registers. Use CodeView if you need these advanced features.

## Other Debugging Features

The Visual Workbench debugger also has debugging commands and options for switching between Hard and Soft mode debugging, for viewing numbers in hexadecimal or decimal format, and for viewing source code as mixed source and assembly listings.

## Hard/Soft Mode Debugging

Hard/Soft debug mode is set in the Debug dialog box, accessed from the Options menu. This option selects whether or not the debugger traps messages from the system queue when in break mode. In Hard mode, all keyboard and mouse input goes to the debugger. This effectively disables all other applications. The debugger automatically switches in and out of Hard mode when necessary.

## What is Hard Mode?

Applications written for Windows typically run in soft mode. Hard mode appeared with Windows version 3.1 as a response to requests for specific debugging requirements.

Hard mode is a “task exclusive” mode that any Windows-based application can switch to using the appropriate API. While in hard mode, Windows directs all input from either the mouse or the keyboard to the application that installed itself in Hard mode.

## How Does Hard Mode Affect the Debugger?

With Windows-hosted debuggers, there is the potential that the debugger might destabilize the Windows system by interacting with it. For this reason, the Visual Workbench integrated debugger switches by itself to Hard mode every time it breaks debugging execution (as a result of a step command, stopping at a break-point, or encountering an exception, for example) while the system is in a critical (non-interruptable) processing stage. This can occur while debugging menu events, system modal windows, or intertask messages, among other situations.

## Requirements and Restrictions

Although Soft mode is the default mode for the integrated debugger, you can specifically request that the debugger switch to Hard mode when in break mode. This might be useful to debug time-critical events such as intertask DDE (dynamic data exchange) transactions.

Because Hard mode is, by design, a “task exclusive” mode, no other application can be switched to or started while this mode is operating. As a reminder, the title bar displays [break - hard mode] instead of [break] every time the debugger switches to this mode.

While in Hard mode, no other application can be started. In Visual Workbench, this means you do not have access to the following functionality:

- Help
- App Studio
- Running tools on the Tools menu
- Printing

Also, you cannot minimize Visual Workbench since the Windows default is to switch to another task when this happens. Another restriction of Hard mode is that Windows will not repaint anything on the screen except the regions that belong to the debugger frame and its child windows.

Considering the restrictions inherent in the use of Hard mode debugging, you should consider using this mode only when necessary.

## Debug Display Options

The debugging display options are listed in Table 11.3. The menus or dialog boxes where you find these features are listed in parentheses following the action.

**Table 11.3 Additional Debugging Features**

<b>Feature</b>	<b>Action</b>
Hexadecimal Display	Toggles the format of all numbers displayed in the Locals and Watch windows, as well as in several dialog boxes. The default format is base 10. When this option is selected, the format is base 16. (Options menu, Debug dialog box)
Mixed Source/Asm	Toggles a source display that includes assembly code integrated with source code. When this command is checked, the debugger single-steps assembly lines not source lines and breakpoints can be set in assembly code. (View menu)

See the online reference for complete information about these commands.

# Customizing Visual Workbench

Many parts of Visual Workbench can be customized to suit your programming needs. This chapter discusses how you can:

- Add commands and applications to the Tools menu.
- Modify display colors.
- Change the font type and size in source windows.

Font and window information are saved and loaded as part of the workspace information. For information on naming and saving any custom workspaces you create, see “Using a Workspace” on page 104 in Chapter 8.

You can also customize many features of the Visual Workbench editor using the Editor dialog box, opened from the Options menu. For information on customizing the editor, see Chapter 7, “Using the Editor.”

## Modifying the Tools Menu

The Tools menu is a convenient place from which to run frequently used MS-DOS–based and Windows-based tools while you are in Visual Workbench. Once you’ve added an application to the Tools menu, you can run it from the menu. You use the Tools command on the Options menu to add, delete, and edit Tools menu items.

## Adding Commands to the Tools Menu

You can add up to eight commands to the Tools menu, including MS-DOS commands (with .EXE or .COM files), and MS-DOS–based and Windows-based applications.

To familiarize yourself with the steps in adding a program to the Tools menu, use the Notepad accessory that comes with Windows.



► To add a program to the Tools menu and then run it:

1. From the Options menu, choose Tools.

The Tools dialog box appears (see Figure 12.1).

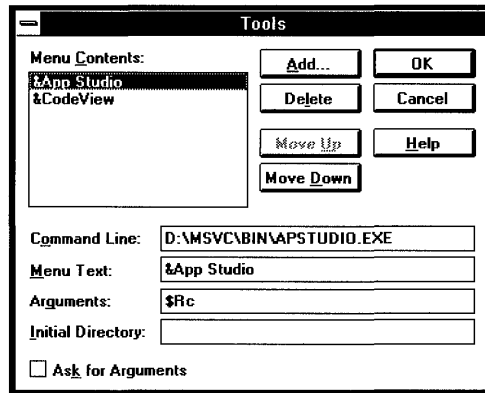


Figure 12.1 The Tools Dialog Box

2. Choose Add to open the Add Tool dialog box (see Figure 12.2).

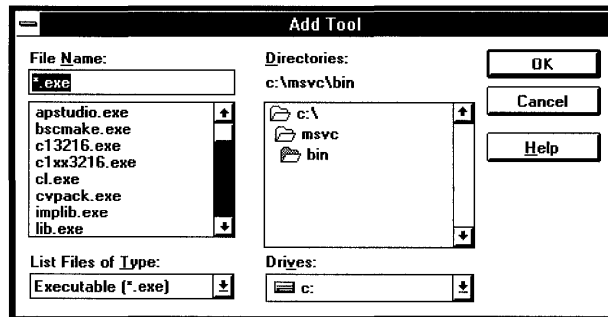


Figure 12.2 The Add Tool Dialog Box

3. Select the directory in which Windows is installed.
4. Select NOTEPAD.EXE from the list of filenames and choose OK.

The Tools dialog box reappears.

You can change the default menu name by editing the Menu Text text box. You can also add arguments to be passed to the program by typing them in the Arguments text box (see “Using Argument Macros” on page 202), or set the initial directory for your program by typing it in the Initial Directory text box.

---

**Note** If the program you are adding to the Tools menu has a .PIF file, the startup directory specified by the .PIF file overrides the directory specified in the Initial Directory text box.

---

5. Choose OK.

The name of the program now appears on the Tools menu. To run the program, choose it from the menu.

## Editing a Tools Menu Command

► **To edit a Tools menu command:**

1. From the Options menu, choose Tools.
2. Under Menu Contents, select the item you want to edit.
3. Perform one or more of the following actions:
  - To move the selected command up one position in the menu, choose Move Up.
  - To move the selected command down one position, choose Move Down.
  - To change the menu title, the command line (tool path), command-line arguments, or the initial directory, type the new information in the appropriate text box.

If you want to specify a letter in the menu title as an access key (a menu accelerator key), precede that letter in the Menu Text text box with an ampersand (&). The first letter in the title is the keyboard access key by default.

If you want to be prompted for command-line arguments each time you run the tool, select the Ask for Arguments check box.

4. Choose OK.

► **To delete a command from the Tools menu:**

1. From the Options menu, choose Tools.
2. Under Menu Contents, select the command you want to delete from the Menu Contents list.
3. Choose Delete to remove the program from the list.
4. Choose OK.

## Tips For Using MS-DOS Tools

If you have particular MS-DOS tools or programs you like to use, here are a couple of tips to make it easier to integrate these tools with Visual Workbench.

To keep an MS-DOS window open with the output of a command-line tool after the tool has been run from the Tools menu, use the Windows PIF Editor to edit the file `_DEFAULT.PIF` and clear the Close Window on Exit check box. You will then need to close the MS-DOS window using the Control-menu box whenever you run the tool, but you will be able to view the tool's output when it has finished.

To use any of the MS-DOS commands (such as `DIR`) or command-line operations (such as piping) that don't have an executable file, type the complete path to `COMMAND.COM` in the Command Line text box and type `/C` followed by the commands you want to invoke in the Arguments text box. For example, the following parameters in the Tools dialog box cause an MS-DOS window to open with a paged directory listing of the project directory when the `Dir` command on the Tools menu is selected:

Text Box	Entry
Command Line	<code>C:\DOS\COMMAND.COM</code>
Menu Text	<code>Dir</code>
Arguments	<code>/C DIR   MORE</code>
Initial Directory	<code>\$ProjDir</code>

## Using Argument Macros

You can specify arguments for any program that you add to the Tools menu by entering the arguments in the Arguments text box. To help you integrate your tools with the current status of the Visual Workbench environment, Visual Workbench provides a set of 10 argument macros (see Table 12.1).

**Table 12.1 Visual Workbench Argument Macros**

Macro Name	Expands to a String Containing
<code>\$File</code>	The complete filename of the current source (defined as <i>drive+path+filename</i> ), blank if a nonsource window is active.
<code>\$FileName</code>	The filename of the current source (defined as <i>filename</i> ), blank if a nonsource window is active.
<code>\$FileDir</code>	The directory of the current source (defined as <i>drive+path</i> ), blank if a nonsource window is active.
<code>\$Proj</code>	The current project base name (defined as <i>filename</i> ), blank if no project is currently open.
<code>\$ProjDir</code>	The directory of the current project (defined as <i>drive+path</i> ), blank if no project is currently open.
<code>\$Line</code>	The current cursor line position within the active window.
<code>\$Col</code>	The current cursor column position within the active window.
<code>\$Dir</code>	The current working directory (defined as <i>drive+path</i> ).

**Table 12.1 Visual Workbench Argument Macros** (*continued*)

Macro Name	Expands to a String Containing
<b>\$Target</b>	The current project target name (defined as <i>drive+path+filename</i> ).
<b>\$SRC</b>	A resource file (*.RC). For Visual Workbench projects, this is the first resource file in the project list. For external projects, it is <b>\$Target.RC</b> . If there is no resource file or there is no active project, <b>\$SRC</b> is blank.

Macro recognition is case insensitive. All path macros end in a backslash (\).

To use a macro as an argument, type the macro name in the Arguments text box. Or, for macros that expand to a directory, you can type the macro name in the Initial Directory box. As an example, the following procedure demonstrates how to add the **\$File** argument macro to the Windows Notepad accessory (installed in a previous procedure).

► **To add the \$File macro to an installed tool and then run it:**

1. From the Options menu, choose Tools.
2. Under Menu Contents, select the command you want to edit.  
In this case, select the Notepad accessory installed earlier.
3. In the Arguments text box, type `$File`.
4. Choose OK to close the Tools dialog box.
5. Open any source file or make an open source file active by clicking in it.
6. From the Tools menu, choose Notepad.

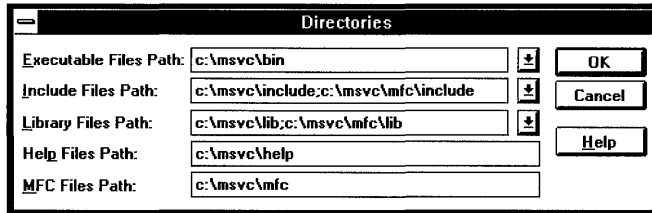
The Windows Notepad editor opens with the active Visual Workbench source file as its text file.

## Setting Directories

When Visual C++ is installed, the Setup program determines the correct directory paths for several file types and updates the Directories dialog box with these paths. The file types are:

- Build utilities (executable files)
- Include files
- Libraries
- Microsoft Foundation Class Library source files
- Help files

The Directories dialog box, accessed by choosing Directories from the Options menu (see Figure 12.3), lets you edit the directory paths where Visual Workbench looks for each of the file types. The directories for the top three file types in the list (executable files, include files, and libraries) are drop-down lists that can contain up to four different paths, which makes it easy to quickly change build environments.



**Figure 12.3 The Directories Dialog Box**

The Directories dialog box contains the following drop-down list boxes:

#### Executable Files Path

Specifies where the build utilities, such as NMAKE, LINK, and BSCMAKE, reside. The build uses MS-DOS tools and requires an MS-DOS path.

#### Include Files Path

Specifies where the compiler should look for include files surrounded by angle brackets (< and >) (for example, `include <stdio.h>`).

#### Library Files Path

Specifies where the linker should look for libraries to resolve external references.

In addition, the Directories dialog box contains the following text boxes:

#### Help Files Path

Specifies where Help files accessed from Visual Workbench are located.

#### MFC Files Path

Specifies where the base directory for the Microsoft Foundation Class Library is located. This is needed when you use the debugger to trace into class library source code or when you use the browser to jump to definitions or references in the class library source code.

Directory information is stored in the MSVC.INI file. When you first install Visual Workbench (or if you delete the MSVC.INI file and then run Visual Workbench), the MS-DOS environment variables PATH, INCLUDE, and LIB are used to build the corresponding directory paths. These environment directories are appended to the Visual C++ directories in which you install Visual Workbench. Thereafter,

Visual Workbench uses the directory paths in the Directories dialog box, regardless of your current MS-DOS environment variables.

## Changing Syntax Coloring

Using different colors for various language elements such as functions and variables gives you immediate visual clues about the structure of your source code. You can change the default colors of these elements as well as the color of other text in the development environment such as reserved words, breakpoints, errors, and tags. You can also turn off syntax coloring for all source files. These changes are global and affect all source files with extensions recognized by Visual Workbench. To make global syntax coloring changes, choose Color from the Options menu.

Visual Workbench performs syntax coloring based on file extensions. You can override Visual Workbench's default coloring for a file by choosing Syntax Coloring from the View menu. This allows you to specify C, C++, or no syntax coloring for any individual file.

## Making Global Display Changes

To familiarize yourself with the steps in changing screen colors, change the color of comments in the source window to black text on a light-blue background.

► **To change the colors in the source window:**

1. From the Options menu, choose Color.

The Color dialog box appears (see Figure 12.4).

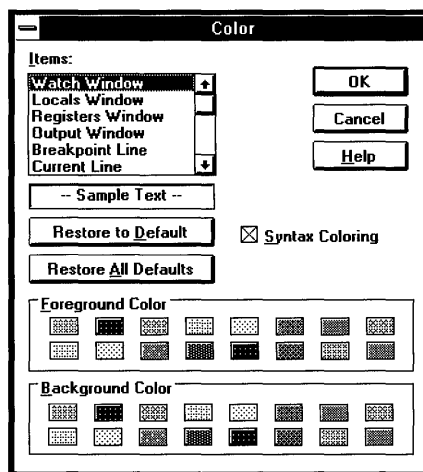


Figure 12.4 The Color Dialog Box

2. From the Items list, select the item you want to change.

For this example, select Comment.

3. In the Foreground Color and Background Color groups, select the color squares to represent the item.

For example, select the light-blue square in the Background Color group and the black square in the Foreground Color group. The new color combination is shown in the Sample Text box.

4. Choose OK to apply the change to the source window.

You can change several items in the Items list before choosing OK.

► **To change a source window item back to its original colors:**

1. From the Options menu, choose Color.
2. From the Items list, select the source item to be restored.

For example, select Comment.

3. Choose Restore to Default.
4. Choose OK.

Display items such as keywords and identifiers can be changed back to their default colors without changing the foreground and background colors of the source text.

To restore all items to their default values, choose Restore All Defaults.

## Source File Syntax Coloring

For any source file in Visual Workbench, you can specify which syntax coloring to apply (C or C++), or whether to turn off syntax coloring altogether. This is useful if you have C or C++ code in a header file with a filename extension other than .H, .HPP, or .HXX, or if you have C or C++ source code with filename extensions that Visual Workbench doesn't recognize. (You should be cautious when using nonstandard filename extensions, however, since the filename extensions .CPP, .CXX, and .C are used by Visual Workbench to determine whether the C++ or C compiler is used during the build.)

► **To change syntax coloring in an individual source file:**

1. Click in the source file window or use the Window menu to make the source window active.

If there are multiple windows open on the source file, select one of them. Syntax coloring changes will appear in all windows opened on the source file.

2. From the View menu, select Syntax Coloring.

A cascading menu appears on the right with three choices: C, C++, and None.

3. Choose C or C++ to determine syntax coloring for that source file.  
Or choose None to turn syntax coloring off.

---

**Note** Global syntax coloring must be enabled before you can use the Syntax Coloring command on the View menu on any specific file. To enable global syntax coloring, choose Color from the Options menu and then select the Syntax Coloring check box in the Color dialog box.

---

## Setting Font Type and Size

You can specify which font type and size appear in a Visual Workbench source window. You can choose any font type and size found in your setup of Windows. You can also determine what will be the default font for any new window.

► **To set the font type, style, and size:**

1. From the Options menu, choose Font.

The Font dialog box appears (see Figure 12.5).

2. From the Font list box, select the font you want to apply.

The sample text in the Sample box changes to the font you selected.

3. Optionally, select the font style from the Font Style list box.

The sample text in the Sample box changes to the font style you selected.

4. From the Size list box, select the font size.

The sample text in the Sample box changes to the font size you selected.

5. Choose OK.

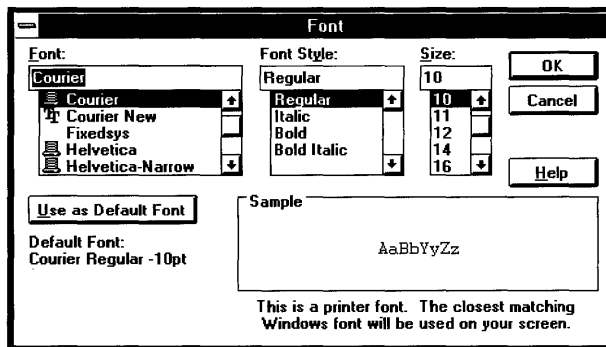


Figure 12.5 The Font Dialog Box



► **To set the default font for new source windows:**

1. Choose a font, style, and size as just described.
2. Choose the Use as Default Font button.

The currently selected font, style, and size appear in the Default Font description.

3. Choose OK.

The default font is automatically applied to any new or opened window. The default font information is stored with other Visual Workbench information and is persistent between sessions.

Text within the source window can be only one font and size. Multiple fonts cannot be displayed in the same source window. However, each source window can contain a different font and size, even when source windows with different fonts are attached to the same source file.

# Using Visual Workbench with Other Visual C++ Tools

To develop a Visual C++ application, you use four Visual C++ tools. Two of these, Visual Workbench and App Studio, are major applications. The other two are specialized “wizards” installed as menu items in these applications:

- AppWizard is used to generate Visual C++ application starter files. AppWizard is accessed from Visual Workbench.
- ClassWizard is used to create classes, map messages to class-member functions, and map controls to member variables. ClassWizard is accessed from either Visual Workbench or App Studio.

This chapter discusses how Visual Workbench fits into the process of developing a Visual C++ application that uses the Microsoft Foundation classes—that is, how you see the entire process from the viewpoint of Visual Workbench. In this context, it focuses on programming activities you encounter while running Visual Workbench, such as opening App Studio and running AppWizard and ClassWizard.

Since AppWizard is only run from Visual Workbench, it is described in detail. Since ClassWizard can be run from either App Studio or Visual Workbench, the activities likely to be performed from a Visual Workbench invocation of ClassWizard are described here. The primary reference information for ClassWizard can be found in Chapter 9, “Using ClassWizard,” in the *App Studio User’s Guide*.

For a global view of how all the tools work together, see Chapter 4 in this manual, “Developing a Microsoft Visual C++ Application.”

## Using AppWizard

AppWizard is used to generate a set of starter files for a Visual C++ application that uses the Microsoft Foundation classes. This set includes all the files required to build a Windows-based application, including source and header files, resource files, a module-definition file, a project file, and so on. AppWizard must be used

first in the development process, primarily so that you have starter files, but also so that your files are compatible with ClassWizard.

Building the resulting project produces a shell application with a wealth of built-in functionality. For example, you get built-in handling of the File menu's Open and Save As commands, an About dialog box, and an icon. By selecting options in AppWizard, you can add a toolbar and status bar, a Printer Setup dialog box, a fully implemented Print Preview, a Print command and toolbar button, context-sensitive Help, support for custom VBX controls (custom controls compatible with Visual Basic), and support for object linking and embedding (OLE).

This section describes the MFC AppWizard dialog box and the Options and Classes dialog boxes that are accessed from it. For tutorial information on using AppWizard to create a Visual C++ application, see Chapter 2, "Creating a New Application with AppWizard," in the *Class Libraries User's Guide*.

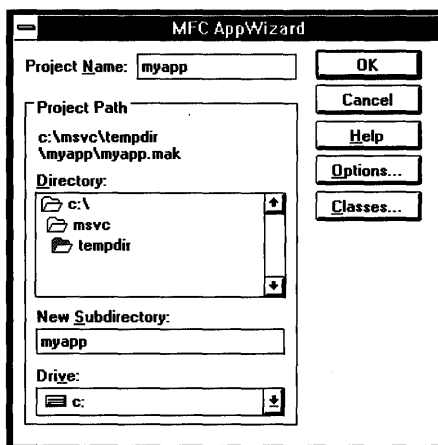
## Opening and Closing AppWizard

To open AppWizard, choose the AppWizard command from the Visual Workbench Project menu. After using AppWizard to specify your application's name, directory, options, and classes—as described in the following sections—choose the OK button to generate the starter files.

When AppWizard is finished, it creates a Visual Workbench project that allows you to immediately build and run the application. A message box appears to tell you AppWizard has generated the files. Choose OK in the message box to close AppWizard and to open the newly created project in Visual Workbench.

## Specifying the Project Name and Location

The MFC AppWizard dialog box (see Figure 13.1) allows you to select the project name and the subdirectory name and root path where the project files will reside. The project name is given the extension .MAK and appears in the complete path at the top of the Project Path group. It is also used as a base to create class names and source and header filenames.



**Figure 13.1** The MFC AppWizard Dialog Box

► **To select the project name and subdirectory name:**

1. Open the MFC AppWizard dialog box, if it isn't already open, by choosing AppWizard from the Project menu.
2. Type a name in the Project Name box.

Notice that the name you type is automatically entered as a directory name in the New Subdirectory box. It is also entered in the project file path above the Directory box.

The maximum number of valid characters in the Project Name box and the New Subdirectory box is eight.

3. If you do not want the subdirectory name to be the same as the project name, you can type a different name in the New Subdirectory box, or you can leave the subdirectory name blank.

See the next procedure if you want to change the directory path that precedes the subdirectory.

You must select the drive and directory path to the subdirectory that will contain the generated files. To do this, you use the Directory box and the Drive box. These two controls work the same as similar controls in the Open File and Save As dialog boxes.

► **To select a drive and directory path for the application's subdirectory:**

1. In the Drive drop-down list box, select the drive you want.
2. In the Directory box, select the directory path by double-clicking the directory icons.

The drive letter and directory path you select are reflected in the project path. You can omit the subdirectory. The format for the entire path is:

*drive:directorypath[\\subdirectory]projectname.MAK*

## Selecting Options

AppWizard can add the basic skeleton code to support a comprehensive set of application options. Obviously, the more functionality your application supports, the larger it will be. However, if your application needs to support some specific functionality—such as providing context-sensitive hooks to Help, for instance—AppWizard makes the addition much easier.

To select the options you want your application to support, choose the Options button in the MFC AppWizard dialog box and then enable or disable the various check boxes in the resulting Options dialog box (see Figure 13.2). When you have finished selecting options, close the Options dialog box by choosing OK.

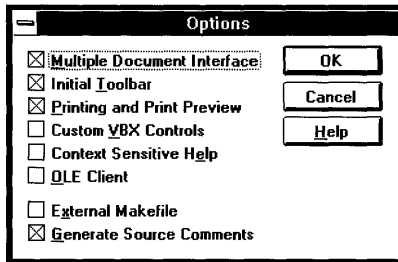


Figure 13.2 AppWizard's Options Dialog Box

You can select these options in the Options dialog box:

### Multiple Document Interface

This option lets you select the type of project you want to create. There are two project types available. When this option is checked, AppWizard creates an MDI application. When this option is cleared (not checked), it creates an SDI application.

- An SDI (single document interface) application allows a user to work with just one document at a time. The Windows Notepad is an example of an SDI application.
- An MDI (multiple document interface) application allows a user to open multiple documents, each with its own window. The Windows File Manager is an example of an MDI application.

### Initial Toolbar

AppWizard generates code for a toolbar and a status bar. The toolbar contains buttons for creating a new document, opening and saving document files, cutting, copying and pasting, printing, displaying the About dialog box, and invoking Help. The status bar contains automatic indicators for the keyboard's CAPS LOCK, NUM LOCK, and SCROLL LOCK keys and a message line that displays help strings for menu commands and toolbar buttons. Enabling this option also adds menu commands to display or hide the toolbar and status bar.

### Printing and Print Preview

AppWizard generates the code to handle print, print setup, and print preview commands by calling member functions in the **CView** class from the Microsoft Foundation Class Library. It also adds commands for these functions to the application's menu.

### Custom VBX Controls

AppWizard enables the use of custom VBX controls (custom controls compatible with Visual Basic). This is accomplished by a single function, **EnableVBX()**, which includes the run-time code required to use VBX controls. You use App Studio to incorporate custom VBX controls in your user interface. To learn how to write the supporting code, see Chapter 17 in the *Class Library User's Guide*.

### Context Sensitive Help

AppWizard generates a set of help files (see page 217) that are used to provide context-sensitive help. Help support requires the help compiler, which is provided with Visual C++ Professional Edition.

### OLE Client

AppWizard generates application support code for an OLE (object linking and embedding) client application. This allows OLE-linked and OLE-embedded objects to be placed in your application's documents. When you select this option, the document class is derived from **COleClientDoc** instead of **CDocument**. Selecting this option also enables the standard OLE resources and adds extra OLE commands to the application's menu bar. For more information about how to write code for OLE, see Chapter 18 in the *Class Library User's Guide*.

### External Makefile

By default, AppWizard generates a project file that is compatible with Visual Workbench (and NMAKE). Select this option if you want AppWizard to generate an NMAKE makefile that can be directly edited but must be used as an external project from within Visual Workbench.

### Generate Source Comments

AppWizard generates and inserts comments in the source files that guide you in writing your program. This includes indicators where you need to add your own code. It is a good idea to enable this option.

## Modifying Classes

AppWizard names your application's classes and files by using the project name you specify in the Project Name box. The Classes dialog box (see Figure 13.3), accessed with the Classes button in the MFC AppWizard dialog box, lets you modify components of each of the four classes that are created by AppWizard. Some classes allow more modification of options than others.

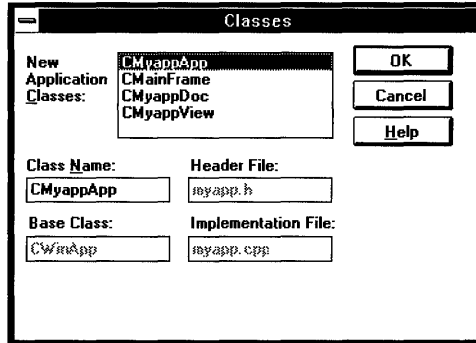


Figure 13.3 AppWizard's Classes Dialog Box

### New Application Classes

Three of the four classes available in the New Application Classes list box are derived from your project name. In each of the following class names, the placeholder *Prjname* indicates the project name you have specified.

#### **CPrjnameApp**

This is the main application class. The Base Class, Header File, and Implementation File text boxes are all dimmed, indicating that you cannot change any of these options, since these must reside in the main project. You can, however, change the name of the class.

#### **CMainFrame**

This is the class that handles the window frame, which contains the toolbar and status bar. You cannot change the base class, since this is determined by the project type you choose (MDI or SDI). However, you can change the name of the class or the names of the header and source files associated with the window frame code by typing new names in the Header File and Implementation File text boxes.

#### **CPrjnameDoc**

This is the class that contains the document data and handles saving it to disk and recalling it from disk, among other things. You cannot change the base class, which

will be **CDocument** (or **COleClientDoc** if OLE support is selected), but you can change the names of the class, the header file, and the source (implementation) file.

A **CPrjnameDoc** class is associated with a document type and has the built-in functionality to automatically serialize (save and load) the document to and from disk in response to the Save As and Open commands on the File menu. You can change the name that is used wherever the application's native document type is referred to by typing a new name (up to six characters) in the Doc Type Name box. Examples of where the document type name is used include the default filename (*CPrjname.DOC*), the Windows shell registration name, the title bar of a document window, and constants referred to in the code created by AppWizard. This name is also used in the File Manager shell and OLE server options when a file type is needed.

You can also choose a filename extension for the document by typing up to three characters in the File Extension box. The filename extension is added to document files saved from the application and appears as a file filter in the application's Open File and Save As dialog boxes.

## **CPrjnameView**

This class is used to display document data inside a window. You can change the name of the class, the header file, or the source file; however, you cannot change the name of the base class.

## **AppWizard-Generated Files**

AppWizard always generates a basic list of files, regardless of which options you choose. The Context Sensitive Help and Initial Toolbar options result in additional files. This section first describes the core files common to all AppWizard-generated applications and then describes files that are added when you select toolbar and Help support.

AppWizard uses the name that you specify in the Project Name box to derive names for most of its files and classes. In the following descriptions, where the full project name is used in the filename, *PRJNAME* is used as a placeholder for the name you specify. For some filenames, the project name is truncated to five characters. In those cases, *PRJNA* is used as a placeholder for the truncated project name.

---

**Note** The name substitutions indicated in these filenames may not apply if you have used the Classes dialog box to alter any of these names.

---

## **Standard AppWizard Files**

This section describes the various files generated by AppWizard and categorizes them by function. In the project directory, you'll also find a file named



README.TXT, which describes each of the files created by AppWizard using the actual filenames created by AppWizard for your specific project.

## Project and Makefiles

### *PRJNAME.MAK*

This is the project file for Visual Workbench. It is also an NMAKE-compatible file. If you select the External Makefile option in AppWizard's Options dialog box, this can only be used as an external project from within Visual Workbench.

### *PRJNAME.CLW*

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to generate and edit message maps and dialog data maps, and to generate prototype member functions.

## Resource and Module-Definition Files

### *PRJNAME.RC, RESOURCE.H*

This is the resource file for the project and its header file. The resource file contains the default menu definition and accelerator and string tables for a generic Visual C++ application. It also specifies a default About dialog box and an icon file (*RES\PRJNAME.ICO*). The resource file includes the file *STDAFX.RC* for standard Microsoft Foundation class resources. If toolbar support has been specified as an option, it also specifies the toolbar bitmap file (*RES\TOOLBAR.BMP*).

### *RES\PRJNAME.ICO*

This is the icon file for the generic Visual C++ application. This icon appears when the application is minimized.

### *PRJNAME.DEF*

This is the module-definition file for the application, which includes the name and description of the project as well as the size and type of the run-time heap and run-time stack. The heap and stack sizes are typical for this type of Microsoft Foundation Class Library application.

## Application Source and Header Files

### *MAINFRM.CPP, MAINFRM.H*

These files derive the `CMainFrame` class from either `CFrameWnd` (for SDI applications), or `CMDIFrameWnd` (for MDI applications). The `CMainFrame` class handles the creation of toolbar buttons and the status bar, if the Initial Toolbar check box is enabled in AppWizard's Options dialog box. The *MAINFRM.CPP* file also contains the array of object IDs for the default toolbar buttons provided with a Visual C++ application.

### *PRJNAME.H*

This is the main include file for the application. It contains all global symbols and `#include` directives for other header files.

### *PRJNAME.CPP*

This file is the main application source file. It creates one instance of the class `CPrjnameApp` (which is derived from `CWinApp`) and overrides the `InitInstance` member function.

`CPrjnameApp::InitInstance` does several things. It registers document templates, which serve as a connection between documents and templates, creates a main frame window, and creates an empty document (or opens a document if one is specified as a command-line argument to the application). It also enables support for custom VBX control and F1-Help, if these options have been chosen.

### *PRJNADOC.CPP, PRJNADOC.H*

These files derive and implement the document class, named `CPrjnameDoc`, and include skeleton member functions to initialize a document, serialize (save and load) a document, and implement debugging diagnostics. In an MDI application, you use `ClassWizard` to add more document classes.

### *PRJNAVW.CPP, PRJNAVW.H*

These files derive and implement the view class, named `CPrjnameView`, that is used to display and print the document data. The `CPrjnameView` class is derived from `CView` and has skeleton member functions to draw the view and implement debugging diagnostics. In an MDI application, you use `ClassWizard` to add more view classes. If you have enabled support for printing, message-map entries are added for print, print setup, and print preview command messages. These entries call the corresponding member functions in the `CView` class.

## Precompiled Header Files

### *STDAFX.CPP, STDAFX.H*

These files are used to build a precompiled header file *PRJNAME.PCH* and a precompiled types file *PRJNAME.PCT*.

## Files Added by Options

Most options use the standard files to implement their features. The exceptions are the Initial Toolbar and Context Sensitive Help options. Toolbar support adds just one file, *RES\TOOLBAR.BMP*. Help support provides a number of files and creates a new directory *\HLP* to contain most of them.

## Help Option

### *MAKEHELP.BAT*

This batch file can be used to create Help for your application.

### *PRJNAME.HPJ*

This is the Help project file used by the help compiler to create your application's help file.

### *HLP\\*.BMP*

These are a collection of bitmap files used by the help file topics.

**HLPALIAS.H**

This file is used to create help topic aliases, which map one context to another existing context. These are used when several commands need to be mapped to the same help topic. This is included in the [ALIAS] section of the Help project (.HPJ) file.

**HLPMAP.H**

This file contains mappings for objects and commands that are application specific. This is included in the [MAP] section of the .HPJ file.

**HLPAFX.RTF**

This file contains the standard help topics for Microsoft Foundation Class Library commands and screen objects.

**HLPKEYS.RTF**

This file contains Help for keys used in applications for Windows.

**HLPMDI.RTF**

This is the template help file for MDI applications.

**HLPTERMS.RTF**

This is the template help file for your application's glossary of terms.

## Running App Studio

App Studio is the application you use to create and edit resources. Although you can run App Studio independently, App Studio, Visual Workbench, and ClassWizard are more easily integrated if you run App Studio from within Visual Workbench. You can run App Studio from Visual Workbench using one of two methods:

- Select App Studio from the Tools menu
- Open a project resource file (assuming the Open .RC Files Using App Studio check box in the Editor dialog box is enabled)

If you occasionally edit your resource files in a text editor, you may prefer to open App Studio from the Tools menu and use the Open File dialog box to open resource files as text in a Visual Workbench source window.

App Studio is installed on the Tools menu with the **\$SRC** argument macro. When you choose App Studio from the Tools menu, the **\$SRC** macro expands to either the first resource file in the project list (for Visual Workbench projects) or the target name with an .RC extension (for external projects). For external projects, the target name is displayed in the Debug Target Name box in the Project Options dialog box.

If no resource file exists, App Studio is opened without one.

Since App Studio creates and edits resources graphically, you may not feel the need to edit your resource files in a text editor. In this case, it can be useful to open App

Studio by simply selecting a resource file in Visual Workbench's Open File dialog box.

---

**Note** To open App Studio on a resource file that you select in the Open File dialog box or from the toolbar's Project Files list, enable the Open RC Files Using App Studio check box in the Editor dialog box (accessed from the Options menu). The state of this check box persists between Visual Workbench sessions.

---

You can use the Visual Workbench Open File dialog box to open App Studio on several file types, including:

- Resource files (.RC).
- Graphics image files (.BMP, .ICO, or .CUR).
- Executable files (.EXE, .DLL, or .VBX).

When you open App Studio on a resource file, the file is first compiled using the resource compiler and the resulting data is stored in memory for App Studio's use. App Studio stores this information in a file with the extension .APS for later recall and for caching when memory is limited.

When you open a graphics image file, it becomes an App Studio resource, and when you open an executable file, its resources are extracted and can be modified in App Studio.

► **To run App Studio on a resource, graphics image, or executable file:**

1. From the Visual Workbench File menu, choose Open.
2. In the Open File dialog box, select the filename.
3. Choose OK.

App Studio is launched with the filename as an argument. App Studio compiles the resource file first. If an error occurs during this stage, the resource file is opened in Visual Workbench as a text file with the insertion point on the line that caused the error.

Since the resource file is usually included in the project list, you can also use the Project Files button on the toolbar to open App Studio.

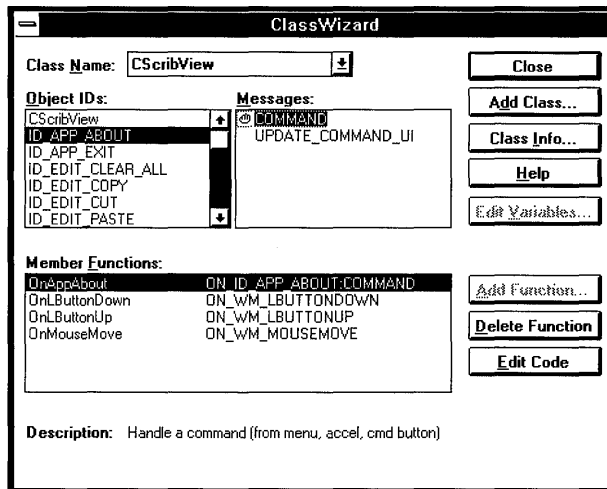
► **To run App Studio on a project resource or graphics file using the toolbar:**

1. Click the Project Files button (on the far left of the toolbar).  
The Project Files list appears.
2. Double-click the filename in the Project Files list.
3. App Studio is launched with the filename as an argument.

## Using ClassWizard

ClassWizard is a Microsoft Foundation Class Library class generator and manager that automates the creation and editing of code to handle messages and dialog-box data. It generates the source code for new classes that can receive messages and also generates member functions and message maps in those classes to bind the messages to code. It also maps dialog-box data to member variables and automates the validation of data entered in a dialog box.

ClassWizard can be opened from either App Studio or Visual Workbench (see Figure 13.4). Since you design the user-interface objects in App Studio, you usually open ClassWizard from App Studio and generate the message-handler functions for the user-interface objects as you go. Chapter 9, “Using ClassWizard,” in the *App Studio User's Guide* contains a comprehensive description of using ClassWizard to do this and more.



**Figure 13.4** ClassWizard

This section describes two kinds of ClassWizard activities that you are likely to perform while running ClassWizard from Visual Workbench: deriving classes and creating message handlers for standard Windows messages. These activities can be performed at any time in ClassWizard, regardless of whether you open it from Visual Workbench or App Studio. However, since deriving classes and creating message handlers aren't directly related to developing resources, you may want to perform these activities while editing and developing code in Visual Workbench.

## Creating New Classes

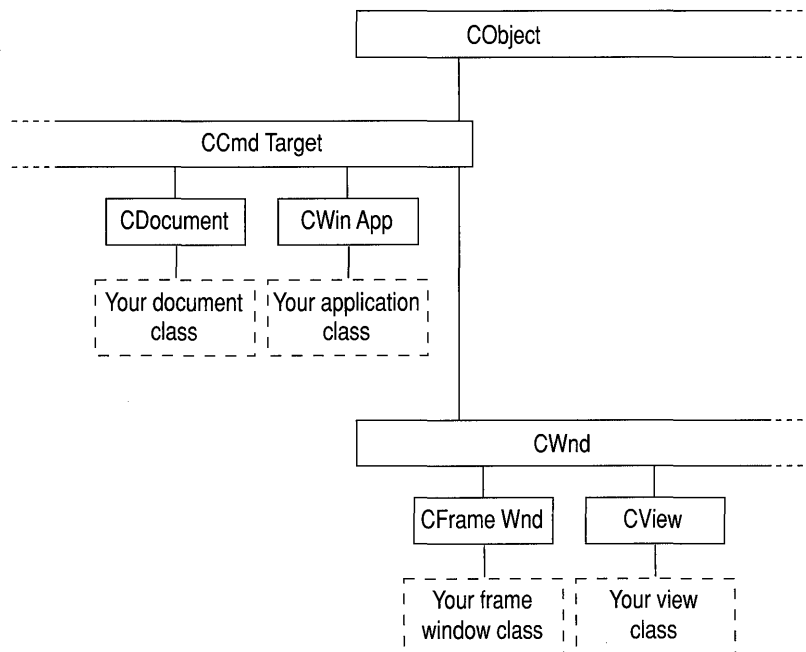
You can use ClassWizard to add new classes to your application. ClassWizard can derive a new class from any class with **CCmdTarget** as a base class.

---

**Important** If you plan to use ClassWizard to create message-handler member functions in any new classes, you should use ClassWizard to create the new classes so that the class code is in a format recognized by ClassWizard.

---

Figure 13.5 shows a segment of the Microsoft Foundation Class Library hierarchy. The **CCmdTarget** class provides message-routing services for user-interface objects, so any classes with **CCmdTarget** as a base class can be used to handle messages from menu items, dialog-box controls, toolbar buttons, and so on. (These are actually IDs passed by **WM\_COMMAND** messages.) This applies to document view classes (those derived from **CView**), which can also handle Windows messages, as well as document structure classes, such as those derived from **CDocument**.



**Figure 13.5** Classes Derived from **CCmdTarget**

Most often, you will create new dialog classes for dialog resources you have created in App Studio. This should be done after creating the dialog resources. Chapter 9, “Using ClassWizard,” in the *App Studio User’s Guide* provides the procedure for creating new dialog classes.

You may also want to create new classes other than dialog classes that have the ability to respond to user input and either display data or contain data. For example, when AppWizard creates an application, it derives one document class and one view class. You might want to add a second view class so a user can display the document data in different ways, for example as a graphic chart or as a table. Both the document class and the view class can respond to user-interface object events. The document class is used to contain the data and to serialize it. The view class is used to display the data and also respond to Windows messages.

► **To derive a new class using ClassWizard:**

1. From the Browse menu, choose ClassWizard.

The ClassWizard dialog box appears.

2. Choose the Add Class button to open the Add Class dialog box (see Figure 13.6).

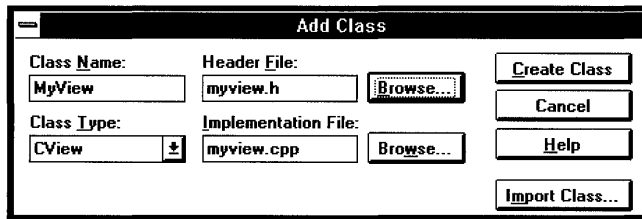


Figure 13.6 ClassWizard's Add Class Dialog Box

3. In the Class Name box, type the name of the class you want to create.  
Notice that as you type the name, the Header File and Implementation File text boxes automatically use the first eight characters of the class name for the suggested filenames.
4. From the Class Type drop-down list, select the type of class from which you want to derive your application's class.
5. If you want the class prototype code to be generated in a different file, type the filename in the Header File text box, or use the Browse button to select the name of an existing header file.
6. If you want the class implementation code to be generated in a different file, type the filename in the Implementation File text box, or use the Browse button to select the name of an existing source file.
7. If you are creating a class that is derived from either **CDialog** or **CFormView**, select the resource identifier from the Dialog ID drop-down list.
8. Choose Create Class.

If you skipped step 7, ClassWizard prompts you with a message box to warn you about using a resource identifier. If your new class is not derived from **CDialog** or **CFormView**, choose Yes.

ClassWizard generates the source code for the new class and returns to the ClassWizard dialog box with the new class selected in the Class Name drop-down list.

## Creating Message Handlers for Windows Messages

The classes that handle Windows message—that is, the classes that are derived from the `CWnd` class—are often called upon to deal with much of the graphical overhead of programming in Windows, such as responding to resize messages (`WM_SIZE`) or mouse messages (such as `WM_LBUTTONDOWN`).

ClassWizard makes it easy to generate skeleton code to handle these standard messages from Windows. It displays a list of all the messages that pertain to each class type so all you need to do is select the message and choose the Add Function button.

### ► To create message-handler code for Windows messages:

1. If ClassWizard is not already open, open it from the Browse menu (or press CTRL+W).
2. In the Class Name drop-down list, select the class to which you want to add the message-handler member function and message-map entry.
3. In the Object IDs box, select the class name, which will be the first entry in the list.

A list of Windows messages is displayed in the Messages box. The contents of this list depend on the Message Filter set for that class, which you can see by choosing the Class Info button. The message filter is automatically set for each class type to display the appropriate types of messages. For example, a class derived from `CView` displays messages filtered for a child window.

To change the selection of the Message Filter box, choose the Class Info button, select a different filter in the Message Filter box, and choose Close.

4. Select a message that you want handled.

Notice that as you select different messages in the Messages list box, a description of each message appears at the bottom of the dialog box (see Figure 13.7).

5. Choose the Add Function button.

ClassWizard automatically writes the member function prototype definition, enters the function name and message name in the message map, and enters the declaration in the header file.

Notice that the function appears in the Member Functions box and that a hand icon appears at the left of the message to indicate that it is handled.

6. If you want, you can now choose the Edit Code button to write code for the message-handler member function that has been inserted in your code.

ClassWizard closes and Visual Workbench opens a source window with the insertion point in the message-handler member function that was just created.



7. If you skipped step 6, close ClassWizard by choosing OK.

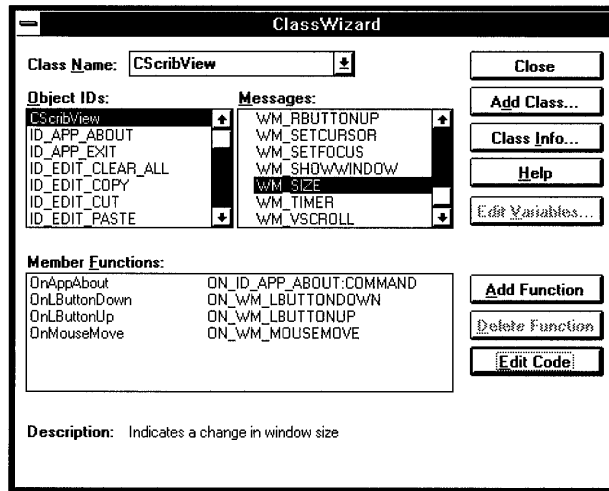


Figure 13.7 Creating Message Handlers for Windows Messages

# Index

32-bit applications 11

## A

Aborting build 104  
 About dialog box 46  
 Accelerator keys *See* Keys, shortcut  
 Add Tool dialog box 200  
 Adding  
   commands to Tools menu 199–200  
   expressions and variables to Watch window 191, 193  
   libraries 7  
   project files 100–101  
 AFX.RTF 218  
 afxDump 180  
 ALIAS.H 218  
 Analysis tools, installation 9  
 App Studio  
   command on Tools menu 43  
   Help file installation 8  
   opening from Visual Workbench 28  
   overview 23  
   resource files 80  
   running  
     from Visual Workbench 81  
     via opening resource file 218–219  
     via Tools menu 218  
   using 27  
 App Studio User's Guide, installing sample programs 8  
 Application development process 21, 26, 30–32, 209  
 AppWizard  
   Classes dialog box 210, 214–215  
   closing 210  
   command on Project menu 39  
   files generated 215–218  
   opening 210  
   Options dialog box 210, 212–213  
   overview 23–24, 209–210  
   running 26–27  
   specifying  
     drive, directory 211  
     project name 210–211  
   starter files 27  
   when to use 26  
 .APS file extension 219  
 Argument macros 202–203  
 AUTOEXEC.BAT, configuring 5, 10

## B

Base Class Graphs  
   browsing 172–173  
   described 170  
 .BMP file extension 217  
 Bookmarks  
   clearing 38, 83  
   moving to 38, 82–83  
   setting 82  
 Breakpoints  
   dialog box *See* Breakpoints dialog box  
   evaluating expressions 187  
   overview 183  
   saving between sessions 183  
   setting  
     and clearing 183, 185  
     in SORTDEMO 184  
     locations 186  
   testing messages 187–188  
   types 186–188  
 Breakpoints dialog box 42, 183, 185–188  
 Browse menu  
   browsing commands 168  
   ClassWizard command 41  
   compared to Browse window 165  
   Go to Definition command 41, 164–165, 168, 177  
   Go to Reference command 41, 164–165, 168, 177  
   Next command 41, 168, 177  
   Open command 41, 163–164  
   Pop Context command 41, 168–169  
   Previous command 41, 168, 177  
 Browse window  
   compared to Browse menu 165  
   jumping to definitions and references 171, 177  
   opening 41, 163  
   using Query controls 166  
 Browser  
   database *See* Browser database  
   graphical icons 171  
   graphs  
     expanding and collapsing 171  
     overview 169  
   Help window *See* Browser Secondary Help Window  
   overview 161  
   wildcard symbols 167

- Browser database
    - creating 161–162
    - opening 163–165
    - querying 165–169
  - Browser Secondary Help Window 71, 73
  - Browsing
    - Base Class Graphs 172–173
    - Call Graphs 174–175
    - class and function graphs 169
    - class member functions and variables 172, 174
    - Definitions and References 164–168, 176–177
    - Derived Class Graphs 172
    - function relationships 175
    - Help information 66
    - shortcut keys for 50
  - .BSC file extension 129, 163
  - BSCMAKE 11, 129
  - Build and compile shortcut keys 50
  - Build mode
    - Debug 182
    - selecting 103
  - Build options
    - accessing Help 72–73
    - customizing 111
    - selecting 103
  - Build toolbar button 61
  - Build utilities directory 203–204
  - Building
    - aborting build 104
    - and running applications 32
    - external projects 109
    - in background 104
    - projects 103
    - sample applications 14–15
    - sample QuickWin programs 18–20
    - using toolbar 61
- C**
- .C file extension 79
  - C Lang/Libs Help, installing 8
  - C/C++ calling convention 119
  - C/C++ Compiler Options dialog box
    - accessing Help 72, 113
    - Build Options buttons 111–112
    - Category Settings group 112–113
    - Code Generation category
      - Calling Convention options 119
      - Check Pointers 121
      - Code Generator options 120
      - CPU options 118
      - Disable Stack Checking 121
    - C/C++ Compiler Options dialog box (*continued*)
      - Code Generation category (*continued*)
        - Floating-Point Calls options 119
        - overview 118
        - Struct Member Byte Alignment options 120
      - Custom Options (C++) category
        - C++ Pointer to Member Representation 125
        - Disable Construction Displacements 127
        - General-Purpose Representation options 126
        - overview 124
        - Representation Model options 125
      - Custom Options category
        - Disable Microsoft Language Extensions 122
        - Eliminate Duplicate Strings 123
        - Enable Function-Level Linking 122–123
        - Other Options 124
        - overview 121–122
        - QuickWin Support 123
        - Suppress Display of Sign-On Banner 123
        - Warning Level options 123
        - Warnings as Errors 123
      - Debug Options category
        - Full (C7 Compatible) 128
        - Full, Use Program Database 128
        - None 128
        - overview 127–128
        - Partial (Line Numbers Only) 128
      - Listing Files category
        - Assembly 130
        - Browser Information 129
        - Don't Pack Information 129
        - Include Local Variables 129
        - Include Machine Code 130
        - Include Source and Machine Code 130
        - overview 129
      - mapping to CL command-line options 51–52
      - Memory Model category
        - Assume 'external' and Uninitialized Data 'far' 132
        - Model options 130
        - New Segment Data Size Threshold 131
        - overview 130
        - Segment Setup options 131
      - Optimizations category
        - Custom Options 134
        - Customize 132, 134
        - Default 132
        - Disable (Debug) 133
        - Inline Expansion of Functions options 135
        - Inline Function Size options 135
        - Maximize Speed 132–133
        - Minimize Size 132–134
        - overview 132

- C/C++ Compile Options dialog box (*continued*)
  - Options String 111, 113
  - overview 118
  - P-Code Generation category
    - Disable P-Code Quoting 137
    - Number of P-Code Entry Tables 137
    - overview 136
    - P-Code Optimization On 137
    - Remove P-Code Native Entry Points 137
    - Sort Local Variables in Occurrence Order 137
  - Precompiled Headers category
    - Automatic Use of Precompiled Headers 138–140
    - overview 138
    - Precompile up to Header 141–142
    - Precompile with Source 141–142
  - Preprocessor category
    - Ignore Standard Places of Include Files 144
    - Include Path 144
    - Individual Symbols to Undefine 143
    - overview 142
    - Symbols and Macros to Define 143
    - Undefine All Symbols 144
  - Segment Names category
    - Code Segment 145
    - Data Segment 145
    - Module Segment 146
    - overview 144
    - P-Code Segment 145
    - V-Table Segment 146
  - Use Project Defaults button 115
  - Windows Prolog/Epilog category
    - Generate Prolog/Epilog For options 147
    - overview 146
    - Protected Mode options 148
- Call Graphs
  - browsing 174–175
  - described 170
  - jumping to definitions and references 175
  - recursive functions 174
- Call Stack dialog box 42
- Call stack, viewing 194–195
- Cancelling include dependency scan 102
- Caps Lock status indicator 63
- Changing
  - project types 94
  - register values 195
  - workspace names 106
- cin iostream object 17
- CL options
  - See also* Compiler options
  - /AC 130
  - /AH 130
  - /AL 130
  - /AM 130
- CL options (*continued*)
  - /AS 130
  - /ASd 131
  - /AT 130
  - /D 143
  - /f 120
  - /f- 120
  - /Fa 130
  - /Fc 130
  - /Fl 130
  - /Fp 141
  - /FPa 119
  - /FPc 119
  - /FPc87 119
  - /FPi 119
  - /FPi87 119
  - /FR 129
  - /Fr 129
  - /G0 through /G3 118
  - /G3 133
  - /GA 146, 148
  - /Gc 119
  - /GD 146, 148
  - /Gd 119
  - /GEe 148
  - /GEf 148
  - /Gf 122–123
  - /Gn 135–137
  - /Gp 137
  - /Gs 121
  - /Gt 131–132
  - /GW 146
  - /Gw 146
  - /Gx- 132
  - /Gy 122–123, 151
  - /I 144
  - /Mq 122
  - /ND 145
  - /NM 146
  - /nologo 122–123
  - /NQ 145
  - /NT 145
  - /NV 146
  - /O1 132–134
  - /O2 132–133
  - /Oa 134
  - /Ob0 135
  - /Ob1 135
  - /Ob2 135
  - /Od 132–133
  - /Oe 134
  - /Of 136
  - /Of- 137
  - /Og 134

CL options (*continued*)

- /Oi 134
  - /Ol 134
  - /Op 134
  - /Oq 136–137, 145
  - /Or 134
  - /Os 134
  - /Ot 134
  - /Ov 136–137
  - /Ov- 137
  - /OV0 through /OV9 135
  - /Ow 134
  - /Ox 134
  - /Oz 134
  - /U 143
  - /u 144
  - /vd0 127
  - /vmb 125
  - /vmg 125–126
  - /vmm 126
  - /vms 126
  - /vmv 126
  - /W0 through /W4 125
  - /WX 122–123
  - /X 144
  - /Yc 138, 141–142
  - /Yu 138, 141–142
  - /YX 138–140
  - /Z7 127–128, 133, 156
  - /Za 122
  - /Zd 127–128, 133, 156
  - /Zi 127–128, 133, 156
  - /Zn 129
  - /Zp1 120
  - /Zp16 120
  - /Zp2 120
  - /Zp4 120
  - /Zp8 120
  - /Zr 121
- CL, MS-DOS-extended 11
- Class graphs, browser 166
- Classes, modifying in AppWizard 214–215
- ClassWizard
- binding user-interface objects to code 25, 29
  - creating new class 25
  - creating message-handler functions 25, 223
  - deriving classes using 221–223
  - dialog-data exchange 26
  - editing code 29
  - managing source code 31
  - menu command 41
  - opening 26
  - overview 24–26, 220

Class Wizard (*continued*)

- using from within Visual Workbench 220
  - when to use 27
- Clearing breakpoints 183
- Clipboard
- copying to 37
  - deleting to 37
  - pasting from 37
- Closing
- AppWizard 210
  - projects 100
  - QuickWin applications 20
  - source files 82
- .CLW file extension 216
- CodeView
- command on Tools menu 43
  - compared to Visual Workbench debugger 179
  - installation
    - CodeView for MS-DOS 9
    - CodeView for Windows 9
    - symbolic debugging information 156
- \$Col argument macro 202
- Color dialog box 44, 86, 205–206
- Color syntax 205–206
- Column number indicator 63
- .COM file extension 156
- Command-line options
- CL *See* CL options
  - LINK *See* LINK options
  - Visual Workbench 9, 60
- Command-line tools
- configuring MS-DOS for 10
  - using on Tools menu 201–202
- Comments, creating with AppWizard 213
- Comparison of Visual C++ editions 4
- Compile File toolbar button 61
- Compiler options
- See also* CL options
  - alphabetical listing 51–52
  - Browser Information 161–162
  - customizing 111
  - project defaults 115
  - using dialog box to set 118–148
  - Visual C++ editions differences 4
- Computer required to run Visual C++ 5
- CompuServe, Microsoft forums on xv–xvi
- Conditional building 93
- Configuration
- system, required 5
  - of Visual C++ for MS-DOS 10
- Context-sensitive Help, described 66–67
- Context-sensitive Help, AppWizard source code option 213
- Conventions, typographical xiii
- Core project types list 94

cout iostream object 17

.CPP file extension 79

### Creating

browser databases 161–162

projects 97–98

source files 76

user-interface objects 27–28

Visual C++ applications 26

Current workspace, described 105

Custom VBX Controls, AppWizard source code option 213

Customizing build options 111

Customizing Visual Workbench

commands

adding to Tools menu 199–200

editing on Tools menu 201–203

overview 199

setting

directories 203–204

font type, size 207–208

syntax coloring, changing 205–206

CVPACK 11, 128

## D

.DBG file extension 156

Debug dialog box 44, 179, 198

### Debug

information, generating 182

kernal, installation 9

### Debug menu

Breakpoints command 42

Go command 14, 32, 42, 104, 109–189

QuickWatch command 43

Restart command 42, 189

Show Call Stack command 42, 194–195

Step Into command 42, 190

Step Out command 42, 190

Step Over command 42, 190

Step to Cursor command 42, 190

Stop Debugging command 42, 189

### Debuggers

*See also* CodeView

CodeView compared to integrated debugger 179

### Debugging

DLLs 179

expanding and collapsing variables 191–192

external projects 107, 109

Hard and Soft mode 196–197

hexadecimal display 198

mixed source, assembly 198

modifying expressions 193

modifying structures, arrays 194

modifying variables 193–194

overview 32–33

### Debugging (*continued*)

phases 179

running, tracing 189

setting build mode for 182

shortcut keys 51

toolbar buttons 62

using breakpoints 183, 185–189

using QuickWatch 192–193

using Registers window 195–196

using Show Call Stack 194–195

using Watch window 190–191

windows 180–181

### Default

file extension when opening files 79

workspace names 106

\_DEFAULT.PIF, modifying for Tools menu commands 202

### Definitions and References

in graphical queries 170–171

querying from Browse window 169, 176–177

querying from source files 164–165, 168

### Deleting

commands from Tools menu 201

expressions, variables from Watch window 191

project files 100–101

text 37

Dependencies, include 101–102

### Derived Class Graphs

browsing 172

described 170

Deriving classes using ClassWizard 221–223

Developing Visual C++ applications 21, 209

### Development stages

application creation 26–27

application development 27

overview 26

### Dialog boxes

Add Tool 200

Breakpoints 42, 183, 185–188

C/C++ Compiler Options 59, 71–73, 111–115, 118–148, 161–162

Call Stack 42

ClassWizard 41, 220

Color 44, 86, 205–206

Debug 44, 108, 179, 198

Directories 44, 189, 204

Edit - Projectname 39, 100–102

Editor 37, 76, 78, 80–81, 85, 218

External Project Options 108–109

Find 37, 86–88

Font 44, 207–208

getting help in 67

Line 38

Linker Options 67, 71–73, 111–115, 148–157

Messages 188

Dialog boxes (*continued*)

- MFC AppWizard 39, 209–211
  - Modify Variable 193–194
  - New Project 39, 59
  - Open File 41, 79, 81, 161
  - Open Project 14, 107
  - Page Setup 36
  - Print 36, 90–91
  - Project Options 14, 44, 59, 103, 111, 182
  - Resource Compiler Options 158–159
  - QuickWatch 43, 191–194
  - Replace 37, 89
  - Save As 36, 77–78
  - Show Call Stack 194–195
  - Tools 43, 200–203
  - Workspace 44, 106
- Dialog-data exchange 26
- Differences between Visual C++ editions 4
- \$Dir argument macro 202
- Directories
- changing settings 203–204
  - original settings 204
- Directories dialog box
- Executable Files Path 204
  - Help Files Path 204
  - Include File Path 204
  - Library Files Path 204
  - MFC Files Path 189, 204
  - purpose of 44, 204
- Duplicating windows 79
- Dynamic-link libraries
- debugging 179
  - files in project list 95
  - project type 95
  - running and debugging 108

**E**

Edit - Projectname dialog box 99–100

## Edit menu

- Copy command 37
- Cut command 37
- Delete command 37
- Find command 37, 87
- Find Matching Brace command 37, 83
- Paste command 37
- Read Only command 37
- Redo command 37
- Replace command 37, 86, 89–90
- Undo command 37

## Editing

*See also* Editor

## keys

- copying 48
- deleting text 48
- inserting 48
- inserting, displaying tabs 48
- insertion point movement 47
- scrolling 48
- searching 49
- selecting text 47

Tools menu command 201

user-interface objects 27–28

Watch window 190–191

## Editor

- closing source files 82
- creating source files 76
- dialog box *See* Editor dialog box
- finding text 86–89
- highlighting language syntax 85–86
- moving around in files 82–83
- opening
  - files using file filters 81
  - recently used files 81
  - resource files 80–81
  - source files 79, 81
- overview 75
- printing 90–91
- replacing text 86, 89–90
- saving source files 76–78
- setting
  - save options 78
  - tabs 85
- using keyboard shortcuts 84
- write-protecting files 86

## Editor dialog box

Open .RC Files Using App Studio 80–81, 218

Prompt Before Saving 78

Save Before Running Tools 78

setting save options in 37, 44, 76

Tab Settings group 85

Tab Stops 85

Environment variables in MS-DOS for Visual C++ 10

## Errors

- accessing from Output window 181
- copying, printing 182
- displayed on status bar 63
- getting Help on 181
- in Watch window 190

Executing external project targets 109

Execution, controlling in debugger 189

Expanding, collapsing graphs 171

- Expressions, Watch window
  - adding to 190–191, 193
  - copying to 191
  - deleting from 191
  - valid expressions 190
- Extensions, file *See* File extensions
- External drives required to install Visual C++ 5
- External Project Options dialog box
  - Debug Build 108
  - Project Type 108
  - Rebuild All Options 109
  - Release Build 109
  - Target File Name 108
- External projects
  - AppWizard source code option 213
  - building 109
  - debugging 107, 109
  - described 107
  - opening 107
  - running 108–109
  - setting build options 108

## F

- F1
  - See also* Help
  - function key 65–67, 71
  - Help on errors 181
  - Help on Options String mnemonics 113
- Fast compiler 120
- Features, Visual C++ 4
- \$File argument macro 202
- File extensions
  - added by include dependency scanning 101
  - .APS 219
  - .BMP 217
  - .BSC 129, 163
  - .C 79
  - .CLW 216
  - .COM 156
  - .CPP 79
  - .DBG 156
  - default used upon opening 79
  - .HPJ 217
  - in Open File dialog box 79, 81
  - in Save As File dialog box 79
  - .MAK 93
  - .MAP 156
  - .PCH 138, 217
  - .PDB 128
  - .RC 80, 158
  - recognized for syntax coloring 206

- File extensions (*continued*)
  - .SBR 129
  - .VCW 93
  - .WSP 106
- File filters *See* Filename filters
- File menu
  - Close command 36, 82
  - Exit command 36
  - most recently used filenames 81
  - New command 18, 36, 76
  - Open command 36, 61, 79–80, 163–164
  - Page Setup command 36
  - Print command 36, 90–91
  - Read Only command 86
  - Save All command 36, 77
  - Save As command 18, 36, 76, 78
  - Save command 36, 61, 76–77
- \$FileDir argument macro 202
- \$Filename argument macro 202
- Filename filters
  - .BSC 163
  - customizing 81
  - described 80
- Files
  - generated by AppWizard 215–218
  - in project list
    - adding 100–101
    - deleting 100–101
    - QuickWin application 95
    - Static library 96
    - Visual Basic Custom Control 96
    - Windows DLL 95
    - Windows-based application 95
- Find dialog box
  - Find Next 88
  - Match Case 87
  - Match Whole Word Only 87
  - parameters for toolbar Find box 89
  - Regular Expression 87
  - Search direction 87
  - Set Bookmarks on All 88
  - using 37, 86–87
- Find Next toolbar button 61
- Find toolbar box 61, 86, 88–89
- Flags, setting and clearing 196
- Font dialog box 44, 207–208
- Font Editor, installation 9
- Fonts, setting
  - for new windows 208
  - for active window 207
  - in different windows on same file 208
- Function graphs, browser 166



**G**

- Generating code 24
- Graphical query types
  - Base Class Graph 169–170
  - Call Graph 169–170
  - Caller Graph 169–170
  - Derived Class Graph 169–170
  - list of 169
- GRAPHICS.LIB 17
- Graphs, browser
  - expanding, collapsing 171
  - overview 169

**H**

- Hard disk space required to run Visual C++ 5
- Hard, Soft mode debugging 196–197
- hdrstop pragma 140
- Help
  - accessing
    - by searching 66
    - from the menu 65
    - in dialog box 65, 67
    - on compiler, linker option controls 72, 113–114
    - on error number 65–66
    - on language keyword 65–66
    - on library function 65
    - on menu command 65, 67
    - on option string mnemonics 72
    - on toolbar button 67
  - compiler, linker options 72
    - accessing alphabetically 73
    - accessing by category 72–73
    - getting Help from dialog box 113–114
  - compiler, installation 9
  - displayed on status bar 63
  - environment versus reference 68–69
  - file installation 8
  - files directory 203–204
  - installation options 7
  - navigating 67–69
  - product support *See* Product support
  - project files 217
  - reference information 67, 70–71, 73
  - resource compiler options 113
  - Visual Workbench environment 67, 69–70
- Help menu
  - About Visual C++ command 46
  - Build Tools command 46, 66
  - C/C++ Language command 46, 66

Help menu (*continued*)

- Foundation Classes command 46, 66
  - Obtaining Technical Support command 46
  - Search for Help On command 46, 66
  - Visual Workbench command 46, 66–67
  - Windows 3.1 SDK command 46, 66
- Hexadecimal display 198
  - Hiding, displaying
    - status bar 63
    - toolbar 60
  - Highlighting syntax 85–86
  - .HPJ file extension 217

**I**

- Include dependencies 101–102
- Include directory 203–204
- Index Help window 68, 70
- Initial installation of Visual C++ 5
- \_\_inline directive 135
- Installation options 7–9
- Installing
  - libraries 8
  - math support 8
  - memory models 8
  - MS-DOS tools 9
  - project types 8
  - sample programs 8
  - Visual C++
    - components 7
    - for command-line operation 5, 10
    - initially 5
    - Professional Edition tools 9
  - Visual Workbench 3, 9
- iostream
  - objects 17
  - operators and QuickWin 95

**K**

- Keys, shortcut
  - browsing 50
  - building, compiling 50
  - debugging 51
  - editing 47–49, 84
  - toolbar 49
  - window management 50
- KEYS.RTF 218

## L

/LIB option string mnemonic 149, 157

## Libraries

- adding 7
- directory 203–204
- installation option 8

Line dialog box 38, 82

Line number indicator 63

\$Line argument macro 202

## LINK

- See also* Linker options
- MS-DOS-extended 11

## LINK options

- See also* Linker options
- /ALIGN 157
- /CO 155–156
- /EXEPACK 151–152
- /FARCALL 151–152
- /INFO 155–156
- /LINE 155–156
- /MAP 155–156
- /NOD 150
- /NOE 150
- /NOI 150–151
- /NOLOGO 154
- /NOPACKF 151
- /ONERROR:NOEXE 155–156
- /PACKC 152
- /PACKD 152
- /SEG 153
- /STACK 153
- /TINY 155–156

## Linker options

- See also* LINK options
- alphabetical listing 51, 55
- customizing 111
- project defaults 115
- using dialog box to set 148–157

## Linker Options dialog box

- accessing Help 72
- Build Options buttons 111–112
  - Common to Both 111
  - Debug Specific 111
  - Release Specific 111
- Category Settings group 112–113
- getting Help on options 113
- Help button 113
- Input category
  - Distinguish Letter Case 151
  - Ignore Default Libraries 150
  - Libraries 149
  - overview 149

Linker Options dialog box (*continued*)

- Input category (*continued*)
  - Prevent Use of Extended Dictionary 150
  - Specific Libraries to Ignore 150
- mapping to LINK command-line options 51, 55
- Memory Image category
  - Don't Remove Unreferenced Packaged Functions 151
  - Max. Number of Segments 153
  - overview 151
  - Pack Code 152
  - Pack Data 152
  - Pack EXE File 152
  - Stack Size 153
  - Translate Far Calls 152
- Miscellaneous category
  - Other Options 154
  - overview 153
  - Suppress Display of Sign-On Banner 154
- Options String, Help on options 113
- Output category
  - Create Map File 156
  - Generate Debugging Information 156
  - Include Line Numbers/Addresses in MAP 156
  - overview 155
  - Prevent Creation of EXE on Linker Error 156
  - Produce COM File 156
  - Produce More Detailed Output 156
  - Segment Alignment 157
- Use Project Defaults button 115
- Windows Libraries category
  - Import Libraries and DLLs 157
  - overview 157

## Loading

- files at startup using the command line 9
- workspaces 106

Locals window 45, 193

## M

Macros, for tools arguments 203

MAINFRM.CPP 216

MAINFRM.H 216

.MAK file extension 93

Make P-Code utility 138

## Makefiles

- external 107
- Visual Workbench 93

MAKEHELP.BAT 217

## Managing source code

- overview 31
- using ClassWizard 31
- using Visual Workbench browser 31–32

.MAP file extension 156

- MAP.H 218
  - Math support, installation option 8
  - MDL.RTF 218
  - Member functions, variables 172, 174
  - Memory
    - models, installation option 8
    - required to run Visual C++ 5
  - Menus
    - Browse *See* Browse menu
    - Debug *See* Debug menu
    - Edit *See* Edit menu
    - File *See* File menu
    - getting help on 65
    - Help *See* Help menu
    - Options *See* Options menu
    - Project *See* Project menu
    - Tools *See* Tools menu
    - View *See* View menu
    - Window *See* Window menu
  - Message-handler functions generated with ClassWizard
    - for Windows messages 223
    - overview 25
  - Messages, breaking on 186–189
  - Messages dialog box 188
  - MFC AppWizard dialog box
    - described 209–210
    - Directory 211
    - Drive 211
    - New Subdirectory 211
    - Project Name 210–211
  - Microsoft CodeView *See* CodeView
  - Microsoft Foundation Class Library
    - application 21, 23
    - breaking on messages 189
    - building, running sample program 14–15
    - creating code 29
    - project type 94
    - sample programs, installing 8
    - source directory 203–204
  - Microsoft Library Manager (LIB) 150
  - Microsoft Overlaid Virtual Environment (MOVE) 97
  - Microsoft product support services *See* Product support
  - Microsoft Programmer's Workbench projects 13, 107, 109
  - Microsoft Resource Compiler 158
  - Microsoft Source Browser *See* Browser
  - Microsoft Visual Workbench *See* Visual Workbench
  - Minimizing Visual Workbench windows 64
  - Mixed source/assembly, debugging display 198
  - Modify Variable dialog box 193–194
  - Modifying
    - expressions while debugging 193
    - structures, arrays while debugging 194
    - variables while debugging 193–194
  - Module-definition files generated by AppWizard 216
  - Monitor type required to run Visual C++ 5
  - Moving to line number 82
  - MS-DOS
    - applications, described 96–97
    - configuring Visual C++ for 5, 10
    - environment variables 204
    - Overlaid application 96–97
    - P-code applications, described 96–97
    - Profiler installation 9
    - programming 17
    - programming in QuickWin 95
    - programs on Tools menu 202
    - required version 5
    - tools installation 9
  - MS-DOS-extended tools 11
  - MSVC.INI
    - build options in 115
    - directory information in 204
  - MSVC.PCH 138–139
  - MSVC.PDB 128
  - MSVCVARS.BAT 10
  - Multifile projects 97
  - Multimedia API reference, Help file installation 8
  - Multiple Document Interface, AppWizard source code
    - option 212
- ## N
- Name of new source window 76
  - Navigating Help
    - compiler, linker options 71–72
    - reference information 70–73
    - Visual Workbench environment 69–70
  - New Project dialog box 39
  - NMAKE 11, 13, 213
  - Num Lock status indicator 63
- ## O
- Object linking and embedding (OLE) 213
  - OLDNAMES.LIB 150
  - OLE Client, AppWizard source code option 213
  - Online reference *See* Help
  - Open File dialog box
    - opening source files 79
    - using filename filters 81
  - Open Project dialog box 14
  - Open toolbar button 61
  - Opening
    - App Studio from Visual Workbench 30
    - AppWizard 210
    - AppWizard's Options dialog box 212
    - Browse window 41
    - browser database 163–165

- Opening (*continued*)
    - ClassWizard 26, 220
    - external projects 107
    - Help contents screen 67
    - more than one window on source file 79
    - project files 93
    - projects 100
    - recently used files 81
    - resource files
      - as text files 80
      - invoking App Studio 80–81
    - source files
      - default file extension 79
      - using toolbar 60–61
  - optimize pragma 137
  - Optimizing
    - compiler 120, 134
    - disk space 5, 7
    - for size 133–134
    - for speed 133
    - options 134
  - Options
    - AppWizard 212–213
    - CL *See* CL options
    - LINK *See* LINK options
    - project *See* Project Options dialog box
  - Options dialog box, AppWizard 212
  - Options menu
    - Color command 44, 86, 205–206
    - Debug command 44, 108, 179, 198
    - Directories command 44, 204
    - Editor command 44, 78, 80, 85
    - Font command 44, 207–208
    - Project command 14, 44, 103, 108–109, 162
    - Tools command 44, 199–203, 218
    - Workspace command 44, 105–106
  - Options, project *See* Project Options dialog box
  - Options string, getting Help on 72
  - Output window
    - build status 104
    - described 180
  - OutputDebugString 180
  - Overlapping Visual Workbench windows 63
  - Overriding
    - global syntax settings 206
    - syntax coloring 205
  - Overtyping status indicator 63
- P**
- P-code options 136–137, 145
  - Page Setup dialog box 36
  - Pascal calling convention 119
  - PCH *See* Precompiled headers
    - .PCH file extension 138, 217
    - .PDB file extension 128
  - Pen windows API reference, Help file installation 8
  - Precompiled headers 138–142
  - Preferences
    - customizing 199–208
    - prompt before saving 76, 78
    - save before running tools 78
  - Print dialog box 36
    - Print Range 90–91
    - Printer 91
    - Setup button 91
  - Print Preview, AppWizard source code option 213
  - Printer options 91
  - printf function 17
  - Printing
    - AppWizard source code option 213
    - errors 182
    - selecting default printer 90
    - using the Print dialog box 91
  - Processor type required to run Visual C++ 5
  - Product support
    - within the United States xv–xix
    - worldwide xix–xxv
  - Program database file 128
  - Programs
    - adding to Tools menu 199–200
    - debugging 179–181
  - \$Proj argument macro 202
  - Project files
    - AppWizard-generated files 216
    - opening from the toolbar 93
  - Project Files toolbar button 61, 81
  - Project menu
    - AppWizard 24, 26, 39
    - Build command 14, 19, 32, 39, 103, 108–109
    - Close command 39
    - Compile File command 39
    - Edit command 39, 100
    - Execute command 14, 19, 40, 104, 109
    - Load Workspace command 39, 106
    - New command 39
    - Open command 14, 39, 107
    - Rebuild All command 32, 39, 104, 109
    - Save Workspace command 40, 105
    - Scan All Dependencies command 40, 102
    - Scan Dependencies command 40
    - Stop Build command 40, 104
  - \$ProjDir argument macro 202
  - Project options, overview 103
  - Project Options dialog box 14, 44, 111, 182
  - Project status file 93

## Project types

- added by Visual C++ Professional Edition 4
- changing 94
- choosing 94
- Microsoft Foundation Class 94
- MS-DOS application 97
- MS-DOS Overlaid application 97
- MS-DOS P-code application 97
- QuickWin application (.EXE) 19, 95
- Static library 96
- Visual Basic Custom Control 96
- Visual C++ Professional Edition 96
- Windows DLL 95
- Windows P-code application 96
- Windows-based application 94–95

## Project types list 94

## Projects

- AppWizard 210–211
  - benefits of using 93
  - building 103, 109
  - building versus rebuilding 103
  - closing 100
  - compiler options 103
  - creating 94, 97
  - debug versus release mode 103
  - default compiler options 115
  - default linker options 115
  - default when starting Visual Workbench 60
  - described 93, 97
  - external 107, 109
  - file extensions 93
  - introduction 13
  - linker options 103
  - opening 100
  - porting 115
  - rebuilding 104
  - selecting build mode 103
- PWB projects, using in Visual Workbench 13, 107, 109

**Q**

## Querying browser database 165–169

## QuickWatch dialog box

- Add to Watch Window button 193
- availability 193
- expanding, collapsing variables 191–192
- Modify button 193–194
- overview 192
- Zoom button 193

## QuickWatch toolbar button 62

## QuickWin

- closing applications 20
- overview 17–18
- project type 95
- writing, building, running sample program 18–19

## QuickWin application project type

- described 95
- files in project list 95

## Quitting Visual Workbench 60

## QWGDemo.CPP 18–19

## QWINTTEST.CPP 19, 75

**R**

## RAM required to run Visual C++ 5

## \$RC argument macro 81, 202, 218

## .RC file extension 80, 158

## RC options 158–159

## Re-creating work environment 104

## Read-only status indicator 63, 86

## README.TXT, App Studio-generated file 215

## Rebuild All toolbar button 61

## Rebuilding project 104

## Recalling previous Browse window queries 167

Reference Help *See* Help

## Regenerating file dependencies 102

## Registers

- 32-bit 196
- native 195
- p-code 196
- using in breakpoint addresses 187

## Registers window 45, 195–196

## Regular expressions 86–87

## Reinstalling Visual C++ 7

## Renaming source files 78

## Replace dialog box 37, 89

## Replacing text 86, 89–90

## Resource Compiler Options dialog box 158–159

## Resource compiler

- accessing Help 113
- options, customizing 111, 158–159

Resource editor *See* App Studio

## Resource files

- generated by AppWizard 216
  - opening in Visual Workbench 80
- Restoring original syntax coloring 206

## Reversing

- edit action 37
- undo 37

Run toolbar button 62, 184, 189  
Run-time libraries, Help file installation 8  
Running  
  App Studio 80–81, 218–219  
  applications 32  
  external projects 108  
  external target files 109  
  programs  
    in debugger 104  
    outside debugger 104  
  Setup program 5

## S

Sample programs  
  HELLO 14  
  installing 8  
  QWGDEMO 18  
  SCRIBBLE 21, 161  
  SORTDEMO 180, 182, 184  
Sample projects  
  GENERIC.MAK 97  
  HELLO.MAK 97–98  
Save As dialog box 36, 77–78  
Save toolbar button 61  
Saving  
  source files  
    using File menu 76–77  
    using toolbar 60–61  
    with new name 78  
  workspaces 105  
.SBR file extension 129  
scanf function 17  
SCRIBBLE sample program 21, 161  
Search dialog box 46  
Searching  
  and replacing 37  
  for Help 66  
  for text  
    using Find command 87–88  
    using toolbar 60–61, 88–89  
Secondary Help window 68–71, 73  
Setting  
  bookmarks 82  
  breakpoints 183  
  breakpoints with toolbar 62  
  directories in Visual Workbench 203–204  
  up Visual C++ 5  
Setup program  
  disk space optimization 5  
  running 5–7

Shortcut keys  
  editing 84  
  expanding, collapsing browser graphs 171  
  F9 183  
  searching for text 86, 88  
  summary 47–51  
Show Call Stack dialog box 194–195  
Single Document Interface, AppWizard source code  
  option 212  
SORTDEMO sample program 180, 182, 184  
Source files  
  automatically loading 105  
  closing 82  
  creating 76  
  multiple windows, opening 79  
  opening 61, 79, 81  
  saving 61, 76–78  
Speed and size of debug programs 182  
Standard I/O, QuickWin 17  
Starting Visual Workbench 60  
Startup, loading files at 9  
Static library project type 96  
Status bar  
  described 63  
  hiding, displaying 63  
  menu, toolbar descriptions 63, 67  
STDAFX.CPP 217  
STDAFX.H 217  
STDAFX.RC 216  
Step Into toolbar button 62, 190  
Step Out toolbar button 62, 190  
Step Over toolbar button 62, 190  
Stopping build 104  
Support services *See* Product support  
Syntax  
  coloring 205–206  
  highlighting 85–86  
SYSINCL.DAT 102  
System requirements 5

## T

Tabs, setting 85  
\$Target argument macro 202  
Target files 93  
Technical support *See* Product support  
TERMS.RTF 218  
Text  
  editor *See* Editor  
  files, ASCII 79  
Tiling Visual Workbench windows 63  
Toggle Breakpoint toolbar button 62, 183–184  
Tool argument macros 81, 202–203

## Toolbar

- AppWizard-generated source code 213
- buttons *See* Toolbar buttons
- categories 60
- general description 60
- getting help 67
- hiding, displaying 60
- shortcut keys 49

## Toolbar buttons

- Build 32, 39, 61
- Compile File 39, 61
- Find (box) 37, 88–89
- Find Next 37, 61, 86, 88–89
- Open 36, 61, 164
- Project Files 36, 61, 80–81
- QuickWatch 42, 62, 192–193
- Rebuild All 32, 39, 61
- Run 42, 62, 184, 189
- Save 36, 61, 76–77
- Step Into 42, 62, 189
- Step Out 42, 62, 189
- Step Over 42, 62, 189
- summary table 62
- Toggle Breakpoint 42, 62, 183–184

## TOOLBAR.BMP 217

## Tools

- installation options 9
- Visual C++ application development 22, 209

## Tools dialog box 44, 200–203

## Tools menu

- adding commands 199–200
- App Studio command 43, 81, 218
- CodeView command 43
- deleting commands 201
- editing commands 201
- modifying 199
- using argument macros 202–203

## Typographical conventions xiii

# U

## User-interface objects

- connecting to code 25, 28–29
- creating, editing 27–28
- introduced 21

# V

## Variables in Watch window

- adding to 190–191, 193
- copying to 191
- deleting from 191

## VBX custom controls, AppWizard 210, 213

## .VCW file extension 93

## View menu

- Clear All Bookmarks command 38, 83
- Line command 38, 82
- Mixed Source/Asm command 38, 198
- Next Bookmark command 38, 82–83
- Next Error command 38, 181
- Previous Bookmark command 38, 82–83
- Previous Error command 38, 181
- Status Bar command 38, 63
- Syntax Coloring command 38, 205–206
- Toggle Bookmark command 38, 82–83
- Toolbar command 38, 60

## Viewing function call stack 194–195

## Viewing variables, expressions

- expanding and collapsing variables 191
- using QuickWatch 192
- using Watch window 190–191

## Visual Basic Custom Control

- See also* VBX custom control, AppWizard described 96

## Visual C++

- additional components 7
- applications *See* Visual C++ applications, introduction
- directories where installed 7
- Help system 67, 69–70
- installing 3–11
- MS-DOS configuration 10
- Standard and Professional editions 4, 59
- system requirements 5
- tools
  - App Studio overview 23
  - applications 22
  - AppWizard overview 23–24
  - ClassWizard overview 24–26
  - Visual Workbench overview 22
  - Windows-hosted development 3
  - wizards 23
- what is 3

## Visual C++ applications, introduction

- building, running 32
- debugging 32–33
- described 21
- development process 21, 26–33

## Visual C++ Professional Edition

- additional project types 4, 96
- additional tools 8–9
- project types 94–97

## Visual C++ Standard Edition

- MS-DOS-style applications 17
- project types 94–96

## Visual Workbench

- as center of development tools 22, 31
- browser
  - database *See* Browser database
  - graphical icons 171
  - graphs 169–171
  - Help window *See* Browser Secondary Help Window
  - managing source code 31–32
  - overview 161
  - wildcard symbols 167
- command-line options 9, 60
- customizing preferences 199–208
- debugger *See* Debugging
- editor
  - closing source files 82
  - creating and saving source files 76
  - creating source files 76
  - finding text 86–89
  - highlighting language syntax 85–86
  - keyboard shortcuts 84
  - moving around in files 82–83
  - opening files 81
  - opening resource files 80–81
  - opening source files 79, 81
  - overview 75
  - printing 90–91
  - replacing text 86, 89–90
  - saving all source files 77
  - saving source files 76–78
  - setting save options 78
  - setting tabs 85
  - write-protecting files 86
- features 60
- Help file installation 8
- Help system 65–68
- introduction xi
- installing 3, 9
- managing source code 31
- overview 31–33, 59
- quitting 60
- setting command-line options 9
- starting 60
- toolbar 60
- using Programmer's WorkBench projects 109
- Visual C++ Standard versus Professional editions 59
- windows
  - arranging 63
  - minimizing 64

## W

- Watch window 45, 180, 190–191, 193
- Wildcards, browser
  - matching more than one symbol 167
  - types, defined 167
  - using to find member functions, variables 174
- Window management shortcut keys 50
- Window menu
  - Cascade command 45, 63
  - Duplicate command 45, 79
  - Locals command 45, 180, 191–193
  - Output command 45, 180–182
  - Registers command 45, 180
  - source file names 79, 83
  - Tile command 45, 63
  - Watch command 45, 180, 190–193
- Windows DLL, application project type 95
- Windows, Microsoft version required to run Visual C++ 5
- Windows P-code application project type 96
- Windows Software Development Kit
  - sample programs 8
  - using with Visual C++ tools 21, 23
- Windows, version 3.1 API, Help file installation 8
- Windows, Visual Workbench
  - debugging 180–181, 190, 195
  - Help 67–69
  - new source files 76
  - setting fonts in 207–208
- Windows-based application, project type 94–95
- Windows-hosted development tools, Visual C++ 3
- Wizards
  - AppWizard 23–24, 209
  - ClassWizard 24–26, 209, 220–223
- Workspace dialog box 105–107
- Workspaces
  - current 105
  - defining initial 106
  - described 104–105
  - file extensions 106
  - information saved 199
  - Last Workspace Used 105
  - loading 106
  - saving 105
- .WSP file extension 106





App Studio User's Guide →

---

← Visual Workbench  
User's Guide



# App Studio User's Guide



# Contents

## Part 1 Using App Studio

<b>Chapter 1 App Studio Overview</b> .....	<b>3</b>
App Studio Basics .....	4
Starting App Studio .....	4
Using the Resource Browser Window .....	5
Creating a New Resource .....	6
Editing an Existing Resource .....	7
Directly Manipulating User-Interface Objects .....	7
Using Undo and Redo .....	8
Using the Properties Window .....	9
Viewing Property Pages .....	10
Controlling the Properties Window .....	10
Introduction to App Studio Editing Windows .....	11
Dialog Editor .....	11
Menu Editor .....	12
Accelerator Table Editor .....	12
String Editor .....	12
Graphics Editor .....	13
Binary Data Editor .....	14
The Symbol Browser .....	14
Using ClassWizard .....	14
Simplifying Message Handling .....	15
Gathering and Validating Dialog-Box Data .....	15
Understanding Windows Resources .....	15
Predefined Windows Resources .....	16
Dialog Box .....	16
Menu .....	16
Accelerator Table .....	17
String Table .....	18
Bitmap .....	18
Icon .....	18
Cursor .....	19
App Studio Sample Resources .....	19
Where to Go from Here .....	20

<b>Chapter 2 Working With Files and Symbols</b> .....	<b>21</b>
Working with Files .....	22
Creating a New Resource File .....	22
Opening a Visual C++ Resource File .....	23
Reading Resource Files Not Created with App Studio or AppWizard .....	24
Converting Existing Resource Files to App Studio Format .....	25
Features Supported Only in Microsoft Foundation Class Library Resource Files .....	26
Copying Resources Between Files .....	26
Using Advanced Resource File Techniques .....	28
Changing the Name of the Symbols Header File .....	29
Using Shared (Read-Only) or Calculated Symbols .....	29
Including Resources From Other Files .....	30
Working with Symbols .....	31
Changing a Symbol or Symbol Name .....	32
Changing a Symbol's Numerical Value .....	33
Managing Symbols with the Symbol Browser .....	34
Creating New Symbols .....	35
Changing Unassigned Symbols .....	35
Opening the Resource Editor for a Given Symbol .....	36
Symbol Name and Value Restrictions .....	36
Symbol Name Restrictions .....	36
Symbol Value Restrictions .....	37
<b>Chapter 3 Using the Dialog Editor</b> .....	<b>39</b>
Types of Controls .....	39
Creating Dialog Boxes .....	40
Opening New or Existing Dialog Boxes .....	40
Dialog Box Coordinates .....	42
Adding Controls .....	42
Selecting Controls .....	45
Moving Controls .....	46
Deleting and Copying Controls .....	46
Sizing Individual Controls .....	47
Sizing a Control to Fit Its Caption .....	47
Sizing Combo Box Drop-downs .....	47
Changing the Tab Order .....	48
Defining Dialog Box Keyboard Access .....	50

Arranging Controls . . . . .	51
The Dominant Control . . . . .	51
Using Snap to Grid . . . . .	51
Aligning Controls . . . . .	52
Aligning . . . . .	52
Making Spacing Even . . . . .	53
Centering in the Dialog Box . . . . .	53
Arranging Pushbuttons . . . . .	54
Resizing Controls . . . . .	54
Using Custom Controls . . . . .	54
Using VBX Controls . . . . .	55
Editing VBX Control Properties . . . . .	56
Working with User-Defined Controls . . . . .	57
Connecting to Program Code . . . . .	58
Creating a Form View . . . . .	59
Testing a Dialog Box . . . . .	59
<b>Chapter 4 Using the Menu Editor . . . . .</b>	<b>61</b>
Working with Menus and Menu Items . . . . .	62
Opening New or Existing Menu Resources . . . . .	62
Creating Menus or Menu Items . . . . .	63
Selecting Menus or Menu Items . . . . .	64
Moving and Copying Menus or Menu Items . . . . .	65
Viewing the Menu Resource as a Drop-down Menu . . . . .	66
Associating a Menu Item with an Accelerator Key . . . . .	66
<b>Chapter 5 Using the Accelerator Table Editor . . . . .</b>	<b>69</b>
Opening a New or Existing Accelerator Table . . . . .	69
Editing the Accelerator Table . . . . .	70
Editing Accelerator Properties . . . . .	72
Typing in Accelerator Values . . . . .	73
Associating a Menu Item with an Accelerator Key . . . . .	73
<b>Chapter 6 Using the String Editor . . . . .</b>	<b>75</b>
String Tables . . . . .	75
Opening a New or Existing String Table . . . . .	75
Editing the String Table . . . . .	77
Editing a String's Properties . . . . .	79



<b>Chapter 7 Using the Graphics Editor</b> .....	<b>81</b>
Windows and Tools for Editing Graphics .....	81
The Image Editor Window .....	82
The Graphics Palette .....	82
The Status Bar .....	83
The Image Menu .....	83
Editing Graphical Resources .....	83
Opening New or Existing Graphical Resources .....	83
Setting Properties .....	85
Drawing and Painting .....	85
Showing and Hiding the Graphics Palette .....	86
Selecting Foreground and Background Colors .....	86
Freehand Drawing and Erasing .....	86
Drawing Lines and Closed Figures .....	87
Filling Bounded Areas .....	88
Picking Up Colors .....	89
Using the Selection Tool .....	89
Cutting, Copying, Clearing, and Moving .....	91
Choosing Opaque and Transparent Backgrounds .....	92
Flipping the Selection .....	93
Inverting Colors .....	93
Creating a Custom Brush .....	93
Resizing a Bitmap .....	94
Changing the Number of Colors in a Bitmap .....	96
Managing the Graphics-Editor Workspace .....	97
Using Image-Editor Panes .....	97
Selecting Panes .....	97
Sizing Image-Editor Panes .....	97
Zooming In and Out .....	98
Changing the Magnification Factor .....	99
Displaying and Hiding the Pixel Grid .....	100
Editing Icons and Cursors .....	100
Creating and Selecting Images .....	100
Editing Device Descriptions .....	102
Drawing with Screen and Inverse Colors .....	103
Setting a Cursor's Hotspot .....	104
Editing Toolbar Graphics .....	104

---

Managing Colors and Palettes .....	107
Creating Color Palettes .....	108
Saving and Loading Palettes .....	110
<b>Chapter 8 Using the Binary Data Editor .....</b>	<b>111</b>
Creating a New Data Resource or Custom Resource .....	111
Opening the Binary Data Editor .....	111
Editing Data .....	112
<b>Chapter 9 Using ClassWizard .....</b>	<b>113</b>
Adding a New Class or Importing an Existing One .....	114
Adding a New Class .....	115
Importing Existing Classes .....	117
Mapping Messages to Functions .....	117
Defining Message Handlers .....	118
Deleting Message Handlers .....	120
Jumping to Source Code from ClassWizard .....	120
Working with Dialog-Box Data .....	121
Dialog Data Exchange .....	121
Using DDX Variables .....	124
Dialog Data Validation .....	124
Custom Data Exchange and Validation .....	125
Example: Building a Dialog Box with ClassWizard .....	125
Example, Part 1: Defining a Message Handler for a Dialog-Box Control ...	126
Step One: Create the Dialog-Box Resource .....	126
Step Two: Define The Dialog-Box Class .....	126
Step Three: Define the Message Handler .....	127
Step Four: Begin Filling in the Message Handler .....	127
Example, Part 2: Using DDX/DDV .....	128
Step One: Define the Member Variable Using ClassWizard .....	128
Step Two: Set the Maximum Number of Characters for the Variable ...	128
Step Three: Set the Initial Value for the Variable in the Dialog-Box	
Class Constructor .....	129
Step Four: Create Control Variables to Use in the Message Handler ...	130
Keeping ClassWizard Updated When Code Changes .....	131
Deleting Classes .....	131
Renaming or Moving Classes .....	132
Rebuilding the ClassWizard (.CLW) File .....	132
Updating Existing Code for Use with ClassWizard .....	133

**Part 2 App Studio Reference**

<b>Chapter 10 App Studio Quick Reference</b> .....	<b>137</b>
Task Reference .....	137
Managing Files .....	137
Editing .....	138
Handling Resources .....	138
General .....	139
Dialog Boxes — General .....	139
Dialog Boxes — Alignment .....	140
Dialog Boxes — Spacing and Positioning .....	141
Dialog Boxes — Sizing .....	141
Menus .....	142
Bitmaps, Icons, and Cursors — General .....	142
Bitmaps, Icons, and Cursors — Colors .....	143
String Tables .....	144
Accelerator Tables .....	144
Class Wizard .....	144
Managing Symbols .....	145
Installing VBX Controls .....	145
Managing App Studio .....	145
Getting Online Help .....	146
Menu Reference .....	146
Managing Files .....	146
Editing .....	147
Creating and Editing Resources .....	147
Laying Out Dialog Boxes .....	148
Working with Images .....	148
Managing App Studio .....	149
Getting Online Help .....	149
Toolbar and Palette Reference .....	149
The App Studio Toolbar .....	150
The Properties Window Toolbar .....	150
The Dialog Editor Toolbar .....	150
The Graphics Editor Palette .....	151
The Control Palette .....	151
The Icon Editing Toolbar .....	151
The Cursor Editing Toolbar .....	152

Key Reference .....	152
Managing Files.....	152
Editing .....	152
Creating and Editing Resources .....	153
Managing Windows.....	154
Using the Properties Window .....	154
Editing Graphics .....	155
Using the Dialog Editor.....	156
Using the Menu Editor.....	157
Using the String Editor .....	157
Editing Resources as Binary Data.....	157
Getting Online Help.....	157
<b>Chapter 11 Property Page Reference .....</b>	<b>159</b>
Resource Property Page .....	159
Accel Table: Accel Properties — General.....	160
Bitmap Properties — General.....	161
Cursor Properties — General .....	161
Dialog Properties — General .....	162
Dialog Properties — Styles.....	163
Dialog: Check Box Properties — General.....	164
Dialog: Combo Box Properties — General.....	165
Dialog: Combo Box Properties — Styles.....	166
Dialog: Edit Box Properties — General.....	166
Dialog: Edit Box Properties — Styles .....	166
Dialog: Group Box Properties — General.....	168
Dialog: List Box Properties — General.....	168
Dialog: List Box Properties — Styles .....	169
Dialog: Picture Properties — General .....	170
Dialog: Pushbutton Properties — General.....	171
Dialog: Radio Button Properties — General.....	172
Dialog: Scrollbar Properties — General .....	172
Dialog: Text Properties — General .....	173
Dialog: User Control Properties — General .....	174
Dialog: VBX Control Properties — General.....	174
Dialog: VBX Control Properties — Styles .....	174
Icon Properties — General .....	175
Menu Properties — General.....	175
Menu: Menu Item — General .....	175
String Editor: String Properties — General.....	176

## Appendix

<b>Appendix A APSTUDIO.INI Settings</b> .....	<b>179</b>
Setting the Default .RC File Type.....	179
Setting the Number of Undo Levels .....	180
Using Default Dialog-Box Buttons.....	180
Setting the Default Magnification Factor.....	180
Describing Cursor Devices .....	180
Describing Icon Devices .....	180
<b>Index</b> .....	<b>181</b>

---

# Figures and Tables

## Figures

- Figure 1.1 App Studio Workspace
- Figure 1.2 App Studio Resource Browser
- Figure 1.3 Dragging and Dropping
- Figure 1.4 App Studio Properties Window
- Figure 1.5 Dialog Editor
- Figure 1.6 Menu Editor
- Figure 1.7 Accelerator Table Editor
- Figure 1.8 String Editor
- Figure 1.9 Graphics Editor
- Figure 1.10 Binary Data Editor
- Figure 1.11 Symbol Browser
- Figure 1.12 Using Class Wizard to Define a Message Handler
- Figure 1.13 Menu Example
- Figure 1.14 Toolbar Bitmap Generated by AppWizard
- Figure 1.15 Cursor Example: Dragging a Radio Button
- Figure 2.1 Using Drag and Drop to Copy Resources Between Files
- Figure 2.2 Set Includes Dialog Box
- Figure 2.3 Symbol Browser
- Figure 3.1 App Studio Controls
- Figure 3.2 New Dialog Box
- Figure 3.3 Dialog Editor Position Indicators
- Figure 3.4 Dragging a Control from the Control Palette
- Figure 3.5 Selecting Multiple Controls
- Figure 3.6 Sizing a Control
- Figure 3.7 Sizing the Drop-down Portion of a Combo Box
- Figure 3.8 Setting Tab Order
- Figure 3.9 Changing the Existing Tab Order
- Figure 3.10 Dialog Editor Toolbar
- Figure 3.11 Using Space Evenly Down
- Figure 3.12 Install Controls Dialog Box
- Figure 3.13 General Property Page for a VBX Control
- Figure 3.14 Styles Property Page for a VBX Control
- Figure 3.15 General Property Page for a User-Defined Control
- Figure 4.1 Menu Terminology

- Figure 4.2 Menu Editor New-Item Boxes
- Figure 4.3 Example: Moving a Cascading Menu to the Menu Bar
- Figure 5.1 Using the Accelerator Table Editor and Properties Window
- Figure 5.2 General Property Page for Accelerators
- Figure 6.1 Using the String Editor
- Figure 7.1 Image Editor Window and Graphics Palette
- Figure 7.2 Cross hairs for Defining the Selection
- Figure 7.3 Border Enclosing the Selected Area
- Figure 7.4 Cropping, Extending, Shrinking, and Stretching a Bitmap
- Figure 7.5 Moving the Image-Editor Pane Splitter
- Figure 7.6 Zooming in on the Actual-Size View
- Figure 7.7 New Icon Image Dialog Box
- Figure 7.8 Selectors for Screen Color and Inverse Color
- Figure 7.9 Typical Toolbar Graphic
- Figure 7.10 Grid Settings Dialog Box
- Figure 7.11 Image Editor Window with a Tile Grid
- Figure 7.12 Resizing the Toolbar Bitmap
- Figure 7.13 Selecting Button Images
- Figure 7.14 Moving the Selected Button Images
- Figure 7.15 Color Dialog Box
- Figure 7.16 Custom-Color Dialog Box
- Figure 8.1 The Binary Data Editor
- Figure 9.1 Add Class Dialog Box
- Figure 9.2 Import Class Dialog Box
- Figure 9.3 Defining a Message or Command Handler
- Figure 9.4 Add Variable Member Dialog Box
- Figure 9.5 Dialog Data Validation
- Figure 9.6 Password Dialog Box
- Figure 9.7 DDX/DDV Example: Add Member Variable Dialog Box
- Figure 9.8 DDX/DDV Example: Using Built-in DDV for a CString Variable
- Figure 9.9 DDX/DDV Example Defining a DDX Control Variable
- Figure 9.10 Repair Class Information Dialog Box
- Figure 9.11 Select Files Dialog Box for Generating a New .CLW File

## Tables

- Table 1.1 App Studio Toolbar New-Resource Buttons
- Table 3.1 Comparison of App Studio Support for VBX Controls and User-Defined Controls
- Table 6.1 Formatting and Special Characters in Strings
- Table 9.1 Types of Classes Created in ClassWizard

---

Table 9.2	User-Interface Objects and Associated Messages
Table 9.3	DDX Variable Types for the Value Property
Table 9.4	DDX Variable Types Defined with the Control Property
Table 9.5	DDV Variable Types





---

# Introduction

This manual contains information and procedures for working with App Studio, a powerful and easy-to-use Microsoft® Windows™ operating system resource editor. In addition to editing Windows resources, App Studio works together with the Microsoft Foundation Class Library and ClassWizard to let you connect your resources to code. With ClassWizard, you can also quickly define class member variables that make it easier to work with dialog-box controls.

Chapter 1 provides an overview of the App Studio user interface, including the main App Studio window, the resource browser window. Chapter 1 also contains important information on how to use the App Studio Properties window, a convenient way for setting the Windows styles of each resource and controlling the resource's appearance and behavior.

Chapter 2 discusses how to work with App Studio resource script files and details important issues to consider when updating existing resource files for use with App Studio. Also in Chapter 2 is information on working with resource identifiers in App Studio.

Chapters 3 through 8 offer specific step-by-step procedures for using App Studio's editing windows to create or modify Windows resources. Among the topics covered are the following:

- Using the dialog editor to create dialog-box and form-view templates, and using drag-and-drop to place dialog-box controls or transfer them from one dialog box to another.
- Using the menu editor to quickly create and modify menu resources in an editing window that lets you work with the menu in a menu-bar format similar in appearance to the finished application.
- Using the accelerator table editor to create and edit accelerator-table entries.
- Using the string editor to create, find, or modify strings in your application's string table.
- Using App Studio's full-featured graphics editor to edit bitmaps, icons, cursors and special-purpose images such as toolbars.

Chapter 9 introduces ClassWizard, a kind of programmer's assistant that makes it easier for you to do certain routine tasks such as creating new classes and gathering data from controls in a dialog box or form view.

Part 2 of this manual provides a reference to the resource properties that you can set in the App Studio Properties window and a complete reference to the App Studio menus and toolbars. This information (including additional information about App Studio dialog boxes) is also contained in online Help.

## Document Conventions

This book uses the following typographic conventions:

Example	Description
STDIO.H	Uppercase letters indicate filenames, segment names, registers, and terms used at the operating-system command level.
<b>char, _setcolor, __far</b>	Bold type indicates C and C++ keywords, operators, language-specific characters, and library routines. Within discussions of syntax, bold type indicates that the text must be entered exactly as shown.  Many functions and constants begin with either a single or double underscore. These are part of the name and are mandatory. For example, to have the <b>__cplusplus</b> manifest constant be recognized by the compiler, you must enter the leading double underscore.
<i>expression</i>	Words in italics indicate placeholders for information you must supply, such as a filename. Italic type is also used occasionally for emphasis in the text.
<b>[[option]]</b>	Items inside double square brackets are optional.
<b>#pragma pack {1   2}</b>	Braces and a vertical bar indicate a choice among two or more items. You must choose one of these items unless double square brackets ([[ ]]) surround the braces.
<code>#include &lt;io.h&gt;</code>	This font is used for examples, user input, program output, and error messages in text.
CL <b>[[option...]]file...</b>	Three dots (an ellipsis) following an item indicate that more items having the same form may appear.
<code>while() { . . . }</code>	A column or row of three dots tells you that part of an example program has been intentionally omitted.

---

<b>Example</b>	<b>Description</b>
CTRL+ENTER	<p>Small capital letters are used to indicate the names of keys on the keyboard. When you see a plus sign (+) between two key names, you should hold down the first key while pressing the second.</p> <p>The carriage-return key, sometimes marked as a bent arrow on the keyboard, is called ENTER.</p>
“argument”	<p>Quotation marks enclose a new term the first time it is defined in text.</p>
"C string"	<p>Some C constructs, such as strings, require quotation marks. Quotation marks required by the language have the form " " and ' ' rather than “ ” and ‘ ’.</p>
Color Graphics Adapter (CGA)	<p>The first time an acronym is used, it is usually spelled out.</p>



P A R T 1

# Using App Studio

Chapter 1	App Studio Overview . . . . .	3
Chapter 2	Working with Files and Symbols . . . . .	21
Chapter 3	Using the Dialog Editor . . . . .	39
Chapter 4	Using the Menu Editor . . . . .	61
Chapter 5	Using the Accelerator Table Editor . . . . .	69
Chapter 6	Using the String Editor . . . . .	75
Chapter 7	Using the Graphics Editor . . . . .	81
Chapter 8	Using the Binary Data Editor . . . . .	111
Chapter 9	Using ClassWizard . . . . .	113



---

## CHAPTER 1

# App Studio Overview

App Studio is a powerful, easy-to-use resource editor. You can:

- Edit both the appearance and behavior of common user-interface objects such as menus, dialog boxes, accelerator tables, and string tables.
- Quickly and easily incorporate any of the large number of VBX custom controls into your program.
- Quickly design and edit bitmaps, cursors, icons, and toolbars in the App Studio graphics editor.

However, App Studio is also more than just a resource editor. While working in App Studio with Microsoft Foundation Class Library resource files you can:

- Attach user-interface objects to code by using ClassWizard to quickly and easily define message-handling functions.
- Create class member variables that automatically use the Microsoft Foundation Class Library routines for gathering and validating dialog-box data.

If you're new to programming for Windows, see the "Understanding Windows Resources" section near the end of this chapter. It explains the role of resources in a Windows program and gives examples of the most common resources.

You can also use the sample icons, cursors, and toolbar buttons in the sample resource file supplied with Microsoft Visual C++™, COMMON.RES.



# App Studio Basics

Once you have installed App Studio on your machine as part of the Visual C++ installation process, you are ready to begin editing user-interface objects and other resources. For information on installing Visual C++, see Chapter 1 of the *Visual Workbench User's Guide*.

---

**Note** A mouse or other pointing device is required for many App Studio functions.

---

## Starting App Studio

To begin your App Studio session, use the Visual Workbench Tools menu to open your project's resource script (.RC) file for editing. You can create the menus, accelerators, dialog boxes, and other resources that make up the user interface of your Windows program.

► **To run App Studio:**

- From the Visual Workbench Tools menu, choose App Studio.  
App Studio opens your current project's .RC file.

For more detailed information on running App Studio, see "Opening a Visual C++ Resource File" in Chapter 2, page 23.

If you used AppWizard when you began to write your program, App Studio opens a resource file that provides you with not only the basic outline of a fully functioning Windows application, but with some basic resource building blocks as well. For more information on AppWizard, see the *Visual Workbench User's Guide*.

Figure 1.1 shows the App Studio resource browser as it appears at the beginning of a new project. In this illustration, several other important App Studio windows are open. The major parts of App Studio are described in the sections that follow.

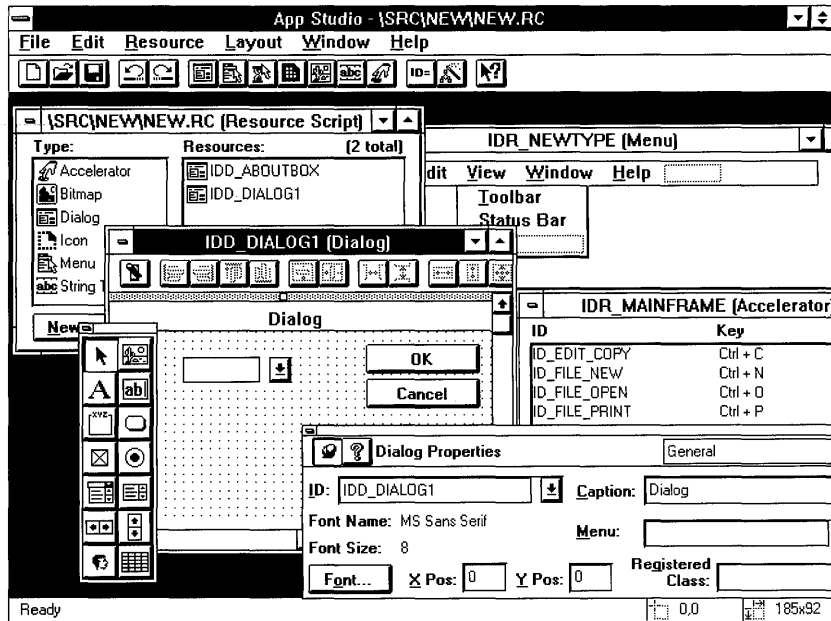


Figure 1.1 App Studio Workspace

## Using the Resource Browser Window

At the beginning of a resource editing session, the first App Studio window to appear is the resource browser (see Figure 1.2). Use the resource browser window to:

- View the resources in the file by type.
- Create new resources.
- Delete existing resources.
- Move resources from one file to another.
- View and edit a resource's basic properties in the Properties window.

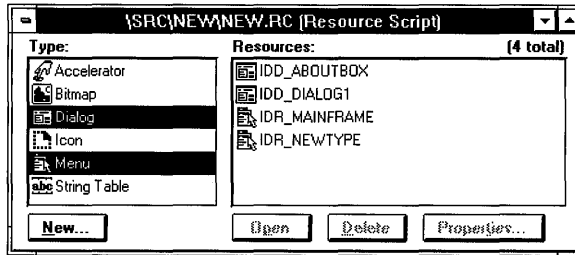


Figure 1.2 App Studio Resource Browser

► **To browse your application's resources:**

- Select a resource in the Type box.

The resources in your file that are of the type or types you selected are listed in the Resources box.

Select a single resource by clicking with the mouse. Select multiple contiguous resources by holding down the SHIFT key and clicking. Select multiple noncontiguous resources by holding down the CTRL key and clicking.

When a resource in the Resources box is selected but is not currently being edited in an App Studio editing window (or the editing window is minimized), you can view and edit the resource's basic properties in the Properties window. For more information on the Resource property pages and other property pages that appear in the Properties window, see Chapter 11.

## Creating a New Resource

There are several ways to create a new resource in App Studio. You can use the resource browser window or the App Studio toolbar.

► **To create a new resource:**

- On the App Studio toolbar, click the button you want.








Table 1.1 shows the toolbar's new-resource buttons.

–Or–

- In the resource browser window, click New, or from the Resources menu, choose New (CTRL+R). Then select the resource you want from the list of choices and click OK or press ENTER.

The editor window for the resource you chose appears.

**Table 1.1 App Studio Toolbar New-Resource Buttons**

Resource	Toolbar Button
Dialog box	
Menu	
Cursor	
Icon	
Bitmap	
String table	
Accelerator table	

For a complete reference to the App Studio toolbar, see Chapter 10.

## Editing an Existing Resource

► **To edit an existing resource:**

1. Move to the resource browser window.
2. In the Type box, select the type of resource you want.  
A list of the resources of that type in the current file appears in the Resources box.
3. In the Resources box, select the name or identifier of the resource you want to edit.
4. Choose Open.  
Or select the resource and press ENTER.  
Or double-click the resource.  
Or from the Resource menu, choose Open.

The editing window for the resource you chose appears.

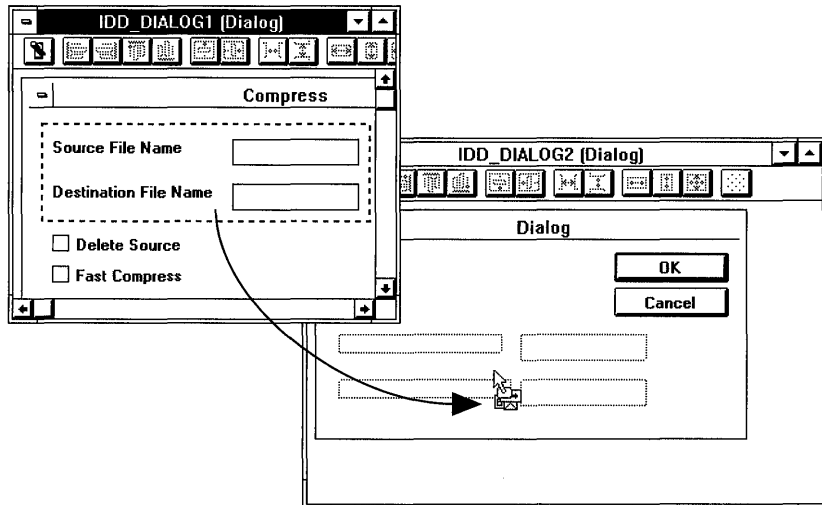
## Directly Manipulating User-Interface Objects

In many cases App Studio allows you to edit user-interface objects by directly manipulating them. For example, you can:

- Place controls in a dialog box by dragging them from the control palette into the dialog box (see Chapter 3).

- Rearrange menu items by moving them from one place to another with the mouse (see Chapter 4).

One of the most common direct-manipulation techniques is referred to throughout this manual as “drag and drop.” This means that you position an object by dragging it with the mouse or other pointing device and dropping it in its new location, as shown in Figure 1.3.



**Figure 1.3** Dragging and Dropping

- ▶ **To move or reposition an object using “drag and drop”:**
  1. Point to the object you want to move or position and hold down the left mouse button.
  2. While still holding down the button, drag the object to its new location.
  3. When the object is in position, drop it by releasing the button.

## Using Undo and Redo

Multiple-level undo and redo is a powerful tool for iterative design in App Studio. Undo reverses the effect of the last operation and redo reverses the effect of the last undo.

Normally you can undo your last ten actions, but you can change this number by changing the value in APSTUDIO.INI (see Appendix A, page 179). Most common operations in App Studio can be undone, including actions such as dragging and dropping that involve moving from one App Studio window to another.

► **To undo an operation:**



- On the App Studio toolbar, click the Undo button.  
Or from the Edit menu, choose Undo (CTRL+Z).

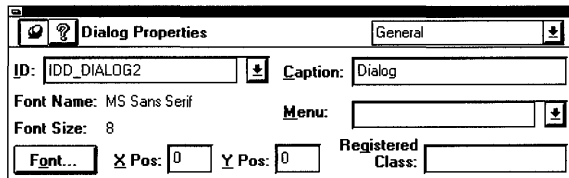
► **To redo an operation:**



- On the App Studio toolbar, click the Redo button.  
Or from the Edit menu, choose Redo (CTRL+A.)

## Using the Properties Window

The Properties window is one of the most important features of App Studio; it allows you to control the appearance and behavior of the resources you create. Figure 1.4 shows the Properties window as it appears when you first open a dialog box for editing.



**Figure 1.4** App Studio Properties Window

Once you use App Studio for a short time, you will become quite familiar with how the Properties window works. To get a head start, however, there are a few things you should know:

- The Properties window contains one or more property pages that apply to the resource or resource object that is currently selected.
- If the editing window for the selected resource is closed or minimized, only the Resource property page (showing basic information about the resource) is available. See page 159 for more information.
- You can control whether the Properties window stays visible while you are working in another window (for example, an editor window) or is dismissed once you switch to another window.
- Changes made on a property page take effect as soon as you make them. For an edit box, changes take effect as soon as you switch to another window or control.
- You can use App Studio's property pages and multiple-level undo as a design tool. Experiment with various property settings and then back them out with CTRL+Z if they're not what you want.

## Viewing Property Pages

Property pages divide resource properties into logical groupings so that they are easier to view and edit.

► **To display the Properties window:**

- From the Window menu, choose Show Properties.  
–Or–
- Choose one of the following shortcuts:
  - For resources with a caption, select the resource and start typing the caption.
  - Press ENTER when a resource is selected in an editing window.
  - Press ALT+ENTER at any time.
  - Choose the Properties command at the bottom of the resource browser window.

---

**Tip** You can display the Properties window quickly by double-clicking the object whose properties you want to edit.

---

► **To move from one property page to another in the Properties window:**

1. In the Resources box, select the resource whose properties you want to edit.
2. Display the Properties window if it is not currently displayed: from the Window menu, choose Show Properties.
3. In the property page box at the upper-right corner of the Properties window, choose the category of properties you want to edit from the list of property pages.



---

**Tip** With the Properties window displayed, you can quickly move from one property page to another using the PAGE UP and PAGE DOWN keys.

---

## Controlling the Properties Window

You can control the behavior of the Properties window to suit your working style or the nature of the resource editing task. Use the “pushpin” command button in the upper-left corner of the Properties window:

-  ■ When the button is in the down position, the Properties window stays visible even when you are working in another window. This is convenient if, during an editing session, you want to move back and forth frequently between setting properties and editing objects. Pressing ENTER after you change a value in the Properties window returns you to the editing window but leaves the Properties window visible. Pressing ESC cancels any changes you made and returns you to the editing window.
-  ■ When the button is in the up position, you can dismiss the active Properties window by pressing ENTER or ESC. This is useful if you want to concentrate on working in an editing window but need to bring up the Properties window briefly to change one or two values.

## Introduction to App Studio Editing Windows

This section presents a visual overview of the main App Studio editing windows and App Studio dialog boxes. For a complete description of each editor, see the appropriate chapter.

### Dialog Editor

Use the dialog editor to quickly create dialog boxes, place and arrange controls, and test the finished product. See Chapter 3.

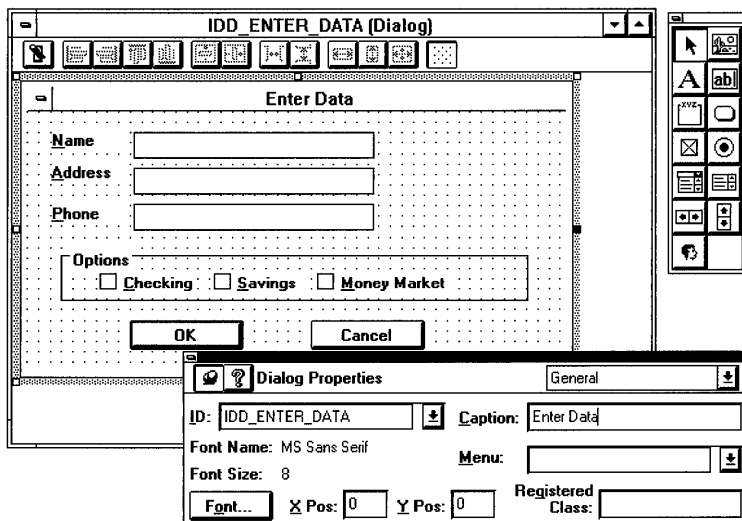


Figure 1.5 Dialog Editor



## Menu Editor

Use the menu editor to create and edit menu items by working directly with a menu bar that closely resembles the one in your finished application. See Chapter 4.

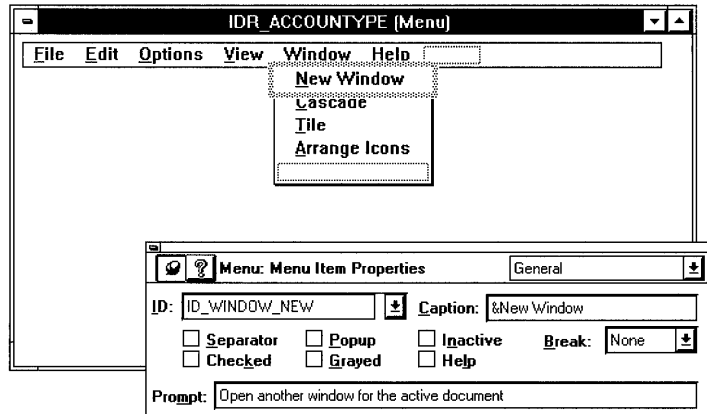


Figure 1.6 Menu Editor

## Accelerator Table Editor

Use the accelerator table editor to add, delete, change, or browse the accelerator-key assignments in your project. See Chapter 5.

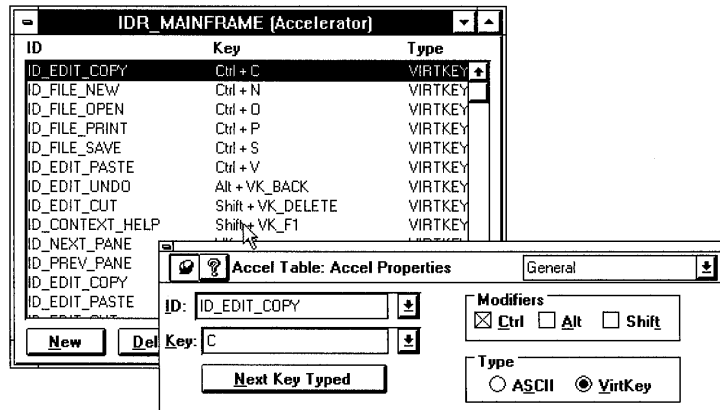


Figure 1.7 Accelerator Table Editor

## String Editor

Use the string editor to change or add to your program's standard string-table resource. See Chapter 6.

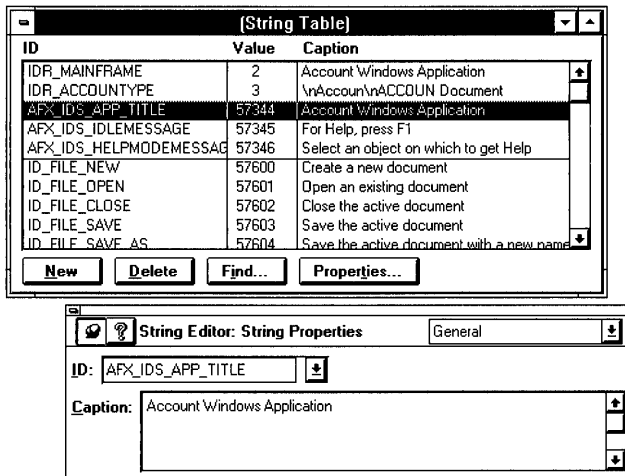


Figure 1.8 String Editor

## Graphics Editor

Use the graphics editor to edit your program's cursor, icon, and bitmap resources. See Chapter 7.

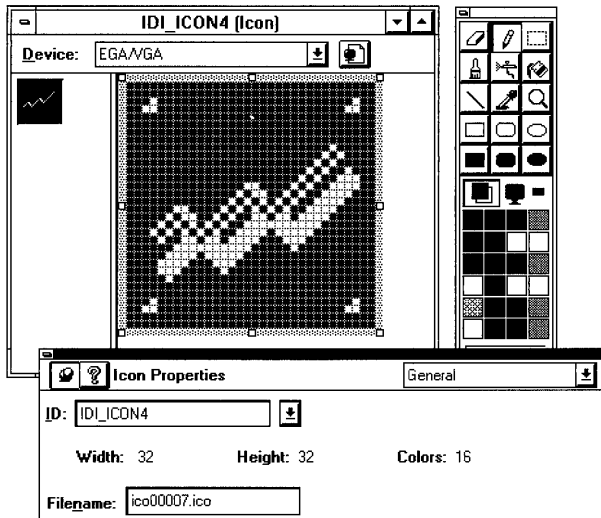


Figure 1.9 Graphics Editor

## Binary Data Editor

Use the binary data editor to edit an existing custom resource at the binary level in either hexadecimal or ASCII format. See Chapter 8.

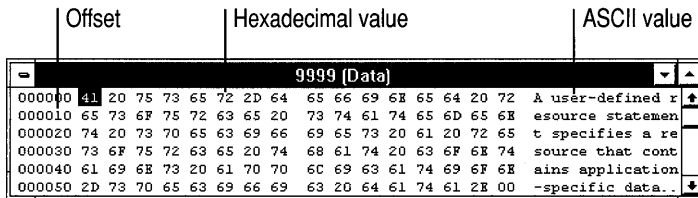


Figure 1.10 Binary Data Editor

## The Symbol Browser

Use the Symbol Browser to edit and browse existing identifiers or create new ones. See Chapter 2.

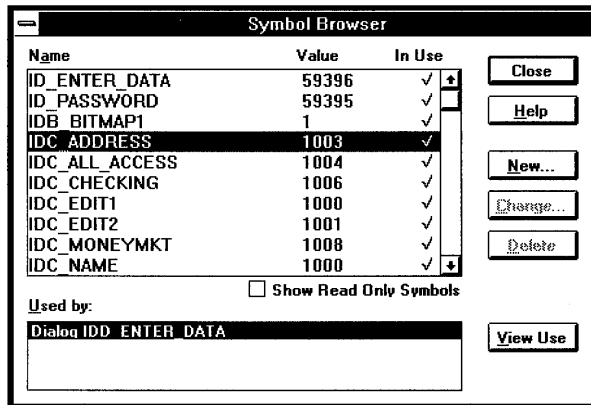


Figure 1.11 Symbol Browser

## Using ClassWizard

App Studio goes beyond just editing resources. You can use ClassWizard to connect resources to code. This section gives a brief overview of ClassWizard; for more information, see Chapter 9.

## Simplifying Message Handling

ClassWizard greatly simplifies the often tedious and time-consuming task of managing Windows message handling. You can browse the Windows messages appropriate to a given user-interface object and define message handlers for them (Figure 1.12).

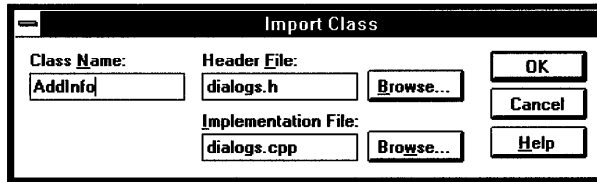


Figure 1.12 Using Class Wizard to Define a Message Handler

## Gathering and Validating Dialog-Box Data

The Microsoft Foundation Class Library has built-in routines that simplify gathering and validating the data associated with dialog-box controls.

When you are building a dialog box in App Studio, you can use ClassWizard to define member variables in the dialog-box class to take advantage of these built-in routines for Dialog Data Exchange (DDX) and Dialog Data Validation (DDV).

## Understanding Windows Resources

Resources are data objects that are separate from the main body of code in a Windows operating system program. Because of this, resources can be built and edited independently of the rest of the development process. This not only makes a project with a complex user interface easier to manage, but it also makes the application easier to translate into other languages.

Typically, resources are used to describe the contents and appearance of user-interface objects such as menus and dialogs. However, you can also define your own resource type if, for example, you have custom data that you want to make part of your program.

To most efficiently manage available memory, Windows usually leaves resources on disk until they are needed, although you can cause individual resources to be loaded at program startup to improve performance. Once loaded, a resource is usually placed in a “discardable” section of memory so that memory can be freed up for other tasks if needed. When the user-interface object is again required, it is loaded from disk.

The appearance and contents of resources are defined in a “resource script” (.RC) file. The .RC file is compiled into a binary form in a separate step when you build your program and is made a part of the executable file during linking.

## Predefined Windows Resources

The following predefined resource types are supported by built-in Windows library routines:

dialog box	menu
accelerator table	string table
bitmap	icon
cursor	

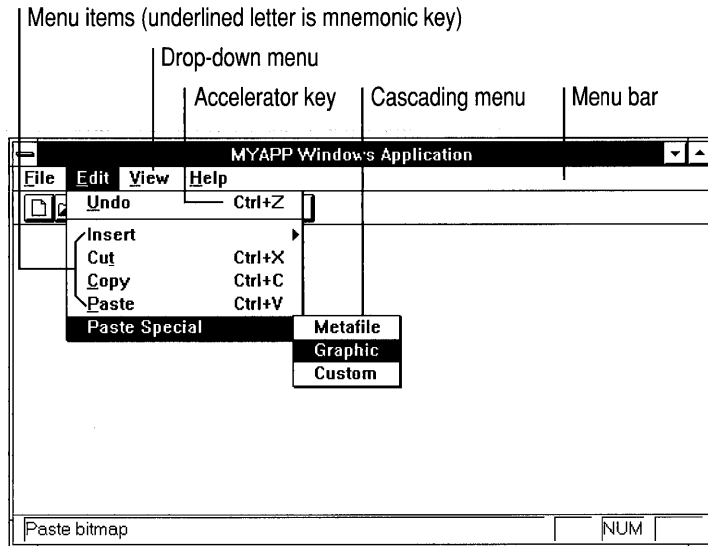
### Dialog Box

A dialog box is a window, usually a pop-up window under the control of the main program window, that allows your program to carry on a two-way conversation with the user. Dialog boxes are used to gather the information needed to comply with a user request (such as the name of a file to open).

App Studio allows you to construct dialog boxes by simply dragging controls from a palette and dropping them into place. You can also arrange controls easily with a variety of layout tools. For information on editing dialog boxes, see Chapter 3.

### Menu

A Windows menu is a list of program commands. Menus offer a way to group multiple commands into logical units, thus reducing the complexity of the user interface. Since these lists of commands drop down from a “menu bar” at the top of the main program window, they also offer a convenient way for the user to browse and select program commands with a pointing device. Figure 1.13 shows a program window with a menu bar and two levels of menus open.



**Figure 1.13** Menu Example

The App Studio menu editor allows you to create new menus by pointing to any empty menu cell and filling in the information in the Properties window. To rearrange menu items, you just drag them from one location to another with the mouse. For information on editing menus, see Chapter 4.

## Accelerator Table

An accelerator table contains a list of accelerator (shortcut) keys that can be used to execute a program command directly from the keyboard without having to pull down a menu and choose the command from a list. In Figure 1.13, the menu example, the Copy command is assigned to the key combination CTRL+C.

Normally, accelerators are associated with program commands that also appear on a menu. However, some applications use accelerator keys for seldom-used program commands that, if placed on a menu, would make the program's menu structure too complex.

For more information on editing accelerator tables, see Chapter 5.

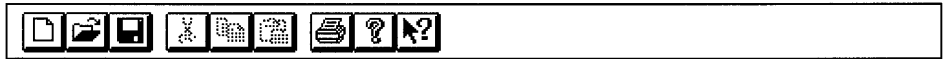
## String Table

A string table is a Windows resource type that allows you to store text strings that will be displayed as part of your user interface. Since this information is stored separately from the program source code, you can easily edit or translate program text without having to recompile the entire program.

The App Studio string editor gives you convenient access to all string resources and their associated attributes. For more information about the App Studio string editor, see Chapter 6.

## Bitmap

A bitmap is a graphical image stored in binary form; the basic element of the bitmap is the pixel, or picture element. One common use of bitmaps in programs for Windows is to define the appearance of the custom command buttons frequently used in toolbars or control palettes. For example, when you create a new application using AppWizard, a basic toolbar bitmap (Figure 1.14) is included as part of the application.



**Figure 1.14** Toolbar Bitmap Generated by AppWizard

The graphics editor in App Studio is full-featured and powerful. In addition to allowing general bitmap editing, it has special provisions for editing bitmaps for use with the **CToolBar** class in the Microsoft Foundation Class Library. For more information on creating toolbars and other bitmaps using the App Studio graphics editor, see Chapter 7.

## Icon

An icon is a unique graphical identifier that represents a program on the Windows desktop. An icon is used to launch a program from the Program Manager and also to represent a program window that has been minimized. Figure 1.9 shows an icon, which is a 32x32-pixel bitmap, being edited in the App Studio graphics editor.

For information on using App Studio to create or modify icons, see Chapter 7. For information on incorporating existing icons into your project, see Chapter 2, page 28.

## Cursor

A cursor is a bitmap, usually 32x32 pixels, that shows the position of the mouse or other pointing device on the screen. Programs use different cursors to show a change in the state or current action.

For example, in the App Studio dialog editor, when you use “drag and drop” (see page 8) to place a radio button in a dialog box, the cursor changes from an arrow to an arrow and a radio button (see Figure 1.15) as a visual reminder of what kind of object you are moving.



**Figure 1.15** Cursor Example: Dragging a Radio Button

For information on using the App Studio graphics editor to edit cursors, see Chapter 7. For information on incorporating existing cursor bitmaps into your project, see Chapter 2, page 28.

## App Studio Sample Resources

Visual C++ includes sample resources that you can use in your own application. These include:

- A large number of icons that represent common business and data-processing tasks.
- Several commonly used cursors that are not included as predefined Windows resources.
- A selection of toolbar-button bitmaps.

These resources are located in a file called `COMMON.RES` in the `\MSVCMFC\SAMPLES\APSTUDIO` directory on the drive where you installed Visual C++. Additional sample resources can be found in this directory.

- ▶ **To copy resources from `COMMON.RES` to your own resource script file:**
  1. Use App Studio’s File menu to open both your `.RC` file and `COMMON.RES` at the same time.
  2. Hold down the `CTRL` key and drag the resources you want from the `COMMON.RES` resource browser window to the resource browser window of your own application.



## Where to Go from Here

<b>For more information on</b>	<b>See</b>
Managing projects and working with resource files	Chapter 2
Using ClassWizard	Chapter 9
Editing specific resource types	The chapter covering the editor for that resource
User-interface object properties and styles	Chapter 11

# Working With Files and Symbols

App Studio supports many different ways of working with resources. You can:

- Load, save, and edit resource script (.RC) files.
- Load, save, and edit Windows executables (.EXE), dynamic link libraries (.DLL), and compiled resource (.RES) files.
- Use App Studio as a stand-alone editor for bitmaps, icons, or cursors.
- Import or export graphics resources to or from your current resource file.

In most cases, it is easiest to work with App Studio's single .RC file. However, App Studio can accommodate other ways of organizing a project. You can:

- Work with nested resource files and conditionally included resource files.
- Include shared or read-only identifiers (symbols) that can't be modified by App Studio.
- Include resources in your executable file (.EXE) that don't require editing (or that you don't want to be edited) during your current project (such as resources that are shared between several projects).
- Include resource types not supported by App Studio.

When you create a new resource or an object within a resource, App Studio assigns it an identifier, or symbol, consisting of a text string mapped to an integer value. You can change the symbol name or value in the Properties window when you create the resource, and you can view existing symbols and add new ones with the Symbol Browser as you work.

## Working with Files

App Studio lets you open the following types of files:

File Extension	Description
.RC	Resource script file
.RES	Compiled resource script file
.EXE	Executable file
.DLL	Dynamic-link library
.BMP, .DIB, .ICO, .CUR	Graphics files containing bitmaps, icons, or cursors

In addition, you can save an existing resource, executable file, or graphics file as one of the following:

- .EXE (for an existing executable)
- .RC
- .RES (a compiled resource file)
- .BMP, .DIB, .ICO, or .CUR (if you are working with a graphics file only and not a resource script file)

App Studio also works with several additional files during your resource editing session:

Filename	Description
RESOURCE.H	Header file generated by App Studio; contains symbol definitions.
<i>projectname</i> .APS	Binary version of the current resource file; used by App Studio for quick loading.
<i>projectname</i> .CLW	File containing information about the current project; used by ClassWizard.

The following sections cover the most common operations you'll need to set up your project and to manage resources and resource files.

## Creating a New Resource File

You can use App Studio to create a new resource script file or graphics resource file. You can also choose to include Microsoft Foundation Class Library support in the resource file you create. App Studio can also create a compiled resource (.RES) file, although the situations in which this is required are rare.

► **To create a new resource file:**

1. From the File menu, choose New.

The New dialog box appears.

2. Choose a file type from the list of choices: bitmap, icon, cursor, resource script file (.RC), or compiled resource file (.RES).
3. Set or clear the Use Microsoft Foundation Classes option as appropriate.

For information on features specific to Microsoft Foundation class resource files, see page 26.

–Or–



- Click the New File button on the App Studio toolbar.

A new resource script file with Microsoft Foundation Class Library support is created.

---

**Note** You can change the type of resource file created by changing the UseMfc setting in APSTUDIO.INI. See Appendix A for details.

---

## Opening a Visual C++ Resource File

This section provides two ways to start a resource editing session from Visual Workbench. The instructions assume that you have started Visual Workbench and created a basic set of new project files using AppWizard. For more information on AppWizard, see chapter 13 of the *Visual Workbench User's Guide*. For information on opening resource files not created with AppWizard or App Studio, see page 24.

► **To open a resource file from Visual Workbench:**

- From the Tools menu, choose App Studio.

This executes App Studio and opens the .RC file with the same base name as the current project.

–Or–

1. From the Options menu, choose Editor.

The Editor dialog box appears.

2. Under Source Files, select the Open RC Files Using App Studio option (normally turned off).
3. Click the Project Files button on the Visual Workbench toolbar and choose your project's .RC file from the list.

–Or–

1. From the File menu, choose Open.
2. Open the .RC file for your project.

Visual Workbench executes App Studio and loads the project resource file.

► **To open a resource file from App Studio:**

1. Start App Studio from the Windows Program Manager.
2. From the App Studio File menu, choose Open.
3. Open your project's .RC file for editing.

For information on a recommended file structure for Visual C++ projects, see the discussion of the files generated by the AppWizard in Chapter 13 of the *Visual Workbench User's Guide*.

## Reading Resource Files Not Created with App Studio or AppWizard

When you start a new Visual C++ product, you normally use AppWizard to create a set of basic starter files. These starter files include an App Studio-compatible .RC file that contains support for Microsoft Foundation Class Library features. However, you can also update existing .RC files for use with App Studio.

► **To update an existing resource file for use with App Studio:**

1. Make a backup copy of your existing .RC file.
2. Add the .RC file to your project using Visual Workbench (see Chapter 8 of the *Visual Workbench User's Guide*).
3. Open the file in App Studio according to the instructions in the previous section.

---

**Note** App Studio uses the include path set in Visual Workbench (from the Visual Workbench Options menu, choose Directories). In addition, relative include paths in an App Studio resource file must be based on the directory where the .RC file is currently located.

---

4. Save the App Studio version of the resource file.

Reading in and then saving resource files not created by AppWizard or App Studio has two results that are important to be aware of:

- App Studio makes several changes to how your resource files are organized so that you can work with all your resources in one place.
- Several resource-related features supported by the Microsoft Foundation Class Library are automatically available when your resource script file includes the library file AFXRES.H. See page 133 for instructions on how to manually add framework support to existing .RC files.

## Converting Existing Resource Files to App Studio Format

Saving a non-App Studio resource script file for the first time in App Studio has several important consequences:

- Resources contained in files that were added to your old .RC file with include statements are written back to disk as part of the main App Studio resource file. For example, if your old project had dialog boxes in a separate .DLG file, this separate file is no longer needed since App Studio moves the dialog boxes into the main .RC file. However, if you have resource files containing resources that you do not want to edit in App Studio, or that you want to continue to store in a separate resource file (such as a version resource file), move the resource to a separate file and add it back by using the File menu's Set Includes command. See page 28 for more information.
- Any symbol definitions included in your old .RC file are marked as read-only symbols by App Studio the first time you save the .RC file. To make the symbols available for modification and editing, remove them from the included header file and place them in the App Studio resource header, RESOURCE.H.
- Symbols that are defined with expressions rather than integers and used in the resource file are evaluated by App Studio, but they are then written to the App Studio symbols header (.H) file as simple integers. To preserve the expressions for calculated symbol values, include them in a read-only symbols header file (page 29).
- Conditional compilation statements in your old .RC file are evaluated by App Studio the first time the file is read in, but they are not written back to disk when you save the .RC file.

---

**Note** To preserve conditional compilation statements you should place the sections of your old .RC file containing these statements in a separate file, then include the file using the File menu's Set Includes command. See page 31 for more information.

---

- Comments in your old .RC file are not preserved.

In most cases, App Studio makes it easy and convenient to work with all your resources and symbols in a single App Studio file. However, App Studio also supports a nested resource file structure, conditionally included resource files, and expressions as symbol values if your project requires it. For more information see the section "Using Advanced Resource File Techniques" later in this chapter.

## Features Supported Only in Microsoft Foundation Class Library Resource Files

Normally when you build a Microsoft Foundation Class Library Windows application from scratch using AppWizard, you start by generating a basic set of files, including a resource file, that contain the core features of the Microsoft Foundation classes.

However, if you are editing a resource file for a Windows program that is not based on the Microsoft Foundation Class Library, the following features specific to the framework are not available in App Studio:

- ClassWizard (Chapter 9)
- Menu prompt strings (page 176)
- Support for VBX controls (page 55)
- List Contents for combo-box controls (page 165)

You can, however, add framework support to existing resource script files that do not have it.

► **To add framework support to .RC files that do not already have it:**

1. From the File menu, choose Set Includes.

The Set Includes dialog box appears.

2. In the Read-Only Symbol Directives box, replace the include statement for WINDOWS.H with the following:

```
#include "afxres.h"
```

3. You may also need to add additional include files using the Compile-Time Directives box. For more information, see the *Class Library Reference*.
4. Close the resource file and then reopen it for the changes to take effect.

## Copying Resources Between Files

The easiest way to copy resources from either an existing resource or an executable file to your current resource file is to have both files open in App Studio at the same time. Then use drag and drop to move items from one resource browser window to another (see Figure 2.1).

---

**Note** Visual C++ includes sample resource files that you can use in your own application. See page 19.

---

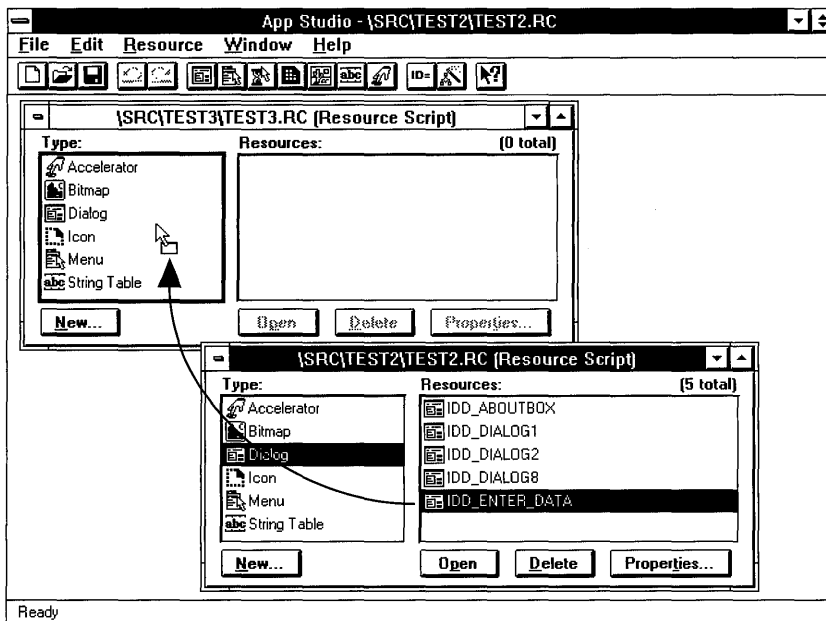
You can use the Resource menu's Import command to add bitmap, icon, or cursor files to your current App Studio project. You can save graphics resources to disk as separate files using the Export command.

► **To copy resources from one file to another:**

1. Open both files. Make sure both resource browser windows are visible.
2. In the resource browser window of the “from” file, select the resource you want to copy.
3. Hold down the CTRL key and drag the resource to the resource browser window of the “to” file.

Dragging the resource without holding down the CTRL key moves the resource rather than copies it.

**Note** To avoid conflicts with symbol names or values in the existing file, App Studio may change the transferred resource's symbol value, or symbol name and value, when you copy it to the new file.



**Figure 2.1** Using Drag and Drop to Copy Resources Between Files



► **To import a separate bitmap, icon, or cursor file into your current resource file:**

1. From the Resource menu, choose Import. The Import dialog box appears.
2. Choose the name of the .BMP, .ICO, or .CUR file you want to import. When you choose OK, the file is added to the current resource file.

---

**Tip** You can also copy a bitmap, icon, or cursor into your current resource file by dragging it from a File Manager window and dropping it into the App Studio resource browser window.

---

► **To save a bitmap, icon, or cursor as a separate file:**

1. Select the bitmap, icon, or cursor you want to export. App Studio exports the graphic selected in the resource browser window or the graphic in the currently active image editor window.
2. From the Resource menu, choose Export.
3. Enter a new filename for the bitmap, icon, or cursor, or press ENTER to accept the current filename.

The graphics file is saved to disk.

## Using Advanced Resource File Techniques

You can use the File menu's Set Includes command to modify App Studio's normal working arrangement of storing all resources in the project .RC file and all symbols in RESOURCE.H. (For more information on symbols, see "Working with Symbols" on page 31.) Figure 2.2 shows the Set Includes dialog box.

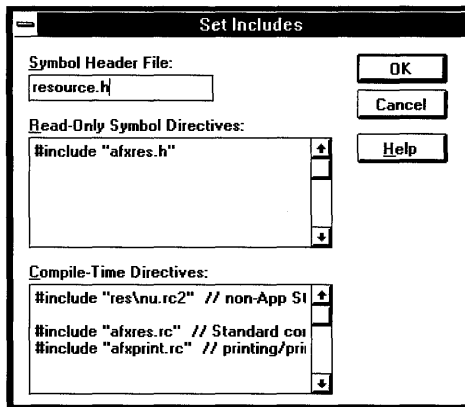


Figure 2.2 Set Includes Dialog Box

Use the Symbol Header File box to change the name of the header file where App Studio stores the symbol definitions for your resource file.

Use the Read-Only Symbol Directives box to include header files that contain symbols that should not be modified during an App Studio editing session. For example, you can use the Read-Only Symbol Directives box to include a symbol file that has been created to be shared among several projects. The Read-Only Symbol Directives box is also used to include Microsoft Foundation Class Library .H files.

Use the Compile-Time Directives box to include resource files that:

- Are created and edited separately from the resources in your main resource file.
- Contain compile-time directives, such as directives that conditionally include resources.
- Contain resources in a custom format.

The Compile Time Directives box is also used to include standard Microsoft Foundation Class Library resource files.

Once you've made changes to your resource file using the Set Includes dialog box, you need to close the file and then reopen it for the changes to take effect.

## Changing the Name of the Symbols Header File

Normally App Studio saves all symbol definitions to RESOURCE.H. However, you may need to change this include filename so that you can, for example, work with more than one resource file in the same directory.

### ► To change the name of the resource symbol header file:

1. From the File menu, choose Set Includes.  
The Set Includes dialog box appears.
2. In the Symbol Header File box, type the new name for the include file.
3. Choose OK.

## Using Shared (Read-Only) or Calculated Symbols

The first time App Studio reads a non-App Studio resource file, it marks all included header files as read-only. Subsequently, you can use the File menu's Set Includes command to add additional read-only symbol header files.

One reason you may want to use read-only symbol definitions is for symbol files that you plan to share among several projects.

You would also use included symbol files when you have existing resources with symbol definitions that use expressions rather than simple integers to define the symbol value. For example,

```
#define IDC_CONTROL1 2100
#define IDC_CONTROL2 (IDC_CONTROL1+1)
```

App Studio will correctly interpret these calculated symbols as long as:

- The calculated symbols are placed in a read-only symbols file.
- Your resource file contains resources to which these calculated symbols are already assigned.

► **To include shared (read-only) symbols in your resource file:**

1. From the File menu, choose Set Includes. The Set Includes dialog box appears.
2. In the Read-Only Symbol Directives box, use the `#include` compiler directive to specify the file where you want the read-only symbols to be kept. (The file should not be called RESOURCE.H, since that is the filename normally used by App Studio's main symbol header file.)
3. Place the symbols in the file you specified.

The symbols in files included in this way are evaluated each time you open your resource file, but they are not rewritten to disk by App Studio when you save your file.

---

**Important** What you type in the Read-Only Symbol Directives box is included in the resource file exactly as you type it. Make sure what you type does not contain any spelling or syntax errors.

You should use the Read-Only Symbol Directives box to include files with symbol definitions only, not resource definitions. App Studio places resources included with the Read-Only Symbol Directives box in the main resource file when it is saved, thus creating duplicate resource definitions.

---

## Including Resources From Other Files

Normally it is easy and convenient to work with App Studio's default arrangement of all resources in one .RC file. However, you can add resources in other files to your current project at compile time. Use the Set Includes dialog box's Compile Time Directives box.

There are several reasons to place resources in a file other than App Studio's main resource file:

- To include resources that have already been developed and tested and need no further modification.
- To include resources that are being used by several different projects, or that are part of a source code version-control system, and thus must exist in a central location where modifications will affect all projects.
- To include resources (such as RCDATA resources) that are in a custom format.
- To include statements in your resource file that execute conditionally at compile time using compiler directives such as `#ifdef` and `#else`. For example, your project may have a group of resources that are bracketed by `#ifdef _DEBUG` . . . `#endif` and are thus included only if the constant `_DEBUG` is defined at compile time.
- To include statements in your resource file that modify resource-file syntax by using `#define` to implement simple macros.

If you have sections in your existing resource files that meet any of these conditions, you should place the sections in one or more separate `.RC` files and include them in your project using the Set Includes dialog box. The `projectname.RC2` file created by AppWizard in the RES subdirectory of a new project is used for this purpose.

- ▶ **To include resource files that will be added to your project at compile time:**
  1. Place the resources in a resource script file with a unique filename. (Do not use `projectname.RC`, since this is the filename used for App Studio's main resource file.)
  2. From the File menu, choose Set Includes. The Set Includes dialog box appears.
  3. In the Compile-Time Directives box, use the `#include` compiler directive to include the new resource file in the main App Studio resource file.

The resources in files included in this way are made a part of your executable at compile time. They are not available for editing or modification when you are working on your project's main `.RC` file. You need to work on included resource files in a separate App Studio resource script file.

## Working with Symbols

A symbol is a resource identifier that consists of a text string (name) mapped to an integer value. Symbols provide a descriptive way of referring to resources and user-interface objects, both in your source code and while you're working with them in App Studio.

When you create a new resource or resource object, App Studio provides a default name for the resource (for example, `IDC_RADIO1`) and assigns a value to it. The name-plus-value definition is stored in the App Studio-generated file `RESOURCE.H`.

In working with symbols from within App Studio, you can:

- Change the symbol associated with a resource or object.
- Change a symbol's name or value in the Symbol Browser (if the symbol hasn't been used yet).
- Change a symbol's name in the Properties window (if the symbol is already in use by a single object).
- Use the Symbol Browser to browse existing symbols, add new symbols, and change or delete unused symbols.

---

**Note** When you are copying resources or resource objects from one `.RC` file to another, App Studio may change the transferred resource's symbol value, or symbol name and value, to avoid conflicts with symbol names or values in the existing file.

---

## Changing a Symbol or Symbol Name

When you create a new resource or resource object, App Studio assigns it a default name—for example, `IDD_DIALOG1`. Use the Properties window to change the default symbol name or to change the name of any symbol already associated with a resource.

► **To change a resource's symbol name:**

1. Select the resource and move to the appropriate property page (usually the General property page), or type `CTRL+Q` to move directly to the property page ID box.

–Or–

Open the Symbol Browser (page 34), select the symbol you want to change, and choose **Change**. When the **Change Symbol** dialog box appears, choose **View Use**.

This moves you to the property page of the resource where the symbol is used.

2. In the ID box, type a new symbol name or select from the list of existing symbols.
3. Press `ENTER` to accept the change. If you typed a new symbol name, App Studio assigns it a value automatically.

You can use the Symbol Browser to change the name of symbols not currently assigned to a resource. See “Changing Unassigned Symbols” on page 35.

## Changing a Symbol's Numerical Value

Usually you can let App Studio assign the numerical value associated with the symbol names you define. However, there may be times when you need to change the symbol value associated with a resource—for example, when you want a group of controls or a series of related strings in the string table to have sequential IDs.

For symbols already associated with a single resource, use the Properties window to change the symbol value. For symbols associated with more than one resource or object, make the changes directly in RESOURCE.H using a text editor.

► **To change a symbol value using the Properties window (applies only to symbols assigned to a single resource or object):**

1. Select the resource and move to the appropriate property page (usually the General property page), or type CTRL+Q to move directly to the property page ID box.

–Or–

Open the Symbol Browser (page 34), select the symbol you want to change, and choose Change. When the Change Symbol dialog box appears, choose View Use.

This moves you to the property page of the resource where the symbol is used.

2. In the property page ID box, type the symbol name followed by an equals sign and an integer. For example,

```
IDC_EDITNAME=5100
```

3. Press ENTER to accept the change.

The new value is stored in the symbol header file the next time you save. Only the symbol name remains visible in the ID box; the equals sign and value are not displayed after they are validated.

► **To change the numeric value of a symbol assigned to more than one resource or object:**

1. End your App Studio editing session by closing the current resource file or exiting App Studio.
2. Load RESOURCE.H into Visual Workbench and make the necessary changes.
3. Save RESOURCE.H.

The next time you open the project's .RC file in App Studio, App Studio uses the new symbol values.

**Note** While editing RESOURCE.H, take special care not to define duplicate symbols. App Studio can only detect duplicate symbols if they are created from within App Studio.

You can use the Symbol Browser to change the value of symbols not currently assigned to a resource. See “Changing Unassigned Symbols.”

## Managing Symbols with the Symbol Browser

As your application grows in size and sophistication, so do the number of resources and symbols that must be created. Large numbers of symbols scattered throughout several files can be difficult to keep track of. The Symbol Browser (Figure 2.3) simplifies symbol management by offering a central tool through which you can:

- Quickly browse existing symbol definitions to see the value of each symbol, a list of symbols being used, and the resources assigned to each symbol.
- Create new symbols.
- Change the name and value of a symbol that is not in use.
- Delete a symbol if it is not being used.
- With the View Use command, move quickly to the appropriate App Studio resource editor where the symbol is being used.

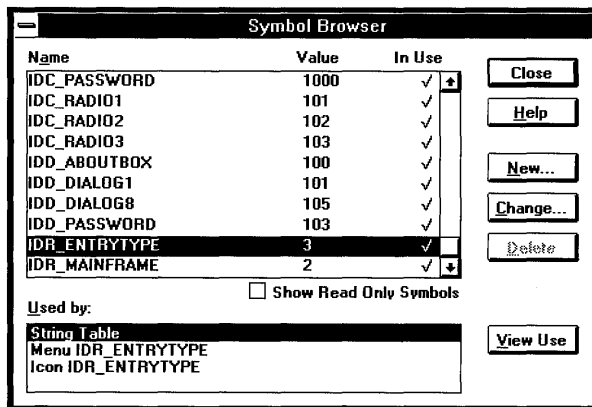


Figure 2.3 Symbol Browser

### ► To access the Symbol Browser dialog box:



- On the App Studio toolbar, click the Symbol Browser button.
- Or–
- From the Edit menu, choose Symbols (CTRL+I).

## Creating New Symbols

When you are beginning a new project, you may find it convenient to map out the symbol names you need before creating the resources they will be assigned to.

► **To create a new symbol using the Symbol Browser:**

1. In the Symbol Browser dialog box, choose New.  
The New Symbol dialog box appears.
2. In the Name box, type a symbol name.
3. Accept the symbol value assigned by App Studio or, in the Value box, type a new value.
4. Choose OK or press ENTER to place the new symbol into the symbol list. The symbol appears in alphabetical order.

If you enter a symbol name that already exists, a message box appears stating that a symbol with that name is already defined. You cannot define two or more symbols with the same name, but you can define different symbols with the same numeric value.

See the “Symbol Name Restrictions” and “Symbol Value Restrictions” sections at the end of this chapter for additional information on symbol names and values.

## Changing Unassigned Symbols

While in the Symbol Browser, you can edit or delete existing symbols that are not already assigned to a resource or object. You can change existing symbols that are in use in only one place by using the Change command to move to the appropriate resource’s property page or by moving to the property page directly. You cannot change read-only symbols. A check mark in the In Use column of the Symbol Browser indicates that the symbol is being used. If Show Read Only Symbols is selected, read-only symbols are also displayed. Editable symbols are displayed as bold text, and read-only symbols are displayed as normal text.

To change the name or value of a symbol already in use, see page 32.

► **To change an unassigned symbol using the Symbol Browser:**

1. Select the unassigned symbol you want and choose Change.  
The Change Symbol dialog box appears.
2. Edit the symbol’s name or value in the boxes provided, then press ENTER to accept the change.



- ▶ **To delete an unassigned symbol using the Symbol Browser:**
  - Select the unassigned symbol that you want to delete, and choose Delete (DEL).

---

**Note** Before deleting an unused symbol in a resource file, make sure it is not used elsewhere in the program or by resource files included at compile time.

---

## Opening the Resource Editor for a Given Symbol

When you are browsing symbols in the Symbol Editor, you may want more information on how a particular symbol is used. The View Use command provides a quick way to get this information.

- ▶ **To move to the resource editor where a symbol is being used:**
  1. In the Name box of the Symbol Browser, select the symbol you want.
  2. In the Used by box, select the resource type that interests you.
  3. Choose View Use.

The editor for the resource you selected appears.

## Symbol Name and Value Restrictions

There are several restrictions to be aware of when you use symbol names and values.

### Symbol Name Restrictions

All symbol names must be unique within the scope of the application. This prevents conflicting symbol definitions in the header files. Legal characters for a symbol name include A - Z, a - z, 0 - 9, and the underscore (\_). Symbol names cannot begin with a number and are limited to 247 characters. Symbol names are case insensitive, but the case of the first symbol definition is preserved.

Symbol names can be used more than once in your application. For example, if you are writing a data-entry program with several dialogs containing an edit box for a person's Social Security number, you may want to give all the related edit boxes a symbol name of IDC\_SSN. To do this, you can define a single symbol and use it as many times as needed.

While it is not required, symbol names are often given descriptive prefixes which indicate the kind of resource or object they represent. The Microsoft Foundation Class Library uses the following symbol naming conventions:

Category	Prefix	Use
Resources	IDR_	Accelerator or menu (and associated resources)
	IDD_	Dialog
	IDC_	Cursor
	IDI_	Icon
	IDB_	Bitmap
Menu items and commands	IDM_	Menu item
	ID_	Command
Controls and child windows	IDC_	Control
Strings	IDS_	String in the string table
	IDP_	String-table string used for message boxes

For more information on framework naming conventions, see Technical Note 20, which can be found in `MSVC\HELP\MFCNOTES.HLP`.

## Symbol Value Restrictions

In App Studio, a symbol value can be any integer expressed in the normal manner for `#define` preprocessor directives. Here are some examples of symbol values:

```
18
4001
0x0012
-3456
```

Symbol values for resources can be decimal numbers in the range from 0 to 32767. Symbol values for parts of objects (such as dialog box controls or individual strings in the string table) can be from 0 to 65534 or from -32768 to 32767.

---

**Note** Some number ranges are used by App Studio and the Microsoft Foundation Class Library for special purposes. For more information see Technical Note 20, which can be found in `MSVC\HELP\MFCNOTES.HLP`.

---

In App Studio, you cannot define a symbol value using other symbol strings. For example, the following symbol definition is not supported:

```
#define IDC_MYEDIT IDC_OTHEREDIT //not supported
```

You also cannot use preprocessor macros with arguments as value definitions. For example,

```
#define IDD_ABOUT ID(7) //not supported
```

is not a valid expression in App Studio regardless of what ID evaluates to at compile time.

If you have an existing file containing symbols defined with expressions, see page 29 for instructions on how to include the symbols as read-only symbols.

---

## CHAPTER 3

# Using the Dialog Editor

The App Studio dialog editor allows you to:

- Create a new dialog-box template.
- Place and arrange controls in a dialog-box template.
- Use custom controls.
- Test dialog boxes.

You can define message handlers and manage data gathering and validation using the Visual C++ ClassWizard.

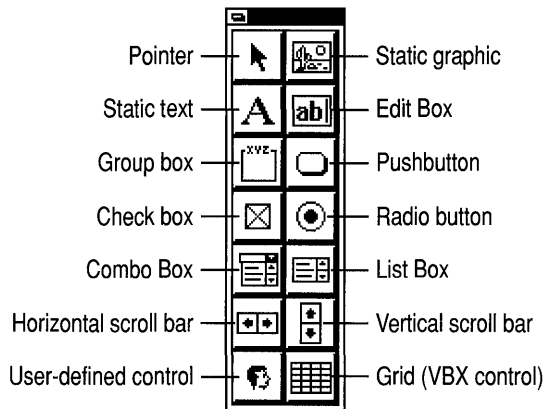
You can also use the App Studio dialog editor to create and edit templates used with form views and dialog bars. A form view is a template for a program window whose client area contains dialog-box controls. See page 59 for more information.

## Types of Controls

The dialog editor lets you create dialog boxes that include the following standard control types (Figure 3.1):

pushbutton	combo box	static text
radio button	group box	scroll bar
check box	static graphic	user-defined control
list box	edit box	

App Studio also comes with the Grid VBX control already installed.



**Figure 3.1** App Studio Controls

The dialog editor and Microsoft Foundation classes support VBX controls—custom controls created in a format compatible with both Microsoft Visual Basic™ and Visual C++. You can add VBX controls to the App Studio control palette, then incorporate them into your dialog boxes in the same way as standard controls. For more information, see page 55.

## Creating Dialog Boxes

This section contains procedures for:

- Creating a new dialog-box template.
- Adding controls.
- Working with more than one control.
- Moving, deleting, and copying controls.
- Resizing controls.
- Modifying the tab order of controls in a dialog box.
- Defining dialog-box keyboard shortcuts.

## Opening New or Existing Dialog Boxes

- ▶ **To create a new dialog-box template:**



- On the App Studio toolbar, click the New Dialog Box button.
- Or–
- In the resource browser window, click New, or from the Resource menu, choose New (CTRL+R). Then select Dialog from the list of choices and click OK or press ENTER.

The dialog editor window appears (Figure 3.2). Press ENTER to move to the property page and type a name for the new dialog box. You can use the Properties window at any time to change the dialog box's properties.

When the dialog box is created, App Studio assigns it a unique symbol name and value. If you need to change the symbol value, you can use the ID box on the General property page. Type the symbol name followed by an equals sign and a new value. For example,

```
IDD_DIALOG1=1001
```

For more information on the property page items, see the “Property Page Reference,” Chapter 11, or choose the Property page Help button.

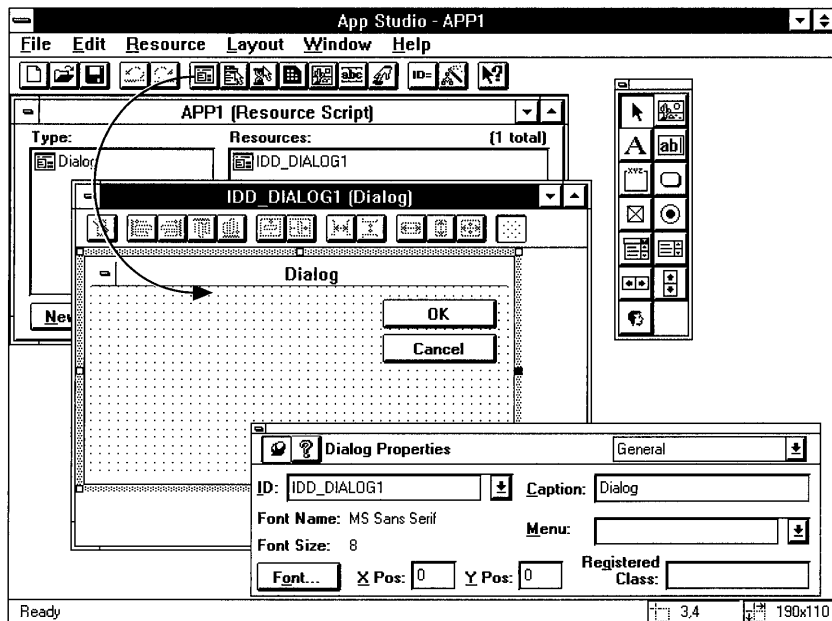


Figure 3.2 New Dialog Box

► **To edit an existing dialog box:**

1. Move to the resource browser window.
2. In the Type box, select Dialog.

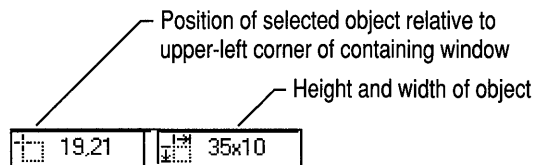
A list of the dialog resources in the current file appears in the Resources box.

3. In the Resources box, select the name or identifier of the dialog box you want to edit.
4. Choose Open.
  - Or select the resource and press ENTER.
  - Or double-click the resource.
  - Or from the Resource menu, choose Open.

The dialog editor window appears.

## Dialog Box Coordinates

The location, height, and width of the current control is displayed in the lower-right corner of the App Studio status bar (Figure 3.3). When more than one control is selected, the position indicators show the position of the dominant control (the control with solid sizing handles). When the dialog box is selected, the status bar displays the position of the dialog box and its height and width.



**Figure 3.3 Dialog Editor Position Indicators**

The location and size of a dialog box, as well as the location and size of controls within it, are measured in dialog box units (DLUs). A DLU is based on the size of the dialog-box font, normally 8-point MS Sans Serif. A horizontal DLU is the average width of the dialog-box font divided by four. A vertical DLU is the average height of the font divided by eight.

## Adding Controls

You add controls to a dialog box by selecting the control you want from the control palette. When displayed, the palette stays positioned above other open windows on your desktop.

► **To show or hide the control palette:**

- Press F2 to display or hide the control palette.
- Or–
- From the Window menu, choose Show Control Palette or Hide Control Palette.

The fastest way to add controls to a dialog box, reposition existing controls, or move controls from one dialog box to another, is to use the drag and drop method described in Chapter 1 (page 8). When you add a control to a dialog box with drag and drop, the control is given a standard height appropriate to that type of control.

You can also add a new control by clicking the control-palette button for the control you want and:

- “Drawing” the control in the dialog box. This is a good method when you want to specify the initial size of the object.
- Clicking in the dialog box at the location you want. This is an alternative method to dragging and dropping.

When you add a control to a dialog box or reposition it, its final placement may be determined by whether or not you have Snap to Grid turned on. For information about Snap to Grid and other placement and alignment tools, see the section on Arranging Controls beginning on page 51.

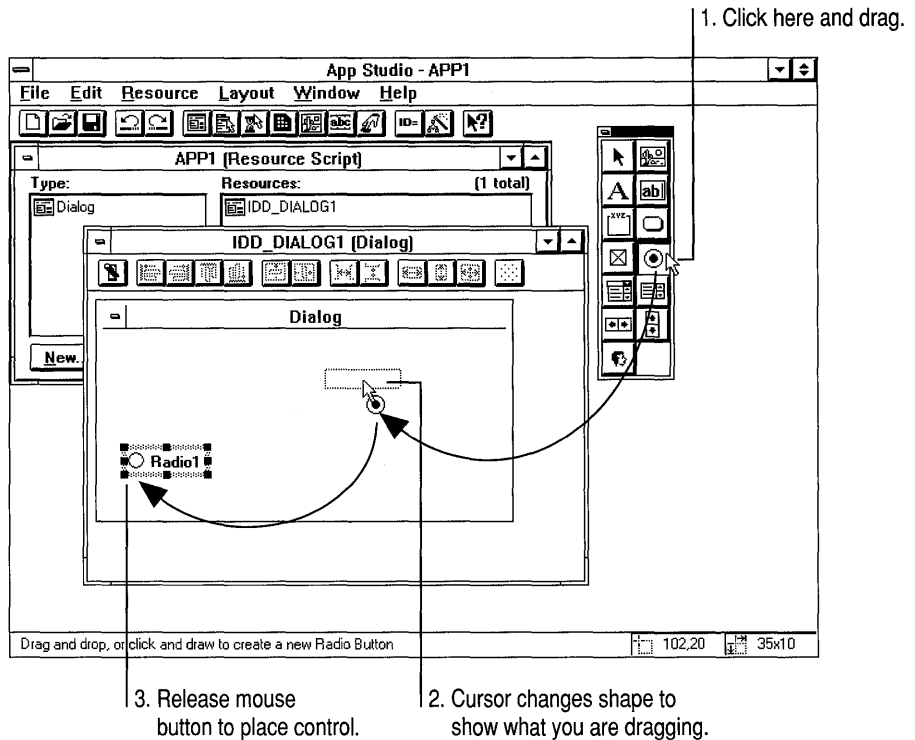
► **To add a control to a dialog box using drag and drop:**

1. Drag a control from the control palette to the dialog editor window (Figure 3.4).  
As you drag the control into the dialog box, a dotted outline of the control indicates its position.
2. When the dotted outline of the control is in the position you want, release the mouse button. The control is given a standard size appropriate to that type of control.

► **To add controls to a dialog box using point and click:**

1. On the control palette, click the button for the control you want.  
Or to add multiple controls of the same type, hold down the CTRL key, and click the control you want.
2. Move the mouse pointer to the dialog box and click at the position(s) you want.  
The control is given a standard size appropriate to that type of control.
3. Press ESC when you are finished placing controls.





**Figure 3.4 Dragging a Control from the Control Palette**

► **To add a control by “drawing” it with the pointing device:**

1. On the control palette, click the control you want.  
Or to add multiple controls of the same type, hold down the CTRL key, and click the control you want.
2. Place the pointer where you want the upper-left corner of the control to be located.
3. Hold down the left mouse button.
4. Move the pointer to the right and down; a dotted outline of the control appears.
5. When the control is the size you want, release the mouse button.
6. Press ESC when you are finished placing controls.

When you have added a control to the dialog box, you can change its caption or any other of its properties in the Properties window.

## Selecting Controls

To move, copy, delete, or align controls, you select them and then perform the operation you want. In most cases, you need to select more than one control to use the sizing and alignment tools on the dialog-editor toolbar.

When a control is selected, it has a shaded border around it with solid (active) or hollow (inactive) “sizing handles,” small squares that appear in the selection border.

When you are sizing or aligning multiple controls, App Studio uses the “dominant control” to determine how the other controls are sized or aligned. When multiple controls are selected, the dominant control has solid sizing handles; all the other selected controls have hollow sizing handles. You can set the dominant control by holding down the CTRL key while clicking that control.

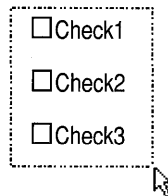
### ► To select a control:

- Point to the control you want and click. The currently selected object (control or dialog box) is deselected.
- Or–
- Use TAB to move forward or SHIFT+TAB to move backward through the controls in the dialog box.

### ► To select more than one control:

1. From the control palette, select the pointer tool.
2. Hold down the left mouse button and drag to draw a selection box around the controls you want to select (Figure 3.5). Controls partially outside the selection box are not selected.

When you release the mouse button, all controls inside the selection box are selected.



1. Drag the mouse pointer down and right, drawing a box around the controls you want to select.



2. Release the mouse button and the controls are selected.

**Figure 3.5** Selecting Multiple Controls

Once you have selected one or more controls, you can remove or add individual controls without disturbing the selection as a whole.

- ▶ **To remove from or add to an existing selection:**
  - Hold down the SHIFT key and click the control you want to remove from or add to the existing selection.

## Moving Controls

You can use the following procedures to move one or more controls from one location to another in a dialog box or from one dialog box to another. If Snap to Grid is on (see page 51), the control snaps to the alignment grid. For information on other ways to align multiple controls, see page 52.

- ▶ **To move a control from one location to another in a dialog box:**
  - Drag the control to its new location.  
–Or–
  - For a single control, select the control and use the arrow keys to move the control one DLU at a time.
- ▶ **To move a control from one dialog box to another:**
  - If both dialog boxes are visible, drag the control to its new location. (Hold down the CTRL key while dragging if you just want to copy the control.)  
–Or–
  - Use the Edit menu's Cut and Paste commands. The control is placed in the same position as in the original dialog box.

---

**Note** In rare cases, you may need to place a control outside a dialog box. To do this, hold down the ALT key while dragging the control.

---

## Deleting and Copying Controls

- ▶ **To delete a control:**
  1. Select the control.
  2. From the Edit menu, choose Cut (CTRL+X) or Delete (DEL).
- ▶ **To copy a control:**
  - Drag the control while holding down the CTRL key.  
–Or–
  - Use the Edit menu's Copy (CTRL+C) and Paste (CTRL+V) commands.

When you paste a control into a new dialog, it is placed in the same position it was in the old dialog.

## Sizing Individual Controls

Use the sizing handles to resize a control. When the mouse cursor is positioned on a sizing handle, it changes shape to indicate the direction in which the control will be resized (see Figure 3.6). Active sizing handles are solid; if a sizing handle is hollow, the control cannot be resized along that axis. For information on sizing multiple controls, see page 54.

When you change the size of a control, its final shape may be affected by whether or not you have Snap to Grid turned on (see page 51).

► **To size a control:**

1. Click the control or select it with the TAB key.
2. Use the sizing handles to change the size of the control:
  - Sizing handles at the top and sides change the horizontal or vertical size.
  - Sizing handles at the corners change both horizontal and vertical size.

–Or–

Use the SHIFT key plus the arrow keys to resize the control one DLU at a time.

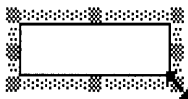


Figure 3.6 Sizing a Control

## Sizing a Control to Fit Its Caption

You can automatically change the size of a control so that it is the appropriate size for its text caption.

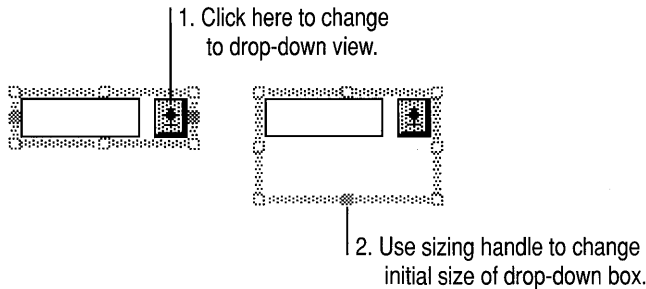
► **To resize a control to fit its caption:**

1. Select the control.
2. From the Layout menu, choose Size to Content (F7).

## Sizing Combo Box Drop-downs

When you select a drop-down or drop-list combo box to size it, only the right and left sizing handles are active (Figure 3.7). Use these handles to set the width of the combo box as it is initially displayed.

You can also set the vertical size of the drop-down portion of the combo box.



**Figure 3.7** Sizing the Drop-down Portion of a Combo Box

- ▶ **To set the size of the combo box drop-down area:**
  1. Click the drop-down arrow at the right of the combo box (Figure 3.7).

The outline of the control changes to show the size of the combo box with the drop-down area extended.
  2. Use the bottom sizing handle to change the initial size of the drop-down area.
  3. Click the drop-down arrow again to close the drop-down portion of the combo box.

## Changing the Tab Order

Tab order is the order in which the TAB key moves the input focus from one control to the next within a dialog box. Usually the tab order proceeds from left to right in a dialog box, and from top to bottom. The Tabstop option on the control's General property page (normally set) determines if a control actually receives input focus or not.

Even controls that do not have the Tabstop property set need to be part of the tab order. This can be important, for example, when you define mnemonics for controls that do not have captions. Static text that contains a mnemonic for a related control must immediately precede the related control in the tab order.

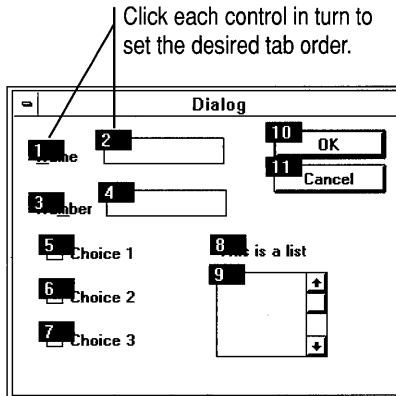
---

**Note** If your dialog box contains overlapping controls, changing the tab order may change the way the controls are displayed. Controls that come first in the tab order are always displayed on top of any overlapping controls that follow them in the tab order.

---

► **To change the tab order for all controls in a dialog box:**

1. From the Layout menu, choose Set Tab Order (CTRL+D).  
A number at the upper left of each control shows the current tab order.
2. Set the tab order by clicking each control in the same order you want the TAB key to follow (Figure 3.8).
3. Press ENTER to exit Set Tab Order mode.



**Figure 3.8** Setting Tab Order

To change the existing tab order, you usually need to change the selected control first, as explained in the following procedure. The selected control determines the number of the control you click next. For example, if you are in Set Tab Order mode and control number 1 is selected, the next control you click is set to number 2.

► **To change the existing tab order:**

1. From the Layout menu, choose Set Tab Order (CTRL+D).
2. Change the selected control. To do this, hold down the CTRL key and click the control *prior* to the one where you want the changed order to begin.

For example, if you want to change the order of controls 7 through 9, select control 6 first (see Figure 3.9).

---

**Note** To set a specific control to number 1 (first in the tab order), double-click the control.

---

3. Reset the tab order by clicking the controls in the order you want the TAB key to follow.
4. Press ENTER to exit Set Tab Order mode.

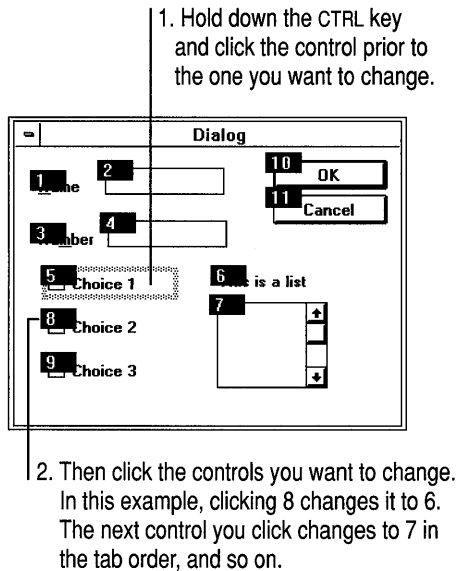


Figure 3.9 Changing the Existing Tab Order

## Defining Dialog Box Keyboard Access

Normally keyboard users move the input focus from one control to another in a dialog box with the TAB and arrow keys. However, you can define a mnemonic key that allows users to choose a control by pressing a single key.

---

**Note** All the mnemonics within a dialog box should be unique.

---

► **To define a mnemonic key for a control with its own visible caption (pushbuttons, check boxes, and radio buttons):**

1. Select the control and move to the General property page.
2. In the Caption box, type an ampersand (&) in front of the letter you want as the mnemonic for that control.

An underline appears in the displayed caption to indicate the mnemonic key.

► **To define a mnemonic for a control without a visible caption:**

1. Make a caption for the control by using a static text control. In the static text caption, type an ampersand (&) in front of the letter you want as the mnemonic.
2. Make sure the static text control immediately precedes the control it labels in the tab order.

## Arranging Controls

App Studio provides layout tools that align and size controls automatically. For most tasks, you can use the dialog-editor toolbar (Figure 3.10). All commands are also available on the Layout menu and most have keyboard equivalents (see Chapter 10).

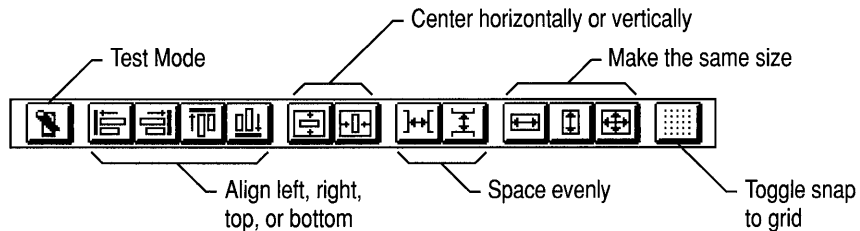


Figure 3.10 Dialog Editor Toolbar

Many layout commands are available only when more than one control is selected. For information on selecting more than one control, see page 45.

### The Dominant Control

When you are resizing or aligning multiple controls, App Studio uses the dominant control to determine how the other controls are sized or aligned. When multiple controls are selected, the dominant control has solid sizing handles; all the other selected controls have hollow sizing handles.

- ▶ **To change the dominant control when more than one control is selected:**
  - Hold down the CTRL key and click the control you want to influence the size or location of the others.
 

The sizing handles change from hollow to solid. All further resizing or alignment is based on this control.

## Using Snap to Grid

When you are placing or arranging controls in a dialog box, you can use the layout grid for more precise positioning. When the grid is turned on, controls appear to “snap to” the dotted lines of the grid as if magnetized. You can turn this “snap to grid” feature on and off and change the size of the layout grid cells.

- ▶ **To turn Snap to Grid on and off for the currently active dialog editor window:**



- On the dialog-editor toolbar, click Snap to Grid (CTRL+G).



- ▶ **To turn Snap to Grid on or off for all dialog editor windows:**
  1. On the Layout menu, choose Grid Settings. The Grid Settings dialog box appears.
  2. Turn the Snap to Grid check box on or off.  
You can still control Snap to Grid in individual dialog editor windows using the Snap to Grid button on the dialog-editor toolbar.
- ▶ **To change the size of the layout grid:**
  1. On the Layout menu, choose Grid Settings. The Grid Settings dialog box appears.
  2. Enter the height and width in DLUs for the cells in the grid. The minimum height or width is 4 DLUs. For more information on DLUs, see page 42.

## Aligning Controls

Once controls are in place, App Studio offers a variety of ways to regularize their position. You can:

- Align a group of controls along their left, right, top, or bottom edges.
- Align a group of controls on their center, either horizontally or vertically.
- Even the spacing between a group of three or more controls.
- Center one or more controls in the dialog box, vertically or horizontally.
- Automatically give pushbuttons a standard position along the bottom or on the right of the dialog box.

### Aligning

- ▶ **To align controls:**
  1. Select the controls you want to align.
  2. Make sure the correct dominant control is selected (see page 45). The final position of the group of controls depends on the position of the dominant control.
  3. Choose one of the following tools on the toolbar:



- **Align Left**—aligns the selected controls along their left side (CTRL+LEFT ARROW).



- **Align Right**—aligns the selected controls along their right side (CTRL+RIGHT ARROW).



- **Align Top**—aligns the selected controls along their top edges (CTRL+UP ARROW).



- **Align Bottom**—aligns the selected controls along their bottom edges (CTRL+DOWN ARROW).

- ▶ **To align controls on their center, vertically or horizontally:**
  1. Select the controls you want to center.
  2. Make sure the correct dominant control is selected (see page 45). The final position of the group of controls depends on the position of the dominant control.
  3. From the Layout menu, choose Align Vert. Center (F9) or Align Horiz. Center (SHIFT+F9).

## Making Spacing Even

- ▶ **To even the spacing between controls:**
  1. Select the controls you want to rearrange.
  2. Choose one of the following tools on the toolbar:



- **Space Evenly Across (ALT+LEFT ARROW or ALT+RIGHT ARROW)**

Controls are spaced evenly between the leftmost and the rightmost control selected.



- **Space Evenly Down (ALT+UP ARROW or ALT+DOWN ARROW)**

Controls are spaced evenly between the topmost and the bottommost control selected (Figure 3.11).

1. Select the controls and click the Space Evenly Down button on the tool bar.

2. Intervals between top and bottom controls are evened out.

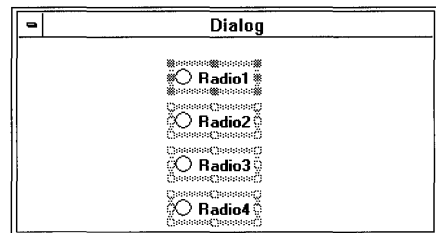
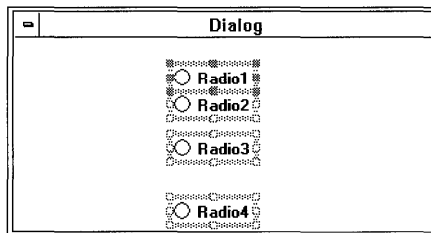


Figure 3.11 Using Space Evenly Down

## Centering in the Dialog Box

- ▶ **To center controls in the dialog box:**
  1. Select the control or controls you want to rearrange.
  2. Choose one of the following tools on the toolbar:



- **Center Vertical (CTRL+F9)**

Controls are centered vertically in the dialog box.



- **Center Horizontal (SHIFT+CTRL+F9)**

Controls are centered horizontally in the dialog box.

## Arranging Pushbuttons

► **To arrange pushbuttons along the right or bottom of the dialog box:**

1. Select one or more pushbuttons.
2. From the Layout menu, choose Arrange Buttons, then choose Right or Bottom.

The selected buttons are positioned in a standard arrangement along the bottom or right side of the dialog box. If controls other than pushbuttons are selected, their position is not affected.

## Resizing Controls

You can resize a group of controls based on the size of the dominant control. You can also resize a control based on the dimensions of its caption text.

► **To make controls the same width, height, or size:**

1. Select the controls you want to resize.
2. Make sure the correct dominant control is selected (see page 45). The final size of the controls in the group depends on the size of the dominant control.
3. Choose one of the following tools on the toolbar:



- Make Same Width (CTRL+MINUS SIGN)
- Make Same Height (CTRL+BACKSLASH)
- Make Same Size (CTRL+EQUAL SIGN)

## Using Custom Controls

A custom control is a special-format dynamic-link library (DLL) or object file used to add additional features and functionality to the user interface of the Windows operating system. A custom control can be a variation on an existing Windows dialog-box control (for example, a text box suitable for use with Windows for Pen Computing) or a totally new category of control.

App Studio supports VBX controls, which are enhanced custom controls with a programming interface supported by both Microsoft Visual Basic and the Microsoft Foundation Class Library. You can also use App Studio to place and arrange user-defined controls—controls created in other formats. The advantage of VBX controls over other control formats is that it is much easier to work with the appearance and behavior of VBX controls while you are designing the dialog box.

For information on the VBX controls available from Microsoft and other vendors, see the *Direct Access for Microsoft C/C++* catalog included with your copy of Visual C++.

User-defined controls are controls written in any other format. This includes the format of the custom-control examples in the Windows Software Development Kit (SDK) documentation.

## Using VBX Controls

You can add VBX controls to the App Studio control palette with the File menu's Install Controls command. Then use VBX controls to create dialog boxes and control bars just as you would any of the standard controls. App Studio comes with one VBX control already installed: the Grid control.

When you install VBX controls, information about them is stored in the App Studio initialization file APSTUDIO.INI, and the controls are reloaded each time you start App Studio.

Working with VBX controls in App Studio requires a resource script (.RC) file that includes Microsoft Foundation Class Library support. For more information on programming with VBX controls, see the *Class Library User's Guide* Chapter 17.

---

**Note** If you are creating an application that uses VBX controls, make sure that your application's setup program places the VBX controls in the user's \WINDOWS\SYSTEM directory.

---

Once a VBX control is installed on the control palette, you can use it like any other control. You can:

- Drag the control from the control palette to a dialog box.
  - View and change all the properties associated with the control.
  - Edit the control's message map and data map with ClassWizard (see Chapter 9).
  - Simulate the control's behavior in App Studio test mode (page 59).
- **To add a VBX control to the App Studio control palette:**
1. From the File menu, choose Install Controls. The Install Controls dialog box appears (see Figure 3.12).
  2. In the Control Filename box, type the name of the custom-control file, or use the Drives and Directories boxes to find the file you want. A single .VBX file can contain one or more VBX controls.
  3. Choose OK to add the control and to dismiss the dialog box.

–Or–

Use the Install command to add multiple VBX files, and then choose Close to dismiss the dialog box.

An icon representing each control installed appears on the App Studio control palette.

1. Select a VBX control to install.

2. Choose OK to add it to the list of installed files, or choose Install to add multiple files.

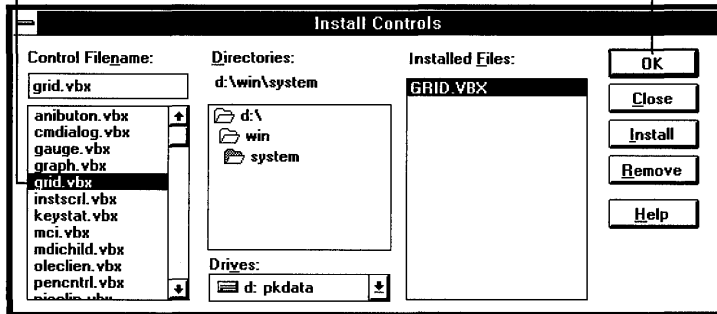


Figure 3.12 Install Controls Dialog Box

► **To delete a VBX control from the App Studio control palette:**

1. From the File menu, choose Install Controls. The Install Controls dialog box appears.
2. In the Installed Files box, select the name of the control file you want to delete.
3. Choose Remove, then choose Close.

## Editing VBX Control Properties

You can edit VBX control properties using the Properties window. Figure 3.13 shows the General property page, which allows you to set such common control properties as captions or identifiers.

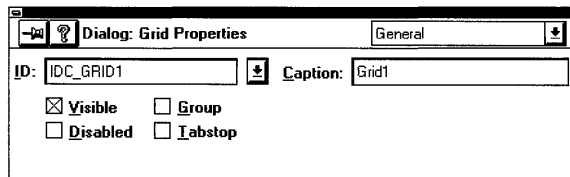
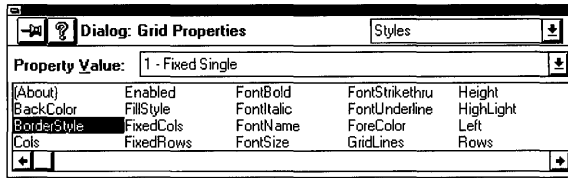


Figure 3.13 General Property Page for a VBX Control

The Styles property page (Figure 3.14) gives you access to all the VBX control's properties that can be edited as you build the dialog box.



**Figure 3.14** Styles Property Page for a VBX Control

► **To edit VBX control properties:**

1. Select the VBX control you want to edit and move to the General or Styles property page.
2. On the General property page, you can modify the properties shown in Figure 3.13.
3. On the Styles property page, select a property and use the edit box to change the property's value. Depending on the type of property, you may also be able to:
  - Choose from a list of property values.
  - Or–
  - Choose the Open Dialog button to the right of the edit box to bring up a dialog box for changing the property's value.

Moving or resizing a VBX control in the dialog editor changes its Top, Left, Right, or Bottom properties as appropriate.

## Working with User-Defined Controls



User-defined controls are custom controls that use a programming interface other than the VBX format. (The App Studio control palette button for user-defined controls is shown at left.) App Studio user-defined controls let you use existing custom controls regardless of their format.

With user-defined controls, App Studio allows you to:

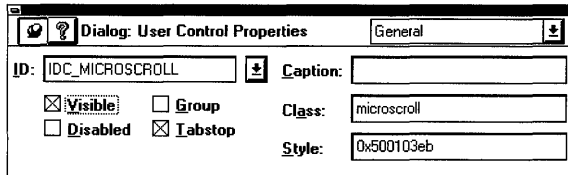
- Set the location in the dialog box.
- Enter a caption.
- Identify the name of the control's Windows class (your application code must register the control by this name).
- Type in a 32-bit hexadecimal value that sets the control's style.

The differences between App Studio support for VBX controls and user-defined controls are shown in Table 3.1.

**Table 3.1 Comparison of App Studio Support for VBX Controls and User-Defined Controls**

VBX Controls	User-Defined Controls
During the design phase, view the control as it will appear at run time.	During the design phase, view the control as a gray square.
Edit style properties by name.	Edit style properties by entering a hexadecimal number.
During test mode, test the control's appearance and behavior.	During test mode, view the control as a gray square (run-time behavior not simulated).

Figure 3.15 shows the General property page for the MicroScroll custom-control example in the Windows SDK documentation.



**Figure 3.15 General Property Page for a User-Defined Control**

- ▶ **To edit user-defined control properties:**
  1. Select the control and move to the General property page.
  2. Enter or modify the information as appropriate.

For more information on user-defined control properties, see the “Property Page Reference,” Chapter 11, or choose the Property page Help button.

## Connecting to Program Code



App Studio makes it easy to create or update the connections between your user interface and the underlying program code. By using the Resource menu's ClassWizard command (or clicking the ClassWizard toolbar button), you can:

- Create a dialog-box class derived from the Microsoft Foundation **CDialog** class.
- Define dialog box and control-message handlers (the dialog-box class's "message map").
- Define class member functions that gather and validate the data entered in the dialog box.

For detailed information on using ClassWizard from within App Studio, see Chapter 9.

## Creating a Form View

You can use App Studio to create a template that is used as a "form view," a **CView**-compatible window that contains dialog-box controls. An application that might need a form view is one in which the primary program function is data entry. In this case, the program's main view contains nothing but dialog-box controls for entering data.

To construct a form view, you create a dialog box as you normally would but set several style properties differently (see the following procedure). You then incorporate the form view into your program using the Microsoft Foundation Class Library **CFormView** class. You can use the same procedure to create a template for use with the **CDialogBar** class. For more information, see the *Class Library Reference*.

- ▶ **To create a dialog-box template for use with the CFormView or CDialogBar class:**
  1. Use the dialog editor in the usual way to create a dialog-box template with the controls arranged as you want them to appear in the form view.
  2. Move to the Styles property page and set the following properties:
    - In the Style box, select Child from the list of choices.
    - In the Border box, select None from the list of choices.
    - Clear the Visible property.
  3. On the General property page, clear the dialog-box template's caption.
  4. Incorporate the template into your program using the **CFormView** class.

## Testing a Dialog Box

You can simulate the run-time behavior of a dialog box from within App Studio without compiling your program. This gives you immediate feedback on how the layout of controls appears and performs and thus speeds up the user-interface design process.



When you are in test mode you can:

- Type in text, select from combo-box lists, turn options on and off, and choose commands.
- Test the tab order.
- Test the grouping of controls such as radio buttons or check boxes.
- Test the dialog box's keyboard interface (controls that have mnemonic keys defined for them).

---

**Note** Connections to dialog-box code made using ClassWizard are not simulated during dialog-box test mode.

---

When you test a dialog box, it is usually displayed at a location relative to the main App Studio program window. If the dialog box's Absolute Align property is selected, the dialog box is displayed at a position relative to the upper-left corner of the screen.

► **To test a dialog-box:**



1. On the dialog-editor toolbar, click the Test button, or from the Resource menu choose Test (CTRL+T).
2. To end the test session, take one of the following actions:
  - Press ESC.
  - Close the dialog box using its control-menu box (ALT+F4).
  - Choose a pushbutton with a symbol name of IDOK or IDCANCEL.

## CHAPTER 4

# Using the Menu Editor

You can create and edit menus in App Studio by working directly with a menu bar that closely resembles the one in your finished application.

In addition, you can use the Visual C++ ClassWizard to hook menu items up to code. For more information on ClassWizard, see Chapter 9.

Figure 4.1 identifies the terms used to describe menu building in the procedures that follow.

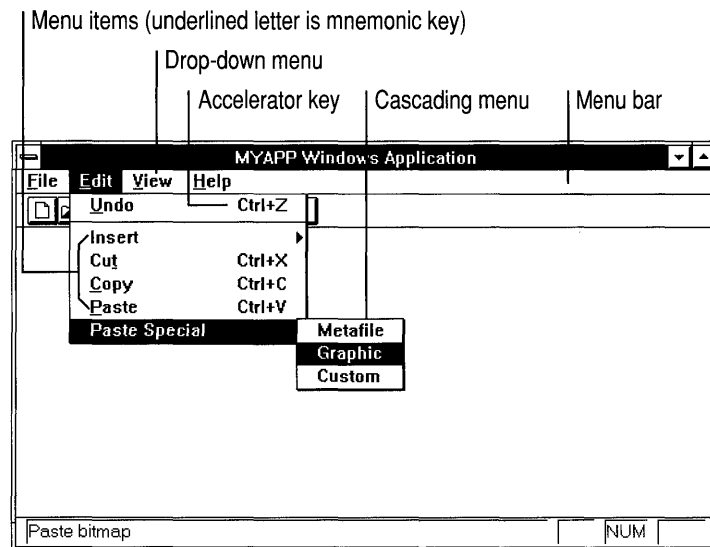


Figure 4.1 Menu Terminology

## Working with Menus and Menu Items

This section contains procedures for the following menu editing tasks:

- Creating a new menu resource or editing an existing one.
- Creating menus and menu items.
- Selecting menus and menu items.
- Moving or copying menus and menu items.
- Viewing the menu resource as a pop-up menu.

## Opening New or Existing Menu Resources

- ▶ **To create a new menu resource:**



- On the App Studio toolbar, click the New Menu button.  
–Or–
- In the resource browser window, click New, or from the Resource menu, choose New (CTRL+R). Then select Menu from the list of choices and click OK or press ENTER.

The menu editor window appears.

- ▶ **To edit an existing menu resource:**

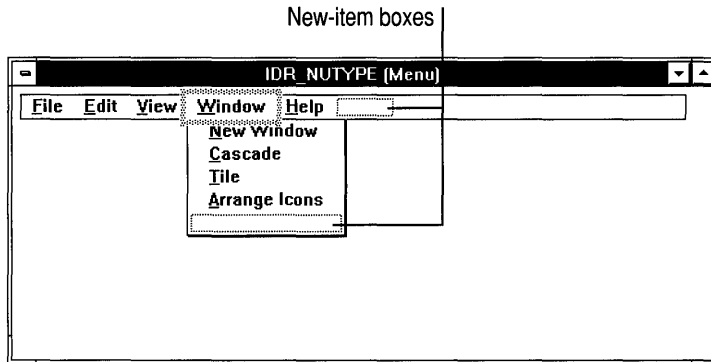
1. Move to the resource browser window.
2. In the Type box, select Menu.  
A list of the menu resources in the current file appears in the Resources box.
3. In the Resources box, select the menu resource you want to edit.
4. Choose Open.  
Or select the resource and press ENTER.  
Or double-click the resource.  
Or from the Resource menu, choose Open.

The menu editor window appears.

## Creating Menus or Menu Items

### ► To create a drop-down menu on the menu bar:

1. Click the new-item box (an empty rectangle) on the menu bar to select it (see Figure 4.2). You can also move to the new-item box with the TAB (move right) and SHIFT+TAB (move left) keys or the right and left arrow keys.



**Figure 4.2** Menu Editor New-Item Boxes

2. Type the name of the menu. When you start typing, focus automatically shifts to the General property page, and the text you type appears both in the Caption box and in the menu editor window.

You can define a mnemonic key that allows the user to select the drop-down menu by typing an ALT plus a single letter. Use an ampersand (&) in front of the letter that represents the mnemonic character you wish to specify. Make sure all the mnemonics on a menu are unique.

Once you have given the menu a name on the menu bar, the new-item box shifts to the right, and another new-item box opens below for adding menu items.

---

**Note** To create a single-item menu on the menu bar, clear the Popup option on the General property page.

---

### ► To create a menu item:

1. First create a drop-down menu according to the steps outlined in the previous procedure.
2. Select the menu's new-item box.

–Or–

Select an existing menu item and press INS. The new-item box is inserted before the selected item.

3. Type the name of the menu item. When you start typing, focus automatically shifts to the General property page, and the text you type appears in the Caption box.

You can define a mnemonic key that allows the user to select the menu command by typing a single letter. Use an ampersand in front of the letter that represents the mnemonic character you wish to specify. The mnemonic allows the user to select the menu command by typing that letter.

4. In the ID box, type in the menu item ID, or select an existing command identifier.
5. On the General property page, select the menu item styles that apply.

For information on what each menu item style means, see Chapter 11, “Property Page Reference,” or choose the Property page Help button.

6. In the General property page’s Prompt box, type the prompt string you want to appear in your application’s status bar. This feature is only available with Microsoft Foundation Class Library resource script (.RC) files.

This creates an entry in the string table with the same resource identifier as the menu item you created.

7. Press ENTER to complete the menu item. The new-item box is selected so you can create additional menu items.

► **To create a cascading (hierarchical) menu:**

1. Select the new-item box on the drop-down menu where you want the cascading menu to appear. Then type the name of the menu item that, when selected, causes the cascading menu to appear.

–Or–

Select an existing menu item that you want to be the parent item of the cascading menu.

2. On the General property page, select the Popup check box. The menu item is marked with the cascading menu symbol (►), and a new-item box appears to the right.
3. Add additional menu items to the cascading menu according to the instructions in the previous procedure.

## Selecting Menus or Menu Items

► **To select a drop-down or cascading menu and display its menu items:**

- Click the menu caption on the menu bar or the parent item of the cascading menu. Then click the menu item you want.

–Or–

- Move to the menu caption with the TAB (move right) and SHIFT+TAB (move left) keys or the right and left arrow keys.
- **To select one or more menu items:**
1. Click the drop-down or cascading menu you want.  
Its menu items are displayed.
  2. Click to select a menu item, or hold down the SHIFT key to select multiple menu items. Holding down the SHIFT key and clicking an already-selected menu item deselects it.
- Or–
- With the mouse pointer outside the menu, hold down the left mouse button and draw a selection box around the menu items you want to select.

## Moving and Copying Menus or Menu Items

Using the App Studio menu editor you can:

- Move or copy menus or menu items.
- Move cascading menus to the menu bar.
- Make a drop-down menu cascading.

The simplest way to edit a menu resource is to use the drag and drop method described in Chapter 1, page 8. You can also use the Cut, Copy, and Paste commands on the App Studio Edit menu.

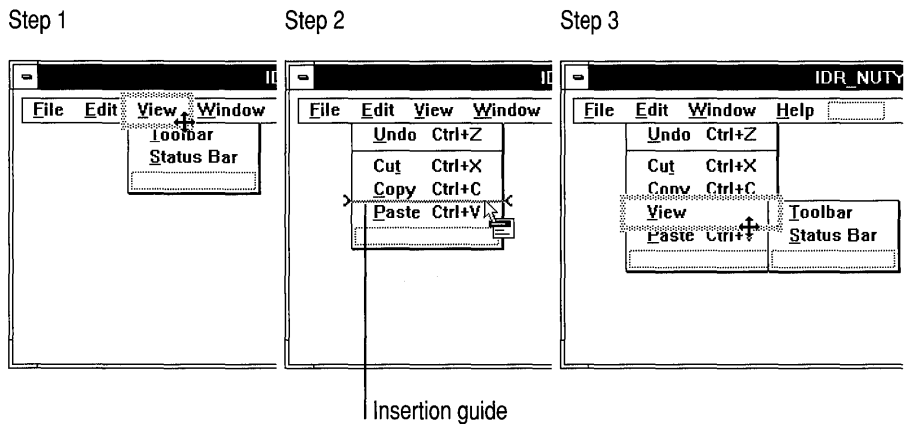


Figure 4.3 Example: Moving a Cascading Menu to the Menu Bar

- ▶ **To move or copy drop-down menus or menu items using drag and drop:**
  1. Drag (or to copy, hold down the CTRL key and drag) the item you want to:
    - A new location in the current menu.
    - A different menu. (You can navigate into other menus by dragging the mouse cursor over them.)
  2. Drop the menu item when the insertion guide (see Figure 4.3) shows the position you want.
  
- ▶ **To move or copy menus or menu items using the App Studio Edit menu:**
  1. Select one or more menus or menu items.
  2. From the Edit menu, choose Cut (to move) or Copy (to copy) the objects. You can also use the keyboard shortcuts for Cut (CTRL+X or SHIFT+DEL) and Copy (CTRL+C or SHIFT+INS).
  3. If you are moving the items to another menu resource or resource script file, make that menu editor window active.
  4. Select the menu or menu item you want. The moved or copied item is placed before the item you select.
  5. From the Edit menu, choose Paste.

## Viewing the Menu Resource as a Drop-down Menu

Normally, when you are working in the menu editor, a menu resource is displayed as a menu bar with top-level menu items and drop-down menus. However, you may want to store a single menu as a separate menu resource so you can add it as a drop-down menu to your application's main menu bar while the program is running. To see what the separate menu resource looks like as a drop-down menu, use the menu editor's View as Popup command.

- ▶ **To view a menu resource as a drop-down menu:**
  - From the Resource menu, choose View as Popup (CTRL+U).  
To change back to the menu-bar view, choose View as Popup (CTRL+U) again.

## Associating a Menu Item with an Accelerator Key

Many times you want a menu item and a keyboard combination to issue the same program command. You do this by assigning the same resource identifier to the menu item and to an entry in your application's accelerator table. You then edit the menu item's caption to show the name of the accelerator key.

► **To associate a menu item with an accelerator key:**

1. In the menu editor, select the menu item you want and move to the General property page.
2. In the ID box, give the menu item a descriptive command identifier (for example, `ID_FILE_OPEN`).
3. In the Caption box, add the name of the accelerator key to the menu caption:
  - Following the menu caption, type the escape sequence for the TAB key (`\t`), so that all the menu's accelerator keys are left-aligned.
  - Type the name of the modifier key (`CTRL`, `ALT`, or `SHIFT`) followed by a plus sign and the name, letter, or symbol of the additional key.

For example, if you assign `CTRL+O` to the File menu's Open command, you modify the menu's caption so that it looks like this:

```
Open\tCtrl+O
```

The menu item in the menu editor is updated to reflect the new caption as you type it.

4. Create the accelerator-table entry and assign it the same identifier as the menu item. Use a key combination that you think will be easy to remember.

See Chapter 5 for more information on creating and naming accelerator resources.





# Using the Accelerator Table Editor

An accelerator table is a Windows resource that contains a list of accelerator keys (also known as shortcut keys) and the command identifiers that are associated with them. A program can have more than one accelerator table.

Normally accelerators are used as keyboard shortcuts for program commands that are also available on a menu or toolbar. However, you can use the accelerator table to define key combinations for commands that don't have a user-interface object associated with them.

You can use Visual C++ ClassWizard to hook accelerator key commands to code. For more information on ClassWizard, see Chapter 9.

In this chapter, you'll see how to use the accelerator table editor to:

- Add, delete, change, or browse the accelerator key assignments in your project.
- View and change the resource identifier associated with each entry in the accelerator table. The identifier is used to reference each accelerator table entry in program code.
- Associate an accelerator key with a menu item.

## Opening a New or Existing Accelerator Table

You can call up the accelerator table editor to edit an existing accelerator table or create a new one.

► **To create a new accelerator table:**



- On the App Studio toolbar, click the New Accelerator Table button.  
–Or–
- In the resource browser window, click New, or from the Resource menu, choose New (CTRL+R). Then select Accelerator from the list of choices and click OK or press ENTER.

The accelerator table editor window appears.

---

**Note** Windows does not allow the creation of empty accelerator tables. If you create an accelerator table with no entries, it is deleted automatically when you exit App Studio.

---

► **To edit an existing accelerator table:**

1. Move to the resource browser window.
2. In the Type box, select Accelerator.

A list of the accelerator tables in the current file appears in the Resources box.

3. In the Resources box, select the accelerator table you want to edit.
4. Choose Open.

Or select the resource and press ENTER.

Or double-click the resource.

Or from the Resource menu, choose Open.

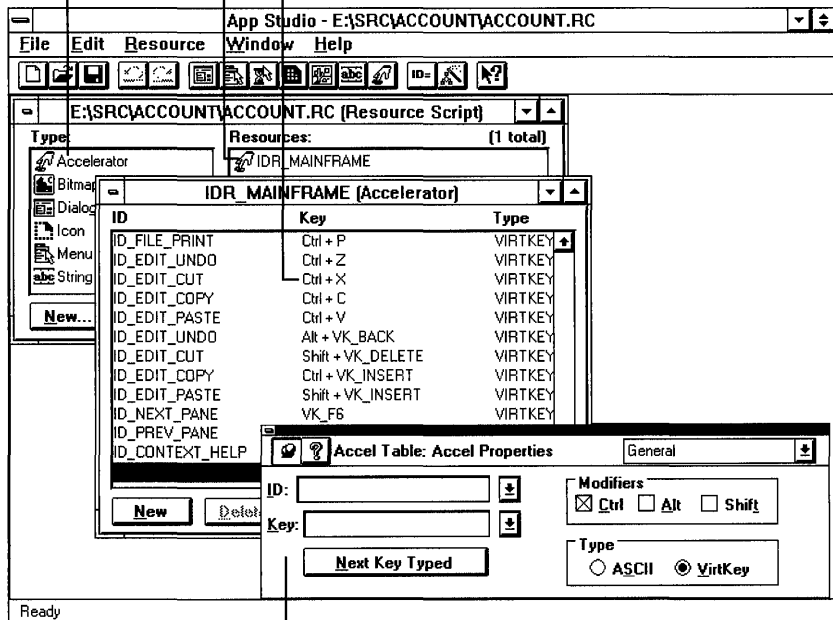
The accelerator table editor window appears.

## Editing the Accelerator Table

When you edit an existing accelerator table or create a new one, the accelerator table editor window appears. Use the accelerator-table window to browse existing accelerator keys, to delete keys, or to select the new-item box (an empty rectangle) at the end of the list. Move to the accelerator-table property page to edit or define values for each accelerator-table entry.

Figure 5.1 shows a portion of the App Studio screen in an arrangement suitable for working with individual accelerator-table entries.

1. In the Type box, select Accelerator.
2. In the Resources box, select an accelerator table to edit.
3. In the Accelerator window, select an entry to modify.



4. Use the General property page to add or change values.

**Figure 5.1** Using the Accelerator Table Editor and Properties Window

► **To add an entry to the accelerator table:**

1. Select the new-item box at the end of the list, or from the Resource menu, choose New Accelerator (INS).
2. Move to the General property page to define the key. (Pressing INS, or typing the key name with the new-item box selected, moves you to the Properties window automatically.)

**Note** Make sure all accelerators you define are unique. When duplicate accelerator keys are assigned, only the first one works correctly.

- ▶ **To delete an entry from the accelerator table:**
  1. Select the entry you want to delete. Use the CTRL or SHIFT keys to select multiple entries.
  2. Choose Delete at the bottom of the accelerator table editor window, or from the Edit menu, choose Delete (DEL).
- ▶ **To move or copy an accelerator-table entry from one resource script file to another:**
  1. Open the accelerator table editor windows in both resource script files.
  2. Use the drag and drop method described in Chapter 1, page 8.
    - Or–
    - Use the Edit menu's Copy (or Cut) and Paste commands.

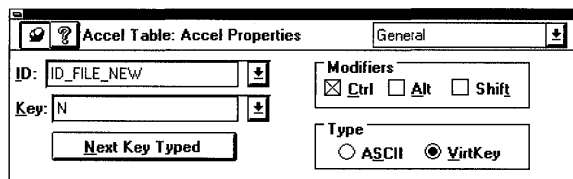
---

**Note** You can also use the CTRL key with drag and drop, or use the Edit menu, to copy keys within an accelerator table. However, exercise caution when doing so as this creates duplicate accelerator keys. App Studio does not prompt you to resolve accelerator key conflicts.

---

## Editing Accelerator Properties

Use the accelerator table's General property page (Figure 5.2) to edit the values for each accelerator.



**Figure 5.2** General Property Page for Accelerators

An accelerator can be either an ASCII value or virtual key. A virtual key is a device-independent identifier. The Key box on the General property page contains a list of standard virtual key identifiers.

- ▶ **To define or change the values for an accelerator-table entry:**
  1. Select the entry you want to define or change.
  2. Move to the General property page.

3. In the Key box, type the name of the key, or choose from the list of virtual key identifiers. Select additional options as appropriate under Modifiers (CTRL, ALT, or SHIFT) and Type (ASCII or VirtKey [virtual key]). When first created, a new key has the CTRL modifier selected.

–Or–

Choose Next Key Typed and then type the key or key combination you want to enter. The Key box and the options under Modifiers and Type are updated automatically.

---

**Note** Accelerator values entered using the Next Key Typed command are always entered as virtual keys, if possible.

---

## Typing in Accelerator Values

The following are legal entries in the Key box of an accelerator table's General property page:

- An integer between 0 and 255 in decimal, hexadecimal, or octal format. The setting of the Type property determines if the number is an ASCII or virtual key value.

Single-digit numbers are always interpreted as the corresponding key, rather than as ASCII values. To enter an ASCII value from 0 to 9, precede it with two zeros (for example, 006).

- A single keyboard character. Uppercase A-Z or the numbers 0 through 9 can be either ASCII or virtual key values; any other character is ASCII only.
- A single keyboard character in the range A-Z (uppercase only), preceded by a caret (^)—for example, ^C. This enters the ASCII value of the key when it is pressed with the CTRL key held down.

---

**Note** When entering an ASCII value, the CTRL and SHIFT modifiers on the General property page are not available. You cannot use a control-key combination entered with a caret to create a virtual accelerator key.

---

- Any valid virtual key identifier. The Key box on the General property page contains a list of standard virtual key identifiers.

## Associating a Menu Item with an Accelerator Key

Many times you want a menu item and a keyboard combination to issue the same program command. You do this by assigning the same resource identifier to the menu item and to an entry in your application's accelerator table. You then edit the menu item's caption to show the name of the accelerator.

► **To associate a menu item with an accelerator key:**

1. In the menu editor (see Chapter 4), select the menu item you want and move to the General property page.
2. In the ID box, give the menu item a descriptive command identifier (for example, `ID_FILE_OPEN`).
3. In the Caption box, add the name of the accelerator key to the menu caption:
  - Following the menu caption, type the escape sequence for the TAB key (`\t`), so that all the menu's accelerator keys are left-aligned.
  - Type the name of the modifier key (`CTRL`, `ALT`, or `SHIFT`) followed by a plus sign and the name, letter, or symbol of the additional key.

For example, if you assign `CTRL+O` to the File menu's Open command, you modify the menu's caption so that it looks like this:

```
Open\tCtrl+O
```

The menu item in the menu editor is updated to reflect the new caption as you type it.

4. Create the accelerator-table entry and assign it the same identifier as the menu item. Use a key combination that you think will be easy to remember.

# Using the String Editor

The App Studio string editor is a window and its associated property page that allow you to edit or add to a program's standard Windows string table resource.

In this chapter, you'll see how to use the string editor to:

- Create a new string table.
- View the string segments in a program.
- Add or delete individual strings, or move them from one segment to another.
- Change a string, or its name or value.

## String Tables

In programming for Windows, strings can be stored in a resource type called a string table. An application can have only one string table.

In a string table, strings are grouped into segments, or blocks, of 16 strings each. The segment a string belongs to is determined by the value of its identifier; for example, strings with identifiers of 0 to 15 are in one segment, strings with values of 16 to 31 are in a second segment, and so on. Thus, to move a string from one segment to another you need to change its identifier.

Individual string segments are loaded on demand in order to conserve memory. For this reason, programmers usually try to group strings into logical groupings of sixteen or less and then to use each group or segment only when it's needed.

## Opening a New or Existing String Table

You use App Studio to create your application's string table or use the string editor to edit an existing one.



► **To create a new string table:**



- On the App Studio toolbar, click the String Table button.
- Or–
- In the resource browser window, click New, or from the Resource menu, choose New (CTRL+R). Then select String from the list of choices and click OK or press ENTER.

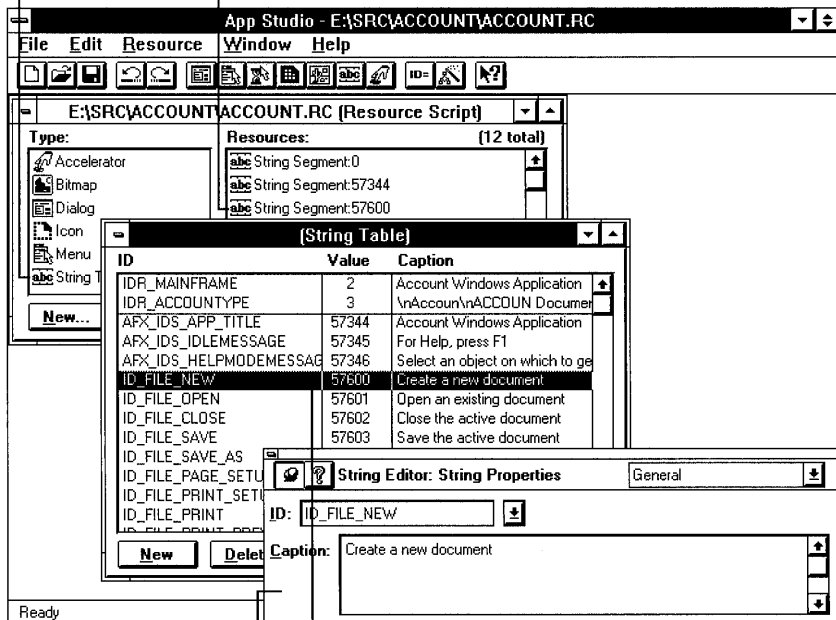
The string editor window appears.

**Note** Windows does not allow the creation of empty string tables. If you create a string table with no entries, it is deleted automatically when you exit App Studio.

Once you create your application's string table, you can use the App Studio resource browser window to view a list of string segments in the string table and to jump to the beginning of a selected segment. Figure 6.1 shows this process.

1. In the Type box, select String Table.

2. In the Resources box, select a string segment to edit.



3. In the String Editor window, select the string to modify.

4. Use the General property page to edit the string or its ID.

**Figure 6.1** Using the String Editor

► **To open an existing string table for editing:**

1. Move to the resource browser window.
2. In the Type box, select String Table.  
A list of the string segments in your application's string table appears.
3. In the Resources box, select the string segment you want to edit.  
Selecting a specific segment of the string table allows you to easily jump to a specific group of strings. However, opening the string editor gives you access to the entire string table.
4. Choose Open.  
Or select the resource and press ENTER.  
Or double-click the resource.  
Or from the Resource menu, choose Open.

The string editor window appears.

## Editing the String Table

The string editor includes a Find command to let you quickly locate strings in the string table by either their caption or resource identifier.

► **To find a string in the string table:**

1. Open the string table by selecting it in the Type box of the resource browser window and then clicking any of the segments listed in the Resources box.
2. At the bottom of the string editor window, choose Find.  
The Find String dialog appears.
3. In the Find What box, type the caption text or resource identifier of the string you want to find. Set or clear the Match Case option as appropriate.
4. Press ENTER.  
If a string or its identifier in the string table matches what you typed, it is selected.

Once the string editor window is displayed, you can add or delete entries in the string table. String table segments are separated by a horizontal line in the string editor window.

► **To add a string-table entry:**

1. Select the new-item box (an empty rectangle) at the end of a string segment.
2. Type the new string.  
Focus shifts to the Properties window as you start typing, and the string is given the next identifier in sequence.  
–Or–
  1. Select an existing entry in the string table.
  2. Choose the New command at the bottom of the string editor window, or from the Resource menu, choose New String (INS).  
A new string is created and given the next available identifier after the currently selected string.
3. Type the new string.

---

**Note** Null strings are not allowed in Windows string tables. If you create an entry in the string table that is a null string, the entry is deleted when you close the string editor.

---

► **To delete a string in the string editor:**

1. Select the string you want to delete.
2. Choose Delete at the bottom of the string editor window, or from the Edit menu, choose Delete (DEL).

► **To move a string from one segment to another:**

1. Select the string you want to move.
2. Move to the General property page. In the ID box, change the string's value so that it falls in the range you want.

For example, to move a string with a name of IDS\_MYSTRING and a value of 100 to a segment in the 200 range, type the following in the ID box:

```
IDS_MYSTRING=201
```

3. Press ENTER to record the change.

► **To move a string from one resource script (.RC) file to another:**

1. Open the string editor windows in both resource script files.
2. Use the drag and drop method described in Chapter 1, page 8.

–Or–

Use the File menu's Cut and Paste commands.

---

**Note** If the moved string's symbol name or value conflicts with an existing identifier in the destination file, the symbol name is changed (if a symbol by that name already exists) or the symbol value is changed (if a symbol with that value already exists).

---

## Editing a String's Properties

► **To change a string or its identifier:**

1. Select the string you want to edit.
2. Move to the string's General property page.
3. In the Caption box, modify the string itself.

In the ID box, modify the string's identifier:

- Type in a new symbol name or choose one from the list.
- Change a string's value by typing the symbol name followed by an equal sign and the new value; for example

```
IDS_ERROR_MSG=2350
```

For more information on editing symbols, see Chapter 2.

► **To add formatting or special characters to a string:**

- Use the standard escape sequences shown in Table 6.1.

**Table 6.1 Formatting and Special Characters in Strings**

To Get This:	Type This:
New line	\n (or press CTRL+ENTER when typing in the string)
Carriage return	\r
Tab	\t (or press CTRL+TAB when typing the string)
Backslash (\)	\\
ASCII character	\ddd (octal notation)
Alert (bell)	\a

---



# Using the Graphics Editor

The App Studio graphics editor has a rich set of tools for drawing bitmaps, icons and cursors, as well as features to support the creation of toolbar bitmaps and the management of icon and cursor images.

With the graphics editor, you can:

- Create and edit bitmaps in 2 or 16 colors, in sizes up to 999 x 999 pixels.
- Use a full complement of drawing tools.
- Easily copy images from one bitmap to another.
- Use custom brushes for special effects.
- Draw cursors and icons with transparent and inverse-color regions.
- Edit bitmaps for toolbar buttons.

Most editing procedures are the same for bitmaps, icons, and cursors. This chapter first shows the procedures common to all graphical resources. Later sections detail procedures and graphics-editor capabilities specific to icons, cursors, and toolbar-button images.

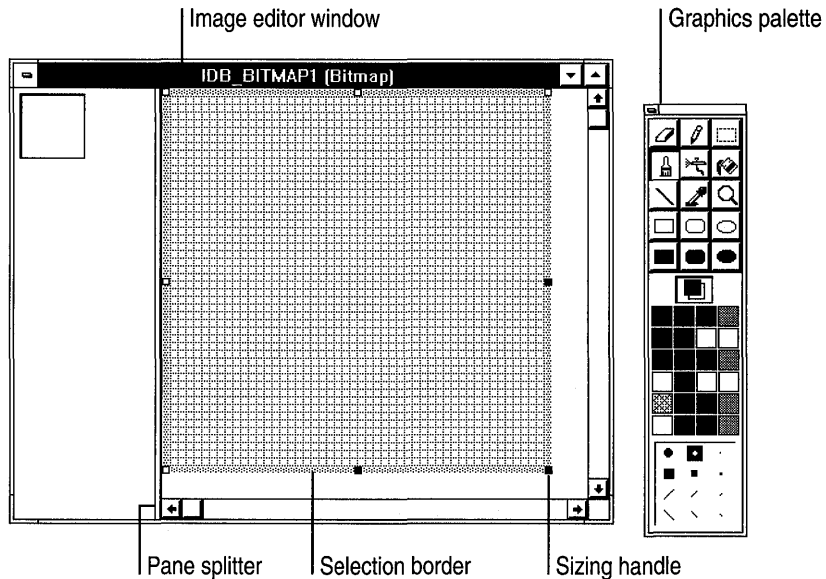
---

**Note** Many of the graphics editor's functions require a mouse or other pointing device.

---

## Windows and Tools for Editing Graphics

You edit bitmaps, icons, and cursors in App Studio's image editor window, using the tools on the graphics palette (Figure 7.1).



**Figure 7.1 Image Editor Window and Graphics Palette**

## The Image Editor Window

The image editor window shows two views of an image. A splitter separates the two panes. You can drag the splitter from side to side to change the relative sizes of the panes. The active pane displays a selection border, as shown in Figure 7.1.

## The Graphics Palette

The graphics palette has four parts, which are shown in Figure 7.1:

- The toolbox, which contains 15 tools for drawing, painting, erasing, and manipulating views
- The color indicator, which shows the foreground and background colors and (for icons and cursors) selectors for “screen” and “inverse” color
- The color palette, which you click to select the foreground and background colors
- The option selector, which you click to select brush widths and other drawing options

To use the toolbox, color palette, and option selector, you click the desired tool, color, or option.

## The Status Bar

The Status bar, at the bottom of the App Studio frame window, displays two panes when an image editor window is open. When the mouse cursor is over an image, the left pane shows the cursor's current position, in pixels, relative to the upper-left corner of the image. During a dragging operation such as selecting, moving, or drawing a rectangle, the right pane shows the size, in pixels, of the affected area.

## The Image Menu

The Image menu, which appears only when the graphics editor is active, has commands for editing images, managing color palettes, and setting image editor window options.

## Editing Graphical Resources

You need to know just a few fundamental editing operations—creating bitmaps, drawing, copying, and so on—to use the graphics editor. This section describes these graphics-editing tasks:

- Creating a new bitmap, icon, or cursor, or editing an existing one
- Setting bitmap properties
- Drawing and painting
- Cutting, copying, clearing, and moving selected parts of a bitmap

You can also import existing bitmaps, icons, and cursors and add them to your project, and you can open files that are not part of a project for “stand-alone” editing. See Chapter 2, “Working with Files and Symbols,” for more information on importing resources.

---

**Note** Most graphics-editor operations are the same for all kinds of graphical resources. Unless the text states otherwise, the procedures described in this section can be performed on bitmaps, cursors, or icons.

---

## Opening New or Existing Graphical Resources

New graphical resources are always added to the current resource script, executable, dynamic-link library, or other file. If more than one file is open, before you create the bitmap, click the resource browser window of the file to which you want to add the new bitmap.



► **To create a new graphical resource:**

- On the App Studio toolbar, click the button for the resource you want to create:



- The bitmap button



- The cursor button



- The icon button

–Or–

- In resource browser window, click New, or from the Resource menu, choose New (CTRL+R). Then select Bitmap, Icon, or Cursor from the list of choices and click OK or press ENTER.

The image editor window appears.

► **To create a new graphical resource for stand-alone editing:**

1. From the File menu, choose New (CTRL+N).
2. From the New dialog box, choose Bitmap, Icon, or Cursor.
3. Choose OK.

The image editor window appears.

► **To open an existing graphical resource for stand-alone editing:**

1. From the File menu, choose Open (CTRL+O). The Open dialog box appears.
2. Use the Open dialog box to navigate directories and choose a file name.

► **To edit an existing graphical resource:**

1. Move to the resource browser window.
2. In the Type box, select Bitmap, Icon, or Cursor.  
A list of the resources in the current file appears in the Resources box.
3. In the Resources box, select the bitmap resource you want to edit.
4. Choose Open.

The image editor window appears.

---

**Tip** The Properties window displays a thumbnail view of each graphical resource as you select its entry in the Resources box. To browse through the bitmaps in the file you are working on, select the first bitmap and then press the DOWN arrow key until the desired bitmap appears.

---

## Setting Properties

You use the Properties window to change most resource properties. Exceptions are the Width, Height, and Colors properties of icons and cursors, which you change by editing the App Studio initialization file, APSTUDIO.INI (see “Editing Device Descriptions,” page 102), and the cursor’s hotspot (see “Setting a Cursor’s Hotspot,” page 104).

► **To change a bitmap’s properties:**

1. Select the bitmap whose properties you want to change.
2. Activate the General property page by pressing ENTER or clicking the Properties window.
3. Change any or all of these properties:
  - In the ID box, modify the resources’s identifier. For a bitmap, App Studio by default assigns the next available identifier in a series: IDB\_BITMAP1, IDB\_BITMAP2, and so forth. Similar names are used for icons and cursors.
  - In the Width and Height boxes, modify the bitmap’s width and height (in pixels). The default value for each is 48.

If you change the dimensions of a bitmap using the General property page, the image is cropped or “blank” space is added to the right of or below the existing image.
  - In the Colors list box, select Monochrome or 16. If you have already drawn the bitmap with a 16-color palette, selecting Monochrome causes App Studio to substitute black and white for the colors in the bitmap. Contrast is not always maintained: for example, adjacent areas of red and green are both converted to black.
  - In the Filename box, modify the name of the file in which the bitmap is to be stored. By default, App Studio assigns a base file name created by removing the first four characters (“IDB\_”) from the default identifier and adding the extension .BMP.
  - Check the Save Compressed check box to cause App Studio to save the bitmap in a compressed format.

## Drawing and Painting

You access most of the graphics editor’s drawing tools either by clicking icons on the graphics palette or by entering keyboard commands. More advanced drawing operations, such as creating custom brushes, can be performed only from the keyboard. App Studio’s keyboard commands are listed in Chapter 10.

## Showing and Hiding the Graphics Palette

Since many of the drawing tools are available from the keyboard, sometimes it is useful to hide the graphics palette.

► **To show or hide the graphics palette:**

- Press F2.

## Selecting Foreground and Background Colors

Except for the eraser, these tools draw with the current foreground or background color when you press the left or right mouse button, respectively.

► **To select a foreground color:**

- With the left mouse button, click the desired color on the color palette.

► **To select a background color:**

- With the right mouse button, click the desired color on the color palette.

## Freehand Drawing and Erasing

The graphics editor's freehand drawing and erasing tools all work in the same way: you select the tool and, if necessary, select foreground and background colors and size and shape options. You then move the cursor to the bitmap and click or drag to draw and erase.

When you have selected the eraser tool, brush tool, or airbrush tool, the option selector displays that tool's options.

---

**Tip** Instead of using the eraser tool, you may find it more convenient to draw in the background color with one of the drawing tools.

---

► **To select and use a drawing tool:**

1. Click a button on the toolbox:



- The eraser tool (SHIFT+P), which “paints over” the image with the current background color when you press the left mouse button and replaces the current foreground color with the current background color when you press the right mouse button.



- The pencil tool (P), which draws freehand in a constant width of one pixel.



- The brush tool (D), whose shape and size are determined by the option selector.



- The airbrush tool (A), which randomly distributes color pixels around the center of the brush.

2. If necessary, select colors and a brush:
    - In the color palette, click with the left button to select a foreground color or with the right button to select a background color.
    - In the option selector, click a shape representing the brush you want to use. Your selection is highlighted.
  3. Move the cursor to the place on the bitmap where you want to start drawing or painting. The brush or cursor appears on the bitmap.
  4. Press the left mouse button (for the foreground color) or the right mouse button (for the background color), and hold it down as you draw.
  5. Release the mouse button.
- **To resize the brush tool or eraser tool to a single pixel:**
- Press the PERIOD key.
- **To change the size of the brush, airbrush, or eraser:**
- Press the PLUS SIGN key to increase the size or the MINUS SIGN key to decrease it.  
–Or–
  - Press PERIOD to choose the smallest size.  
–Or–
  - Choose a brush in the option selector.

## Drawing Lines and Closed Figures

App Studio's tools for drawing lines and closed figures all work in the same way: you put the cursor at one point and drag to another. For lines, these points are the endpoints. For closed figures, these points are opposite corners of a rectangle bounding the figure.

Lines are drawn in a width determined by the current brush selection, and framed figures are drawn in a width determined by the current width selection. Lines and all figures, both framed and filled, are drawn in the current foreground color if you press the left mouse button or in the current background color if you press the right mouse button.

► **To draw a line:**



1. From the toolbox, select the line tool (L).
2. If necessary, select colors: in the color palette, click with the left button to select a foreground color or with the right button to select a background color.
3. If necessary, select a brush: in the option selector, click a shape representing the brush you want to use. Your selection is highlighted.

4. Place the cursor at the line's starting point.
5. Drag the cursor to the line's endpoint.
6. Release the mouse button.

► **To draw a closed figure:**

1. From the toolbox, select a closed-figure drawing tool:



- The framed-rectangle tool (R), which draws a rectangle framed with the foreground or background color.



- The filled-rectangle tool (SHIFT+R), which draws a rectangle filled with the foreground or background color.



- The framed-round-rectangle tool (N), which draws a rectangle with rounded corners framed with the foreground or background color.



- The filled-round-rectangle tool (SHIFT+N), which draws a rectangle with rounded corners filled with the foreground or background color.



- The framed-ellipse tool (E), which draws an ellipse framed with the foreground or background color.



- The filled-ellipse tool (SHIFT+E), which draws an ellipse filled with the foreground or background color.

2. If necessary, select colors: in the color palette, click with the left button to select a foreground color or with the right button to select a background color.
3. If necessary, select a line width: in the option selector, click a shape representing the brush you want to use. A rectangle appears around the option you have selected.
4. Move the cursor to one corner of the rectangular area in which you want to draw the figure.
5. Drag to the diagonally opposite corner.
6. Release the mouse button.

## Filling Bounded Areas

The graphics editor provides the fill (or “paint-bucket”) tool for filling any enclosed bitmap area with the current drawing color or the current background color.

► **To use the fill tool:**



1. From the toolbox, choose the fill tool (F).
2. If necessary, choose drawing colors: in the color palette, click with the left button to select a foreground color or with the right button to select a background color.

3. Move the fill tool to the area you want to fill.
4. Click the left or right mouse button to fill with the foreground color or the background color, respectively.

## Picking Up Colors

The color-pickup tool makes any color on the bitmap the current foreground color or background color, depending on whether you press the left or the right mouse button.

► **To pick up a color:**



1. From the toolbox, select the color-pickup tool (COMMA).  
The cursor changes to the “eyedropper.”
2. Click the bitmap area containing the color you want to pick up:
  - Left mouse button for the foreground color
  - Right mouse button for the background color

After you pick up a color, the graphics editor reactivates the most recent tool you used before the color-pickup tool.

► **To cancel the color-pickup tool:**

- Choose another tool or press ESC.

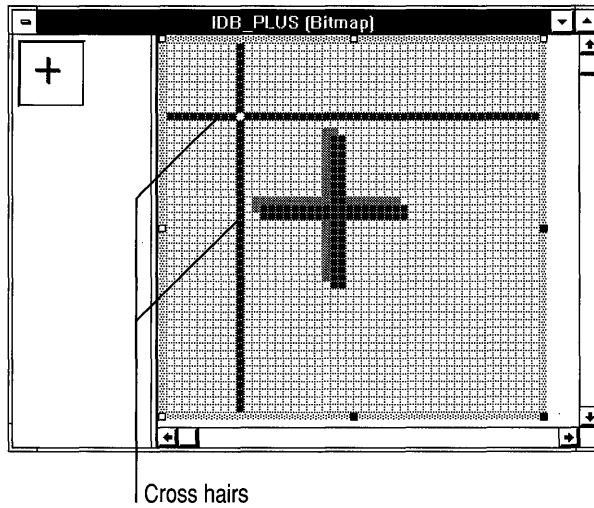
## Using the Selection Tool

The selection tool defines an area of the bitmap that you can cut, copy, clear, resize, invert, or move. You can also create a custom brush from the selection (see “Creating a Custom Brush”).

► **To select an area of the bitmap:**

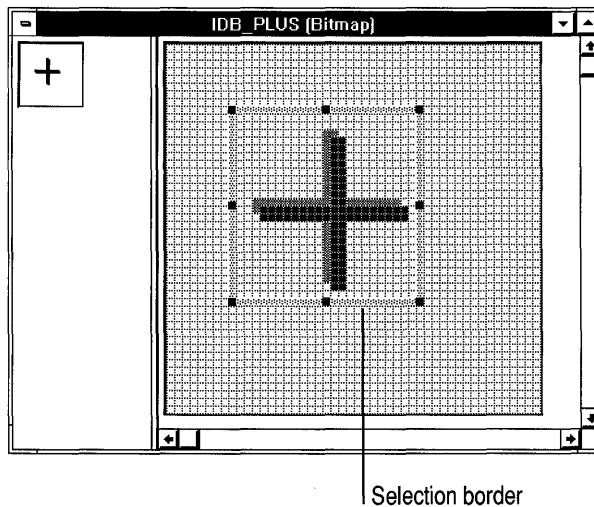


1. In the toolbox, click the selection tool (S).
2. Move the cursor to one corner of the bitmap area that you want to select.  
Cross hairs appear when the cursor is over the bitmap, as shown in Figure 7.2.



**Figure 7.2** Cross hairs for Defining the Selection

3. Drag the cursor to the opposite corner of the area you want to select. A rectangle shows which pixels will be selected. All pixels within the rectangle, including those “under” the rectangle, are included in the selection.
4. Release the button. The “selection border”—a rectangular frame—encloses the selected area, as in Figure 7.3. Now any operation you perform will affect only the pixels within the rectangle.



**Figure 7.3** Border Enclosing the Selected Area

- ▶ **To select the entire bitmap:**
  - Click the bitmap outside the current selection.
    - Or–
  - Press ESC.
    - Or–
  - Choose another tool in the toolbox.

## Cutting, Copying, Clearing, and Moving

You can perform standard editing operations—cutting, copying, clearing, and moving—with the selection, whether the selection is the entire bitmap or just a part of it. Because the graphics editor uses the Windows Clipboard, you can transfer images between App Studio and other applications for Windows, such as Microsoft Paintbrush™ and Microsoft Word for Windows.

In addition, you can resize the selection, whether it includes the entire bitmap or just a part (see “Resizing a Bitmap,” page 94).

- ▶ **To cut the current selection and copy it to the Clipboard:**
  - From the Edit menu, choose Cut (CTRL+X).

The original area of the selection is filled with the current background color, and the selection is now in the Clipboard.
- ▶ **To clear the current selection without copying it to the Clipboard:**
  - From the Edit menu, choose Clear (DEL).

The original area of the selection is filled with the current background color.
- ▶ **To paste the Clipboard contents into the bitmap:**
  1. From the Edit menu, choose Paste (CTRL+V).

The Clipboard contents, surrounded by the selection border, appear in the upper-left corner of the pane.
  2. Position the cursor within the selection border and drag the Clipboard contents to the desired location on the bitmap.
  3. To copy the Clipboard contents into the bitmap at the desired location, click outside of the selection border or choose a new tool.



► **To move the selection:**

1. Position the cursor inside the selection border or anywhere on it except the sizing handles.
2. Drag the selection to its new location.  
The original area of the selection is filled with the current background color.
3. To anchor the selection in the bitmap at its new location, click outside the selection border or choose a new tool.

► **To copy the selection:**

1. Position the cursor inside the selection border or anywhere on it except the sizing handles.
2. Hold down the CTRL key as you drag the selection to a new location.  
The area of the original selection is unchanged.
3. To copy the selection into the bitmap at its current location, click outside the selection cursor or choose a new tool.

► **To draw with the selection:**

1. Position the cursor inside the selection border or anywhere on it except the sizing handles.
2. Hold down the SHIFT key as you drag the selection.  
Copies of the selection are left along the dragging path. The more slowly you drag, the more copies are made.

## Choosing Opaque and Transparent Backgrounds

When you move or copy a selection, any pixels in the selection that match the current background color are by default “transparent”: they do not obscure pixels in the target location. (A custom brush behaves the same way. See “Creating a Custom Brush.”)

► **To toggle the background-color transparency:**

- Press O.

–Or–



- In the option selector, click the appropriate button:
  - Opaque background: existing image is obscured by all parts of the selection.
  - Transparent background: existing image shows through parts of the selection that match the current background color.

You can change the background color while a selection is already in effect to change which parts of the image are transparent.

## Flipping the Selection

- ▶ **To flip the selection along the horizontal axis:**
  - From the Image menu, choose Flip Horizontal (X).
- ▶ **To flip the selection along the vertical axis:**
  - From the Image menu, choose Flip Vertical (Y).

## Inverting Colors

So that you can tell how a bitmap will appear with inverted colors, App Studio provides a convenient way to invert colors in the selected part of the bitmap.

- ▶ **To invert colors in the current selection:**
  - From the Image menu, choose Invert Colors.

## Creating a Custom Brush

A custom brush is a rectangular portion of a bitmap that you “pick up” and use like one of the graphics editor’s ready-made brushes. All operations you can perform on a selection, you can perform on a custom brush as well.

- ▶ **To create a custom brush:**
  1. Select the part of the bitmap that you want to use for a brush (see “Using the Selection Tool,” page 89).
  2. Press CTRL+B.

Pixels in a custom brush that match the current background color are normally “transparent”: they do not paint over the existing image. You can change this behavior so that background-color pixels paint over the existing image.

You can use the custom brush like a “stamp” or a “stencil” to create a variety of special effects.

► **To draw custom-brush shapes in the background color:**

1. Select an opaque or transparent background (see “Choosing Opaque and Transparent Backgrounds,” page 92).
2. Set the background color to the color in which you want to draw.
3. Position the custom brush where you want to draw.
4. Press the right mouse button.

Any opaque regions of the custom brush are drawn in the background color.

► **To double or halve the custom-brush size:**

- Press PLUS to double the brush size, or MINUS to halve it.

► **To cancel the custom brush:**

- Press ESC or choose another drawing tool.

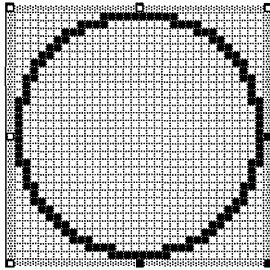
## Resizing a Bitmap

App Studio’s behavior in resizing a bitmap depends on whether the selection includes the entire bitmap or just part of it:

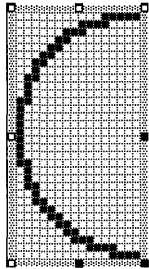
- When the selection includes only a part of the bitmap, App Studio shrinks the selection by deleting rows or columns of pixels and filling the vacated regions with the current background color, or it stretches the selection by duplicating rows or columns of pixels.
- When the selection includes the entire bitmap, App Studio either shrinks and stretches the bitmap, or crops and extends it.

See Figure 7.4 for illustrations of cropping, extending, shrinking, and stretching.

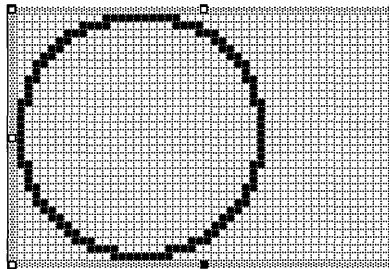
Original bitmap



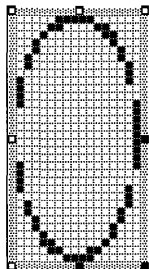
Cropped



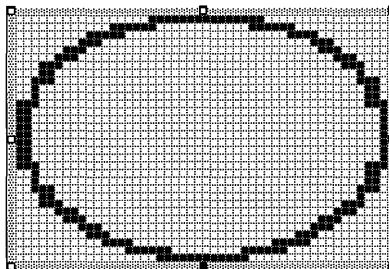
Extended



Shrunken



Stretched

**Figure 7.4** Cropping, Extending, Shrinking, and Stretching a Bitmap

There are two ways to resize the bitmap: the resizing handles and the Properties window. You drag the sizing handles to change the size of all or part of a bitmap. Sizing handles that you can drag are solid, like those on the lower right corner and the midpoints of the right and bottom sides of the bitmaps in Figure 7.4. You cannot drag handles that are hollow. You can use the Properties window to resize only the entire bitmap, not a selected part.

► **To crop or extend an entire bitmap:**

1. Select the entire bitmap.

If a part of the bitmap is currently selected and you want to select the entire bitmap, click anywhere on the bitmap outside the current selection border, press ESC, or choose another drawing tool.

2. Drag a sizing handle until the bitmap is the desired size.

Normally, App Studio crops or enlarges a bitmap when you resize it by moving a sizing handle. If you hold down the **SHIFT** key as you move a sizing handle, App Studio shrinks or stretches the bitmap.

► **To shrink or stretch an entire bitmap:**

1. Select the entire bitmap.

If a part of the bitmap is currently selected and you want to select the entire bitmap, click anywhere on the bitmap outside the current selection border, press **ESC**, or choose another drawing tool.

2. Hold down the **SHIFT** key and drag a sizing handle until the bitmap is the desired size.

► **To shrink or stretch part of a bitmap:**

1. Select the part of the bitmap you want to resize (see “Using the Selection Tool,” page 89).
2. Drag one of the sizing handles until the selection is the desired size.

► **To resize an entire bitmap using the Properties window:**

1. Click the Properties window to activate it.
2. Type the desired dimensions in the Width and Height boxes.

If you are increasing the size of the bitmap, App Studio extends the bitmap to the right or downward, or both, and fills the new region with the current background color. The image is not stretched.

If you are decreasing the size of the bitmap, App Studio crops the bitmap on the right or bottom edge, or both.

You can use the Width and Height properties only to resize the entire bitmap, not to resize a partial selection.

## Changing the Number of Colors in a Bitmap

When you increase the number of colors in a bitmap, the image does not change but you can then add more colors.

► **To change the number of colors in a bitmap:**

1. Click the Properties window to activate it.
2. Choose the appropriate setting in the Colors combo box.

# Managing the Graphics-Editor Workspace

By adjusting the graphics-editor workspace to fit your needs and preferences, you can work more effectively and comfortably. This section describes procedures for:

- Selecting and sizing image-editor panes
- Changing the magnification of image editor windows
- Displaying and hiding pixel grids

## Using Image-Editor Panes

Normally the image editor window displays a bitmap in two panes separated by a splitter. One view is actual size, and the other is enlarged (the default enlargement factor is 6). The views in these two panes are updated automatically: changes you make in one pane are immediately shown in the other. The two panes make it easy for you to work on an enlarged “picture” of your bitmap, in which you can distinguish individual pixels and, at the same time, observe the effect of your work on the actual-size view of the image.

If the bitmap is 200 x 200 pixels or larger, however, only one pane is displayed initially. Move the splitter to display both panes.

You can use the two panes in other ways. For example, you might enlarge the smaller pane and use the two panes to show different regions of a large bitmap.

You move the splitter to change the relative sizes of the panes. The splitter can move all the way to either side if you want to work on only one pane.

## Selecting Panes

- ▶ **To select an image-editor pane:**
  - Press TAB or F6, or click anywhere on the pane. The selection border then shows that the pane is active.

## Sizing Image-Editor Panes

- ▶ **To size the image-editor panes:**
  1. Position the cursor on the splitter.  
The cursor turns into a two-headed arrow (see Figure 7.5).
  2. Drag the splitter to the right or to the left.

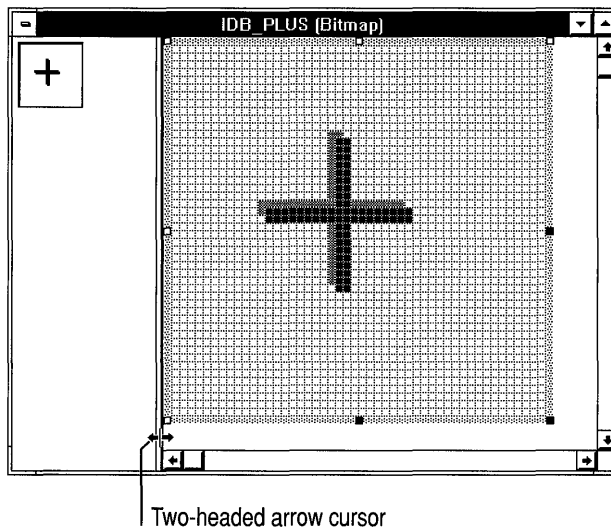


Figure 7.5 Moving the Image-Editor Pane Splitter

## Zooming In and Out

By default, the graphics editor displays the view in the left pane at actual size and the view in the right pane at 6 times actual size. There are several ways to change the magnification factor—the ratio between actual and displayed size—of the image you are editing and to toggle between an actual-size view and an enlarged view:

- The Zoom tool lets you select a magnification factor and choose in advance the portion of the image that appears in the “zoomed” view.
- Zoom In/Out on the Image menu toggles between the enlarged view and the actual-size view.
- Zoom Under Cursor, available from the keyboard only, centers the part of the image that is under the mouse cursor in the image-editor pane.

### ► To zoom in or out on an image-editor pane:

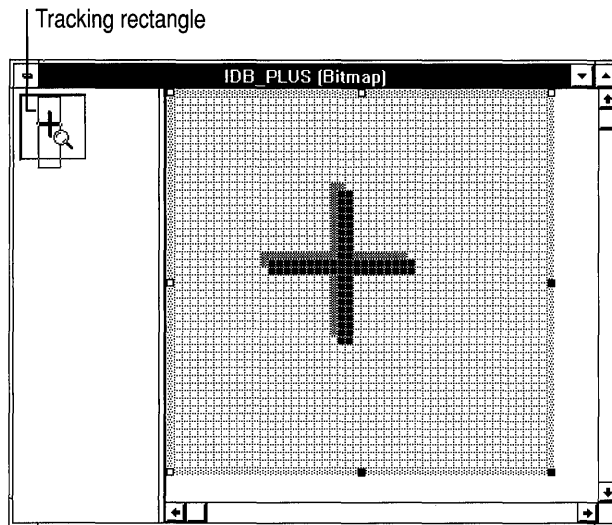


1. On the toolbox, click the Zoom tool (Z).
2. Position the mouse cursor over the pane you want to zoom in or out.

If you are zooming in on an actual-size view and the magnified view will not fit in the pane, a tracking rectangle shows the part of the bitmap that will appear in the magnified view (see Figure 7.6). Position the tracking rectangle over the area you want to view.

3. Click the pane you want to zoom in or out.

If you have used the Zoom Decrease or Zoom Increase command (see “Changing the Magnification Factor”) to change the zoom factor for the active pane, the current factor is used. Otherwise the default factor of 6 is used.



**Figure 7.6** Zooming in on the Actual-Size View

- ▶ **To zoom in on the part of the image under the cursor:**
  1. Position the mouse cursor on the point that you want to have centered in the zoomed view.
  2. Press M.
- ▶ **To toggle the magnification factor:**
  - Press M.

## Changing the Magnification Factor

The magnification factor is the ratio between actual size of the bitmap and the displayed size. The default is 6, and the range is from 1 to 10.

- ▶ **To change the magnification factor:**
  1. Select the image-editor pane whose magnification factor you want to change.
  2. On the toolbox, choose the Zoom tool (Z).





The cursor changes to the Zoom tool, and magnification-factor options appear in the option selector. If the current magnification factor matches an option, that option is highlighted.

3. Click the desired magnification factor.
- Or–
1. Select the image-editor pane whose magnification factor you want to change.
  2. Press > (SHIFT+PERIOD) to increase the magnification factor, or < (SHIFT+COMMA) to decrease the magnification factor.

## Displaying and Hiding the Pixel Grid

For all image-editor panes with a magnification factor of 4 or greater, you can display a grid that delimits the individual pixels in the image (see “Changing the Magnification Factor, page 99).

- ▶ **To display or hide the pixel grid:**
  1. From the Image menu, choose Grid Settings.
  2. In the Grid Settings dialog box, check the Pixel Grid box to display the grid, or clear the box to hide the grid.
  3. Click OK.

–Or–

  - Press G to toggle the grid display.

You can also display a tile grid to simplify editing toolbar graphics (see “Editing Toolbar Graphics,” page 104).

## Editing Icons and Cursors

Icons and cursors are like bitmaps, and you edit them in the same ways. But icons and cursors have attributes that distinguish them from bitmaps. For example, each icon or cursor resource can contain multiple images for different display devices. In addition, cursors have a “hotspot”—the location Windows uses to track the cursor’s position.

## Creating and Selecting Images

When you create a new icon or cursor, the graphics editor first creates an image for the VGA. The image is initially filled with the “screen” (transparent) color. If the image is a cursor, the hotspot is initially the upper-left corner (coordinates 0,0).

By default, the graphics editor supports the creation of images for the following devices:

Device	Colors	Width	Height
EGA/VGA	16	32	32
Monochrome	2	32	32
CGA	2	32	16

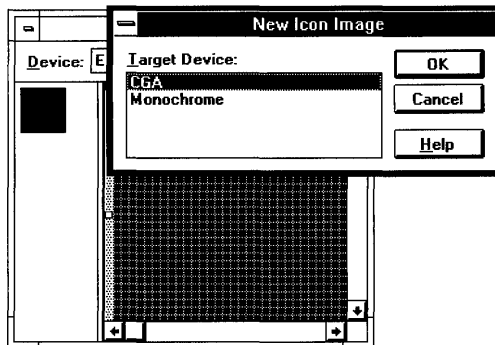
You can create images for other devices if you first create descriptions for those devices in App Studio's configuration file, APSTUDIO.INI (see "Editing Device Descriptions," page 102).

► **To create a new image:**



1. On the control bar of the image editor window, click the Device Image button (INS).

The New Cursor Image or New Icon Image dialog box appears (see Figure 7.7). This dialog box lists all devices for which there is not an image in the current resource.



**Figure 7.7** New Icon Image Dialog Box

2. Select the device from the dialog box.
3. Click OK.
4. Edit the new image. In many cases, you can copy an image for another device and paste it into the new image.

When you open an existing icon or cursor, the image most closely matching the current display device is opened by default.

► **To select a device image:**

- On the control bar of the image editor window, select a device image from the Device list box.

► **To delete a device image:**

1. Display the device image you want to delete.
2. From the Resource menu, choose Delete Device Image.

If this is the last image for the icon or cursor, App Studio displays a message box informing you that the resource will be deleted. Choose OK to delete the resource or Cancel to not delete the image. You cannot delete the last remaining image of a resource opened for stand-alone editing.

3. Choose OK to delete the image or Cancel to cancel the operation.

If the image is the last one in the resource, the resource is also deleted from the resource file.

## Editing Device Descriptions

App Studio's initialization file, APSTUDIO.INI, contains information about the display devices for which App Studio can create icons and cursors. Display devices are described in the sections of the file labeled [IconDevices] and [CursorDevices]. Each device description has the form:

```
DeviceName = ColorCount, Width, Height
```

The ColorCount entry must be 2 or 16, and the Width and Height must each be less than or equal to 999.

If APSTUDIO.INI does not contain an [IconDevices] section, the following device descriptions are used by default:

```
EGA/VGA=16, 32, 32  
Monochrome=2, 32, 32  
CGA=2, 32, 16
```

If APSTUDIO.INI does not contain a [CursorDevices] section, the following device description is used by default:

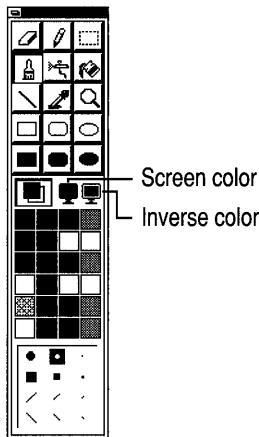
```
Monochrome=2, 32, 32
```

► **To add device descriptions to APSTUDIO.INI:**

1. Open APSTUDIO.INI using Notepad or another text editor.
2. Create a section label of the form [IconDevices] or [CursorDevices], if the appropriate section label does not already exist.
3. Under the section label, enter device descriptions in the form  
DeviceName = ColorCount, Width, Height
4. Close APSTUDIO.INI.
5. Restart App Studio to make the new device descriptions available.

## Drawing with Screen and Inverse Colors

Icon and cursor images are rectangular, but many do not appear so because parts of the image are “transparent”: the underlying image on the screen shows through the icon or cursor. And when you drag an icon, parts of the image may appear in an inverted color. You create these effects by choosing screen- and inverse-color options from the color indicator on the graphics palette, as shown in Figure 7.8.



**Figure 7.8** Selectors for Screen Color and Inverse Color

► **To create transparent or inverse regions in an icon or cursor:**

1. On the graphics palette, click a selector:



- The screen-color selector



- The inverse-color selector

2. Apply the screen or inverse color, using the methods described in “Drawing and Painting,” page 85.

The screen and inverse “colors” you apply to icons and cursors are really attributes. The colors indicate parts of the image possessing those attributes. You can change the colors that represent the screen- and inverse-color attributes for your convenience in editing. These changes do not affect the appearance of the icon or cursor in your application.

► **To change the colors representing screen color and inverse color:**

1. On the graphics palette, double-click the screen color or inverse color indicator.

The Color dialog box appears.

2. Choose a new color from the Color dialog box.

## Setting a Cursor's Hotspot

The hotspot is the point to which Windows refers in tracking the cursor's position. By default, the hotspot is set to the upper-left corner (coordinates 0,0). The General property page and the image-editor control bar show the hotspot coordinates.

► **To set a cursor's hotspot:**



1. On the control bar of the image editor window, click the Hotspot button or from the Image menu, choose Set Cursor Hotspot.
2. Click the pixel you want to designate as the cursor's hotspot.

## Editing Toolbar Graphics

App Studio's graphics editor has special features to simplify creation of tiled graphics for toolbar buttons (see the class **CToolBar** in the *Class Library Reference*). All the button bitmaps for a toolbar are taken from one bitmap, which must contain one image for each button. All images must be the same size: the default is 16 x 15 pixels. Images must be side by side in the bitmap. Typically the order of the images in the bitmap is the order in which they are drawn on the screen.

Each button has one image. The various button states and styles (pressed, up, down, disabled, disabled down, and indeterminate) are generated from that one image. Although bitmaps can be any color, best results are achieved with images in black and shades of gray. Figure 7.9 shows a typical toolbar bitmap.



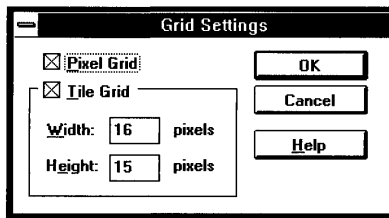
**Figure 7.9 Typical Toolbar Graphic**

The graphics editor can superimpose a tile grid on the view of a bitmap. The tile grid delimits sections of the bitmap in dimensions you specify so that you can properly position tiled images within the bitmap. When the tile grid is displayed, the graphics editor's behavior changes in ways that simplify resizing a tiled bitmap.

► **To set the tile-grid dimensions:**

1. From the Image menu, choose Grid Settings.

The Grid Settings dialog box appears (see Figure 7.10).



**Figure 7.10 Grid Settings Dialog Box**

2. In the Grid Settings dialog box, select the Pixel Grid check box if it is not already checked.

You cannot set the Tile Grid dimensions if the Pixel Grid check box is not checked.

3. Check the Tile Grid check box.
4. If necessary, edit the Width and Height settings.

Typically, these settings are the same as the width and height of your button bitmaps.

5. Click OK.

The image editor window now displays a tile grid, as in Figure 7.11.

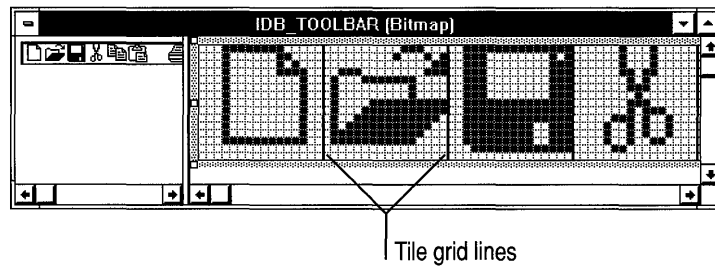


Figure 7.11 Image Editor Window with a Tile Grid

► **To toggle the display of the tile grid:**

- Press CTRL+G.

---

**Tip** When the tile grid is displayed, the sizing handles on the selection border behave differently from when the tile grid is not displayed (see “Resizing a Bitmap,” page 94). When you drag the sizing handles, the bitmap is resized in increments of the tile dimensions.

---

► **To create a new toolbar button:**

1. Display the tile grid.
2. If necessary, select a background color for the new button.
3. To resize the bitmap, drag the sizing handle on the right side of the bitmap to the right (see Figure 7.12). The bitmap's width is increased by an amount equal to the tile grid's Width setting.

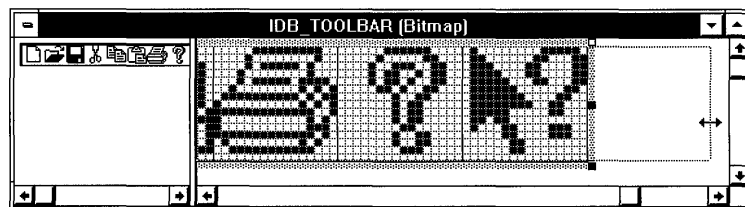


Figure 7.12 Resizing the Toolbar Bitmap

4. If the new button is to be added at a position other than the right end of the bitmap, make room for it:
  - Select the rightmost button images, including the one at the new button's position (see Figure 7.13).

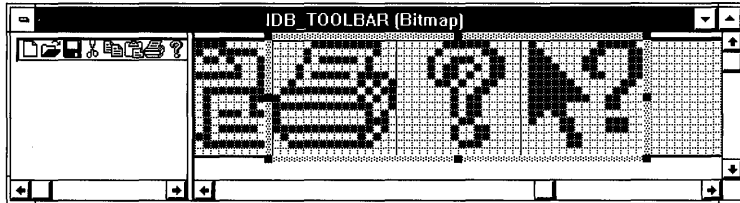


Figure 7.13 Selecting Button Images

- Drag the selection to the right, using the tile grid to align the images (see figure 7.14).
5. Draw the new button image in the space you have created.

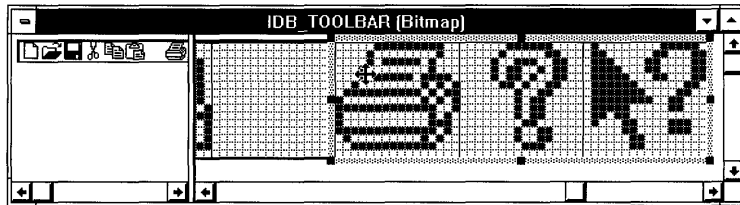


Figure 7.14 Moving the Selected Button Images

## Managing Colors and Palettes

The graphics editor's color palette displays 24 of the 48 "ready-made" colors that are available. In addition to the ready-made colors, you can create your own custom colors. Color-palette selections can be saved to disk and reloaded as needed.



## Creating Color Palettes

► **To change colors on the color palette:**

1. Double-click the color you want to change, or click the color and then from the Image menu, choose Edit Foreground Color.

The Color dialog box appears (see Figure 7.15). The color you are changing is outlined.

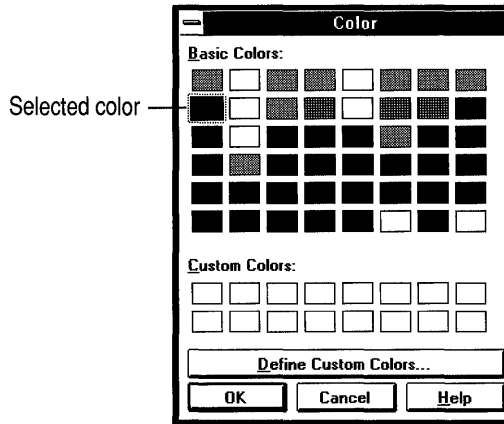


Figure 7.15 Color Dialog Box

2. Click the color you want to assign in place of the current color.
3. Click OK.

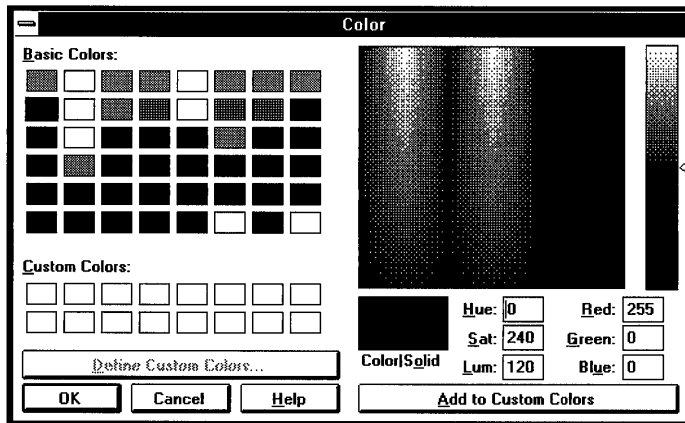
► **To create a custom color:**

1. Double-click on the color you want to change, or from the Image menu, choose Edit Foreground Color.

The Color dialog box appears.

2. Choose Define Custom Colors.

The custom-color dialog box then appears (see Figure 7.16).



**Figure 7.16 Custom-Colors Dialog Box**

3. Click the Custom Colors box in which you want to store the new custom color.
4. Define the color by typing in RGB or HSL values, or by moving the cross hairs on the color box.
5. Set the luminance by moving the slider on the luminance bar.
6. Many custom colors are dithered. If you want the solid color closest to the dithered color, choose the Color|Solid control. (If you later decide you want the dithered color, move the slider or the cross hairs again to restore the dithering.)
7. Choose a disposition for the new color you have created:
  - To add the new color to the Custom Colors control, click Add to Custom Colors.
  - To substitute the new custom color for the current palette color, choose OK. The color will be available for later use only if you add it to the Custom Colors control.
  - To save the custom color for later use without replacing the current palette color, add the color to the Custom Colors control. Choose Cancel.

Custom-color definitions are saved in APSTUDIO.INI and are automatically loaded the next time you start App Studio.

## Saving and Loading Palettes

---

**Tip** Since the graphics editor has no means to restore the default color palette, save the default color palette under a name such as STANDARD.PAL or DEFAULT.PAL so that you can restore the defaults without having to quit and restart App Studio.

---

▶ **To save a custom palette:**

1. From the Image menu, choose Save Palette Colors.
2. Use the Save Palette Colors dialog box to navigate directories and enter a file name.

▶ **To load a custom palette:**

1. From the Image menu, choose Get Palette Colors.
2. Use the Get Palette Colors dialog box to navigate directories and choose a file name.

---

## CHAPTER 8

# Using the Binary Data Editor

The App Studio binary data editor allows you to edit a resource at the binary level in either hexadecimal or ASCII format.

You should use the binary editor only when you need to view or make minor changes to custom resources or resource types not supported by App Studio.

---

**Caution** Editing nondata resources in the binary data editor can corrupt the resource. A corrupted resource can cause App Studio and Windows to behave in unexpected ways.

---

## Creating a New Data Resource or Custom Resource

The binary data editor allows you to browse resources at a binary level and change isolated bytes. However, to create custom or data resources, place them in a separate file using normal .RC file syntax. Then add the file to your project's resource file with the File menu Set Includes command. For more information on working with included resource files, see Chapter 2.

## Opening the Binary Data Editor

- ▶ **To select a resource for binary editing:**
  1. Move to the resource browser window.
  2. In the Type box, choose the resource type you want to edit.
  3. In the Resources box, select the specific resource you want to edit.
  4. From the Resource menu, choose Edit Binary Data (CTRL+B).

The binary data window appears (Figure 8.1).

**Note** If you use the resource browser window to open a resource whose format App Studio does not recognize (such as a VERSION, RCDATA, or custom resource), App Studio automatically opens the resource in the binary data editor.

Offset	Hexadecimal value	ASCII value
9999 [Data]		
000000	41 20 75 73 65 72 2D 64 65 66 69 6E 65 64 20 72	A user-defined r
000010	65 73 6F 75 72 63 65 20 73 74 61 74 65 6D 65 6E	esource statemen
000020	74 20 73 70 65 63 69 66 69 65 73 20 61 20 72 65	t specifies a re
000030	73 6F 75 72 63 65 20 74 68 61 74 20 63 6F 6E 74	source that cont
000040	61 69 6E 73 20 61 70 70 6C 69 63 61 74 69 6F 6E	ains application
000050	2D 73 70 65 63 69 66 69 63 20 64 61 74 61 2E 00	-specific data..

**Figure 8.1** Binary Data Editor

**Note** If you want to use the binary data editor on a resource already being edited in another App Studio editor window, close the other editor window first.

## Editing Data

► **To edit a resource in the binary data editor:**

1. Select the byte you want to edit.

The TAB key moves the focus between the hexadecimal and ASCII sections of the editor window. PAGE UP and PAGE DOWN move you through the resource one screen at a time. You can also move to the top of the resource with CTRL+HOME and to the end with CTRL+END.

2. Type the new value.

# Using ClassWizard

ClassWizard is like a programmer's assistant: it makes it easier for you to do certain routine tasks such as creating new classes, defining message handlers, and gathering data from controls in a dialog box or form view.

One of ClassWizard's main tasks is to work with Microsoft Foundation Class Library "message maps." A message map is a message-dispatch table that associates Windows messages with class member functions. This eliminates the cumbersome switch statements needed in traditional programming for Windows.

With ClassWizard, you can:

- Create new classes derived from any of the main framework base classes that handle Windows messages.
- Browse the Windows messages associated with windows, dialog boxes, controls, menu items, and accelerators.
- Create new message-handling member functions by clicking the messages you want to handle.
- See which messages have message handlers already defined and jump to the handler program code.
- Define member variables that automatically initialize, gather, and validate data entered into dialog boxes or form views.

---

**Important** ClassWizard never changes any of your code. It only works with the special-format message-map and "data-map" (see page 116) sections of your header and implementation files. The Microsoft Foundation Class Library uses this information for message handling and dialog data exchange and validation.

---

ClassWizard is used with Microsoft Foundation Class Library version 2 projects. For information on how to convert other projects (including Microsoft Foundation Class Library version 1 projects) for use with ClassWizard, see "Updating Existing Code for Use with ClassWizard" at the end of this chapter.

The first part of this chapter discusses three processes that can be automated with ClassWizard:

- Adding a new class to your application
- Mapping messages and commands to functions
- Working with dialog-box data

You can also update ClassWizard when you've made minor changes to the source code or rebuild the entire ClassWizard (.CLW) file when the changes you've made are more extensive.

While you can do any of these common programming functions without using ClassWizard, using ClassWizard together with the framework allows you to accomplish these common chores quickly and efficiently.

The last part of this chapter discusses how to update existing code for use with ClassWizard.

## Adding a New Class or Importing an Existing One

Use ClassWizard to add error-free class declarations to your project for classes that contain message-handling functions.

---

**Note** ClassWizard is only for use with user-interface classes that handle messages or manage dialog-box controls. To add a new class that does not handle messages, create the class directly in the Visual Workbench editor.

---

ClassWizard allows you to create classes derived from the Microsoft Foundation classes shown in Table 9.1:

**Table 9.1** Types of Classes Created in ClassWizard

Class	Description
<b>CDialog</b>	Dialog box
<b>CDocument</b>	Class for managing program data
<b>CFormView</b>	Window that can contain dialog-box controls
<b>CFrameWnd</b>	Main single document interface (SDI) frame window
<b>CMDIChildWnd</b>	Multiple document interface (MDI) child frame window
<b>CView</b>	Class for displaying program data
<b>CScrollView</b>	Scrolling window, derived from <b>CView</b>
<b>CWnd</b>	Custom window
<b>Splitter</b>	MDI child frame window containing a <b>CSplitterWnd</b>

For more information on these classes, see the *Class Library Reference*.

When you use ClassWizard to create a new class derived from one of the framework classes listed above, it automatically places a complete and functional class in the header (.H) and implementation (.CPP) files you specify. ClassWizard keeps track of the class's message-handling and data-exchange members, and later you can use it to update this information.

## Adding a New Class

### ► To add a new class to your project using ClassWizard:

1. In Visual Workbench: from the Browse menu, choose ClassWizard (CTRL+W).

–Or–



In App Studio: click the ClassWizard toolbar button, or from the Resource menu, choose ClassWizard (CTRL+W).

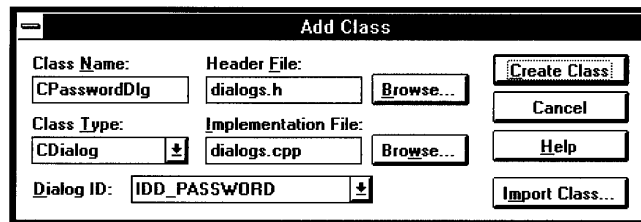
---

**Note** For classes associated with a dialog-box resource—classes derived from **CDialog** or **CFormView**—create the resource in App Studio before you use ClassWizard to create the class.

---

2. Choose Add Class. The Add Class dialog box appears (Figure 9.1).

In App Studio, if you are currently editing a dialog box that is not yet associated with a class, the Add Class dialog box appears automatically.



**Figure 9.1** Add Class Dialog Box

3. Type the name of your new class in the Class Name box and, in the boxes provided, type the names of the header and implementation files where the class is to be defined.
4. In the Class Type box, select the name of the class or class type from which your current class is to be derived (Table 9.1).
5. In the Dialog ID box, select the name of the resource identifier from the list. If you are in App Studio, have a dialog box selected, and are creating a class based on **CDialog** or **CFormView**, the identifier is already filled in for you.
6. Choose Create Class to create the class in the files you specified.



For a specific example of how this general procedure works to create a new dialog-box class, see page 125.

When you use ClassWizard to create a new class, it adds skeletal information on the new class to both the header and implementation files. If you specify filenames that don't yet exist, ClassWizard creates the new files and adds them to your project if Visual Workbench is running.

---

**Note** ClassWizard only updates your project file if Visual Workbench is running. If it is not, you need to add the files to the project yourself.

---

When you add a new class using ClassWizard, special-format comments are placed in your code to mark the sections of the header and implementation files that ClassWizard edits. ClassWizard never modifies code that is outside these commented sections.

For most classes, there are two related sections of code that ClassWizard edits: the member-function definitions in the class header file and the message-map entries in the class implementation file.

The ClassWizard comments in the header file look like this:

```
//{{AFX_MSG(<classname>)
    // ClassWizard adds member functions here
//}}AFX_MSG
```

The ClassWizard section in the implementation file is set off with comments that look like this:

```
//{{AFX_MSG_MAP(<classname>)
    // ClassWizard adds message map macros here
//}}AFX_MSG_MAP
```

For more information about these comments, see “Updating Existing Code for Use with ClassWizard” on page 133.

For dialog boxes, ClassWizard creates and edits three other sections that are marked with special format comments:

- Member variable declarations in the class header file  
(//{{AFX\_DATA...//}}AFX\_DATA)
- Member variable initialization in the class implementation file  
(//{{AFX\_DATA\_INIT...//}}AFX\_DATA\_INIT)
- Data-exchange macros in the implementation file  
(//{{AFX\_DATA\_INIT...//}}AFX\_DATA\_INIT)

For more information see “Working with Dialog-Box Data” on page 121.

## Importing Existing Classes

If you add a message-handling class to your current project by copying code from another project, you can update ClassWizard so that it recognizes the new class.

---

**Note** If the new code you have copied contains more than two or three new message-handling classes, you can save time by rebuilding the ClassWizard file completely rather than importing each new class individually. See page 132 for more information.

---

- **To add a class copied in from another project to the ClassWizard file:**
1. If the code you are importing doesn't already have ClassWizard comments in it, add the special-format comments ClassWizard uses to locate message-map entries. See page ??? for more information.
  2. Bring up the main ClassWizard dialog box and choose Add Class. When the Add Class dialog box appears, choose Import Class.
  3. In the Import Class dialog box (Figure 9.2), type in the name of the new class and the name of the header and implementation files where it can be found.
  4. Choose OK to add the new class to the ClassWizard file.

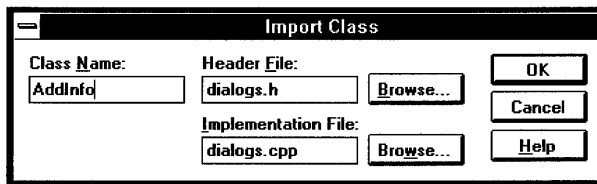


Figure 9.2 Import Class Dialog Box

## Mapping Messages to Functions

ClassWizard lets you browse the messages associated with a user-interface object in your application and quickly define message-handling functions for them. ClassWizard also automatically updates the message-dispatch table, or message map, when you use ClassWizard to define message-handling functions.

The procedure for handling a message is very similar, regardless of the kind of user-interface object you are working with. Table 9.2 shows the types of objects you work with in ClassWizard and the types of messages associated with them.

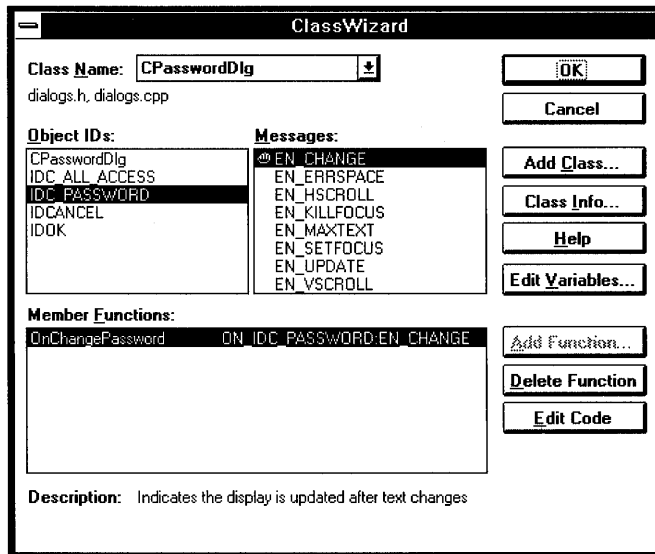
**Table 9.2 User-Interface Objects and Associated Messages**

Object ID	Messages
Class name, representing the containing window (Table 9.1)	Windows messages appropriate to a <b>CWnd</b> -derived class: a dialog box, window, child window, MDI child window, or topmost frame window
Menu or accelerator identifier	COMMAND message (executes the program function) UPDATE_COMMAND_UI message (dynamically updates the menu item)
Control identifier	Control notification messages for the selected control type

## Defining Message Handlers

After creating a class with ClassWizard, or importing an existing class, you can use ClassWizard to browse the messages or control notifications associated with each object and to create handler routines (member functions) as appropriate.

The general procedure for using ClassWizard to define a message or command handler is as follows (see Figure 9.3):

**Figure 9.3 Defining a Message or Command Handler**

► **To define a message handler with ClassWizard:**

1. In Visual Workbench: from the Browse menu, choose ClassWizard (CTRL+W).

–Or–



In App Studio: select the user-interface object you want to work with. Then click the ClassWizard toolbar button, or from the Resource menu, choose ClassWizard (CTRL+W).

The main ClassWizard dialog box appears. If you are in Visual Workbench, ClassWizard displays information about the currently selected class name or the class you last edited with ClassWizard. If you are in App Studio, you see information about the user-interface object that is currently selected.

2. In the Object IDs box, select the name of the user-interface object you want to define a message handler for. Table 9.2 shows the types of objects that will appear in the Object IDs box and the messages appropriate to each type.
3. In the Messages box, select the message for which you want to define a handler and choose Add Function, or double-click the message name.

You can only select messages that do not already have a handler defined. A message with a handler already defined has a picture of a small hand next to it.

---

**Note** The messages you see in the Messages box are those most appropriate to your class. You can change which messages you see by choosing the Class Info command and selecting a new set of messages in the Message Filter box. For information on handling custom messages, see Technical Note 6, which can be found in MSVC\HELP\MFCNOTES.HLP.

---

For messages that don't already have a handler defined in their base class, the Add Member Function dialog box appears.

---

**Tip** Selecting a message displays a brief description of it at the bottom of the ClassWizard dialog box. You can get a more complete description of the message by pressing F1.

---

4. For messages that don't already have a handler defined in their base class, type in the member function name you want and press ENTER, or press ENTER to accept the name proposed by ClassWizard. This returns you to the main ClassWizard dialog box.

A picture of a hand appears next to the message name to show that a member function is defined to handle the message. The name of the new message-handling function appears in the Member Functions box.

At this point you have several options. You can:

- Add more message handlers.

- Choose Edit Code to jump to the empty ClassWizard-generated function in your source code and begin defining the function's behavior.
- Choose OK to close ClassWizard. You can return to ClassWizard any time during the development process.
- Choose Cancel to cancel the changes you have made.

When you choose OK or Edit Code, ClassWizard updates your source code as follows:

- A function declaration is inserted into the header file.
- A complete, correct function definition with a skeletal implementation is inserted into the implementation file.
- The class's message map is updated to include the new message-handling function.

## Deleting Message Handlers

Once you have defined a message handler with ClassWizard, you can also use ClassWizard to delete it. However, you must remove the function definition, as well as any references to the function, from the implementation file yourself. ClassWizard never makes changes to your implementation code—only to the message and data maps.

► **To delete a message-handling function:**

1. In the main ClassWizard dialog box, from the Class Name box, select the class containing the message-handling function you want to delete.
2. In the Member Functions box, select the name of the member function to delete.
3. Use the Edit Code command (see the next section) to open the implementation file containing the member function in Visual Workbench. Delete the function header and function body, or copy it to a file not in the current project.
4. Return to ClassWizard and choose Remove Function. This deletes the member function from the message-map entries for that class in both the header and implementation files.

## Jumping to Source Code from ClassWizard

Once you have defined a procedure with ClassWizard you can use the Edit Code command to jump to the member-function definition and begin to add or modify code. The following procedure assumes that Visual Workbench is running.

- ▶ **To use the ClassWizard Edit Code command (available only when Visual Workbench is running):**
  1. In the Member Functions box, select the function you want to edit.
  2. Choose Edit Code (see Figure 9.3) or double-click the function name.  
ClassWizard switches to Visual Workbench and moves to the beginning of the member function.

## Working with Dialog-Box Data

ClassWizard offers an easy way to take advantage of the dialog data exchange (DDX) and dialog data validation (DDV) capabilities of the Microsoft Foundation classes.

To use DDX, you define member variables in the dialog-box class and associate each of them with a dialog-box control. The framework transfers any initial values to the controls when the dialog box is displayed. It then updates the variables with user-entered data when the user clicks OK to dismiss the dialog box. You can also use DDX with VBX controls.

With DDV, dialog-box information entered by the user is validated automatically. You can set the validation boundaries: the maximum length for string values in an edit-box control or the minimum or maximum numeric values when you expect a number to be entered. You can also use ClassWizard to connect dialog-box controls to your own custom data-validation routines.

For an example of how to use DDX and DDV, see “Example: Part 2: Using DDX/DDV” on page 128.

## Dialog Data Exchange

ClassWizard lets you create variables that use the framework’s automatic dialog data exchange capabilities. For each dialog-box control you want to set an initial value for or gather data from, use ClassWizard to define a data member in the dialog class. The framework then transfers the initial value of each variable to the dialog box when it is created and updates each member variable when the dialog box is dismissed.

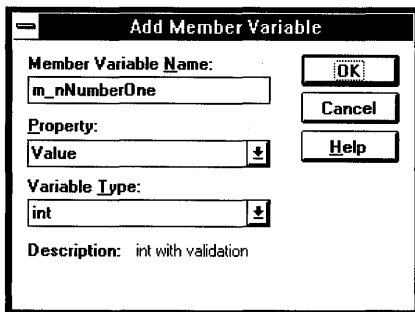
---

**Note** You can also use **CWnd::UpdateData** to transfer data back and forth between controls and member variables while a dialog box is open.

---

► **To define data members for dialog data exchange:**

1. Create your dialog box in App Studio, place in it the controls you want, and set the appropriate control styles in the Properties window. Then use ClassWizard to define a new dialog-box class (page 114).
2. In the ClassWizard dialog box, choose Edit Variables. The Edit Member Variables dialog box appears.
3. Select the control for which you want to set up DDX and choose Add Variable. The Add Member Variable dialog box appears (see Figure 9.4).



**Figure 9.4** Add Variable Member Dialog Box

4. In the Member Variable Name box, type the name of the new variable. ClassWizard provides the `m_` prefix to identify it as a member variable.
5. In the Property box, select whether this variable is a Value variable or a Control variable.

For standard Windows controls, choose Value in the Property box to create a variable that contains the control's text or status as entered by the user. The framework automatically converts the control's data to the data type selected in the Variable Type box (see Table 9.3).

For VBX controls, use the Property box to create a member variable for any of a VBX control's properties. The appropriate data type for a given property is reflected in the Variable Type box.

You can also choose Control in the Property box to create a Control variable that gives you access to the control itself (see Table 9.4).

6. In the Variable Type box, choose from a list of variable types appropriate to the control (see Table 9.3).

Once you've defined a DDX Value variable for a standard Windows control, the framework automatically initializes and updates the variable for you.

Table 9.3 shows the type of DDX Value variables ClassWizard initially provides. To create additional variable types, see Technical Note 26, which can be found in MSVCHELPMFCNOTES.HLP.

**Table 9.3 DDX Variable Types for the Value Property**

<b>Control</b>	<b>Variable Type</b>
Edit box	CString, int, UINT, long, DWORD, float, double
Normal check box	BOOL
Three-state check box	int
Radio button (first in group)	int
Nonsorted list box	CString, int
Drop list combo box	CString, int
All other list box and combo box types	CString
VBX control	CString, int, long, BOOL, float, COLORREF

The following additional notes apply to using DDX Value variables:

- Possible values for three-state check boxes are 0 (off), 1 (on), and 2 (indeterminate).
- Values for a group of radio buttons range from 0 for the first button in the group to  $n-1$  for a group with  $n$  buttons. A value of  $-1$  indicates that no buttons are selected.
- When you are using a group of check boxes or radio buttons with a DDX variable, make sure the Auto property is set for all the controls in the group.
- Set the Group property for the first radio button in a group, and make sure all the other radio buttons immediately follow the first button in the tab order.
- To use an integer value with a combo box or list box, make sure the Sort property is turned off on the control's Styles property page.
- Some VBX custom-control properties are read-only, and their initial values cannot be set with DDX.



Table 9.4 shows the type of DDX Control variables you can define with ClassWizard.

**Table 9.4 DDX Variable Types Defined with the Control Property**

Control	Variable Type
Edit box	CEdit
Check box	CButton
Radio button	CButton
Pushbutton	CButton
List box	CListBox
Combo box or drop list combo box	CComboBox
Static text	CStatic
VBX custom control	CVBControl*

## Using DDX Variables

You can set the initial value of DDX variables by editing the initialization code that ClassWizard places in the constructor for the dialog-box class. (ClassWizard does not disturb these initialization statements once they have been put in place.) The framework transfers the values to the dialog box when it is created.

To see what the user typed once the dialog box is dismissed, access the values of the DDX variables just as you would any C++ member variable.

## Dialog Data Validation

By default, ClassWizard supports the types of data validation shown in Table 9.5, but you can add additional types (see Technical Note 26, which can be found in `MSVC\HELP\MFCNOTES.HLP`).

**Table 9.5 DDV Variable Types**

Variable Type	Data Validation
CString	Maximum length
Numeric (int, UINT, long, DWORD, float, double)	Minimum value, maximum value

You can define the maximum length for a CString DDX variable or the minimum or maximum values for a numeric DDX variable at the time you create it. For example, the ClassWizard Edit Member Variables dialog box in Figure 9.5 shows a

DDX integer variable associated with an edit box. The variable has a minimum value of 1 and a maximum value of 10.

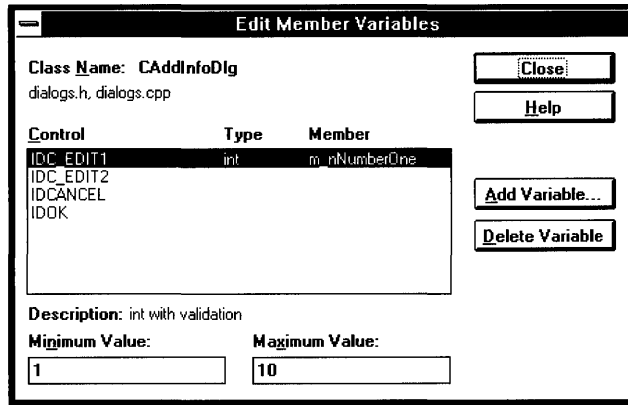


Figure 9.5 Dialog Data Validation

At run time, if the value entered by the user exceeds the range you specify, the framework automatically displays a message box asking the user to reenter the value. The validation of DDX variables takes place all at once when the user chooses OK to accept the entries in the dialog box.

## Custom Data Exchange and Validation

Although you can write a dialog-box class that gathers and validates its own dialog-box data using custom message handlers, you may find that you have routines for data exchange and validation (containing your own variable types and data formats) that you want to use over and over. You can extend the ClassWizard user interface to reuse your own DDX and DDV routines.

For more information, see Technical Note 26, which can be found in MSVCHELPMFCNOTES.HLP.

## Example: Building a Dialog Box with ClassWizard

In this section you'll see how to define a message-handling function for a simple dialog box that prompts a user to type in a password. You'll then see how to use ClassWizard to take advantage of the framework's built-in DDX and DDV routines. This example uses the procedures described in the previous sections.

## Example, Part 1: Defining a Message Handler for a Dialog-Box Control

### Step One: Create the Dialog-Box Resource

To begin, assume that you have created the new dialog box in App Studio (see Chapter 3) and are ready to define message handlers for it. The App Studio screen at the beginning of the procedure looks like Figure 9.6. The dialog box has an ID of `ID_PASSWORD`. The edit-box control that will accept the password has an ID of `IDC_PASSWORD`. The check box has an ID of `IDC_ALL_ACCESS`, and its Disabled property is selected.

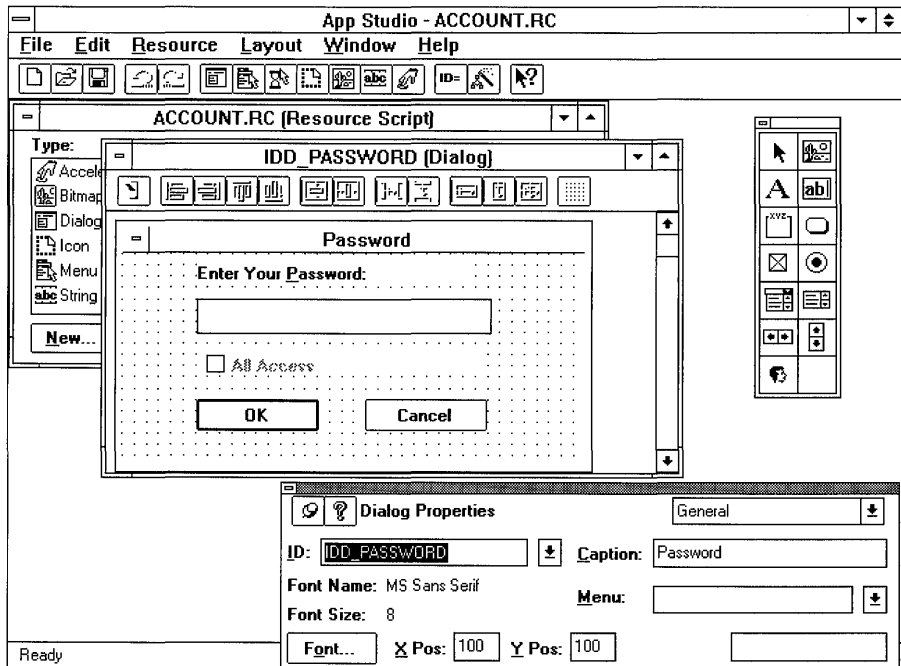


Figure 9.6 Password Dialog Box

### Step Two: Define The Dialog-Box Class

On the App Studio toolbar, click the ClassWizard button. The dialog box you created is open in the dialog editor window so ClassWizard knows what resource identifier to work with. Since the new dialog box isn't yet associated with a class, ClassWizard displays the Add Class dialog box. You supply the name of the new class and the names of its header and implementation files (Figure 9.1).

## Step Three: Define the Message Handler

Next, you want to define a handler to enable the check-box control in the dialog box as soon as the user types something into the edit-box control. Since the edit-box control sends an `EN_CHANGE` message to the dialog box every time the contents of the control is modified, you select the control's identifier, `IDC_PASSWORD`, in the Object IDs box and double-click `EN_CHANGE` in the Messages box to define the handler.

Since you've given the edit box a symbol name of `IDC_PASSWORD`, ClassWizard suggests `OnChangePassword` as the name of the message-handling function. When you press `ENTER`, ClassWizard accepts the name and creates the new member function. The ClassWizard dialog box now looks like Figure 9.3.

## Step Four: Begin Filling in the Message Handler

You now want to fill in the function body, so you select the new member function (`OnChangePassword`) and choose `Edit Code`. (The `Edit Code` command is not available unless Visual Workbench is running.) ClassWizard switches to Visual Workbench and opens `DIALOGS.CPP` to the appropriate location in the source code. The following empty function is already in place:

```
void CPasswordDlg::OnChangePassword()
{
    // TODO: Add your control notification handler code here
}
```

The class's message map has been updated to reflect the message handler (marked ▶):

```
BEGIN_MESSAGE_MAP(CPasswordDlg, CDialog)
    //{AFX_MSG_MAP(CPasswordDlg)
    ON_EN_CHANGE(IDC_PASSWORD, OnChangePassword)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()
```

The header file, `DIALOGS.H`, has been updated as well (marked ▶):

```
// Generated message map functions
//{{AFX_MSG(CPasswordDlg)
afx_msg void OnChangePassword();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
```

You are now ready to add the necessary `EN_CHANGE` message-handling code to `DIALOGS.CPP`. You'll see how to do this at the end of the second part of this example, where the `Add Variables` dialog box is used to create `Control` variables that allow you to enable the check box whenever the edit box has something in it.

## Example, Part 2: Using DDX/DDV

Now assume you want to set up a data member in the `CPasswordDlg` class to contain the password the user types in. You also want to set the password to an initial value of “INVALID” and make sure that it’s no more than eight characters long.

### Step One: Define the Member Variable Using ClassWizard

The first step is to set up a member variable in the dialog-box class. You bring up ClassWizard with the Password dialog box selected, making sure the identifier for the text control that will contain the password (in this case `IDC_PASSWORD`) is selected. Choose Edit Variables. The Edit Member Variables dialog box appears.

You then choose Add Variable and, in the Add Member Variable dialog box that appears, fill in the name of the member variable (`m_strPassword`, in this case). Accept the default values in the Property and Variable Type boxes. At this point, the Add Member Variable dialog box looks like Figure 9.7.

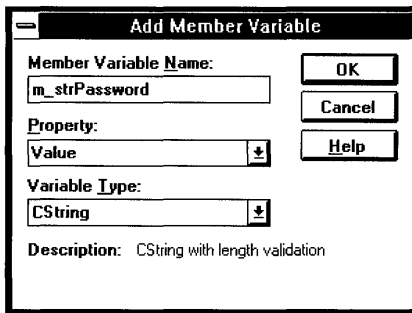


Figure 9.7 DDX/DDV Example: Add Member Variable Dialog Box

### Step Two: Set the Maximum Number of Characters for the Variable

When you choose OK after adding the member variable, ClassWizard returns to the Edit Member Variables dialog box. Since you have now defined a `CString` variable, ClassWizard adds an edit box at the bottom of the dialog box where you can type in the maximum number of characters for this variable. When the program is running and the user types in more than the number of characters you’ve specified, the framework displays a dialog box asking for the correct number of characters.

In this case, you want the password to be no more than eight characters long, so you type 8 in the Maximum Characters box. The Edit Member Variables Dialog Box now looks like Figure 9.8.

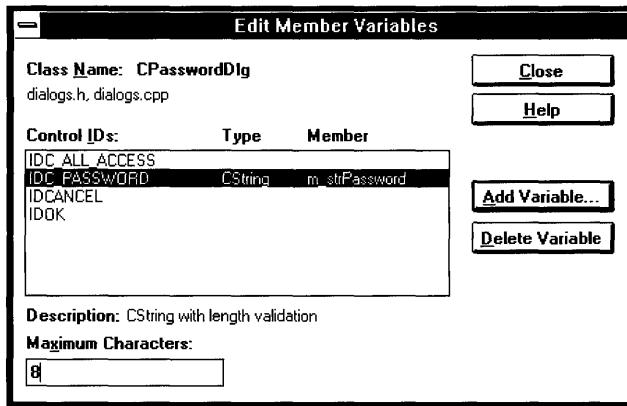


Figure 9.8 DDX/DDV Example: Using Built-in DDV for a CString Variable

ClassWizard adds information to the `DoDataExchange` member function of the dialog-box class in `DIALOGS.CPP` based on what you've entered (marked ▶):

```
void CPasswordDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPasswordDlg)
    DDX_Text(pDX, IDC_PASSWORD, m_strPassword);
    DDV_MaxChars(pDX, m_strPassword, 8);
    //}}AFX_DATA_MAP
}
```

### Step Three: Set the Initial Value for the Variable in the Dialog-Box Class Constructor

When you first defined the member variable `m_strPassword`, ClassWizard placed an initialization statement in the dialog-box class constructor that initialized the variable to an empty string (marked ▶):

```
CPasswordDlg::CPasswordDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CPasswordDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPasswordDlg)
    m_strPassword = "";
    //}}AFX_DATA_INIT
}
```

To change the initial value of the DDX variable, simply edit the `//{{AFX_DATA_INIT` section of the dialog-box class code so it reflects the initial value you want. For example, changing the line marked with ▶:

```

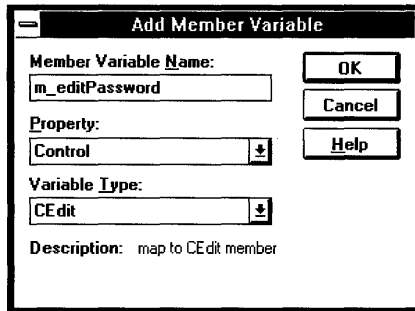
CPasswordDlg::CPasswordDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CPasswordDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CPasswordDlg)
    m_strPassword = "INVALID";
   //}}AFX_DATA_INIT
}

```

changes the initial value of the password variable to “INVALID.”

## Step Four: Create Control Variables to Use in the Message Handler

The final step is to create two DDX Control variables to use in enabling the All Access check box when something is typed into the Password box. In the Edit Member Variables dialog box, select the IDC\_PASSWORD control and choose Add Variable. The Add Member Variables dialog box appears. In the Property box, select Control, and in the Member Variable Name box, type in the name of the variable, in this case `m_editPassword`. The Add Member Variable dialog box now looks like Figure 9.9.



**Figure 9.9** DDX/DDV Example Defining a DDX Control Variable

Once you’ve defined a DDX Control variable for the All Access check box in a similar way, you’re ready to fill in the `EN_CHANGE` message-handling function. The completed function looks like this:

```

void CPasswordDlg::OnChangePassword()
{
    m_btnAllAccess.EnableWindow(
        m_editPassword.GetWindowTextLength() != 0 );
}

```

This completes the example. ClassWizard did most of the work so that you could concentrate on the important part, the actual message-handling code. You also have added a member variable to your dialog-box class that automatically takes

advantage of the framework's built-in routines for custom data exchange and validation.

## Keeping ClassWizard Updated When Code Changes

It's very likely that as your program develops you'll need to delete or modify a class, to delete some resources and add others, or to move a class from one source file to another. ClassWizard is designed to keep in sync with your code as you make these changes: it asks you for the updated information when you next edit the affected class.

ClassWizard stores the information about your application's classes in a file with the file extension .CLW. To accommodate source files that have changed, ClassWizard displays the Repair Class Information dialog box (Figure 9.10) whenever it finds that the information in the .CLW file is out of date.

The Repair Class Information dialog box has two main functions:

- Delete obsolete classes from the ClassWizard file.
- Update the ClassWizard file with the new name or location of classes that you have changed or moved.

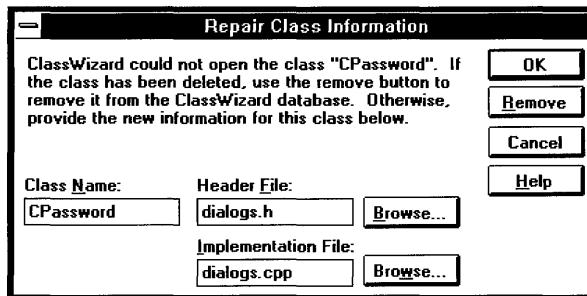


Figure 9.10 Repair Class Information Dialog Box

## Deleting Classes

When you delete a class from your header and implementation files, the information in the .CLW file needs to be updated as well. The next time you select that class in ClassWizard, you are prompted to update the information in the ClassWizard file.

- ▶ **To delete a message-handling class created with ClassWizard:**
  1. Remove the class from your header and implementation files.
  2. Open ClassWizard, and in the Class box, select the class you have modified.



ClassWizard displays a message box warning you that the class cannot be found. Then the Repair Class Information dialog box appears.

3. Choose Remove.

The class is deleted from the ClassWizard file.

## Renaming or Moving Classes

When you change the name of a class or move it from one implementation file to another, you're prompted to update the information in the .CLW file the next time you edit the class in ClassWizard.

► **To change the name of a class or move it from one file to another:**

1. Make the desired changes to your source files.

---

**Note** When you change the name of a class, remember to change it everywhere, including in the special-format comments ClassWizard uses. For example,

```
//{{AFX_MSG_MAP(OldClass) becomes //{{AFX_MSG_MAP(NewClass)
```

---

2. Invoke ClassWizard. In the Class Name box, choose the class you want to change from the list.

ClassWizard displays a message box warning you that the old class could not be found. When you choose OK, the Repair Class Information dialog box appears.

3. In the Class Name, Header File, and Implementation File boxes, supply the new information about the class. If necessary, use the appropriate Browse command to look for the correct name of the header or implementation file.
4. Choose OK to update the ClassWizard file.

## Rebuilding the ClassWizard (.CLW) File

If you have made numerous changes to your code or have added a large number of existing user-interface classes to your current project, you may find it convenient to rebuild the ClassWizard file from scratch rather than update it one class at a time. To do this, you delete your project's ClassWizard (.CLW) file and use a dialog box similar to the Visual Workbench's Edit Project dialog box to generate a new one. The newly-generated .CLW file contains information about all the classes that have the special-format ClassWizard comments.

► **To rebuild the ClassWizard file:**

1. Delete your project's current .CLW file.
2. Open the project resource file in App Studio so that information about your program's resources can be added to the file as well.

3. Invoke ClassWizard. A message box is displayed asking if you want to rebuild the ClassWizard file from your project files. Click Yes.
4. The Select Source Files dialog box appears (Figure 9.11). Use the File Name box and the Add, Add All, and Delete commands to create a list of project files in the Files in Project box.
5. When the list of project files is complete, click OK.  
ClassWizard generates a new .CLW file.

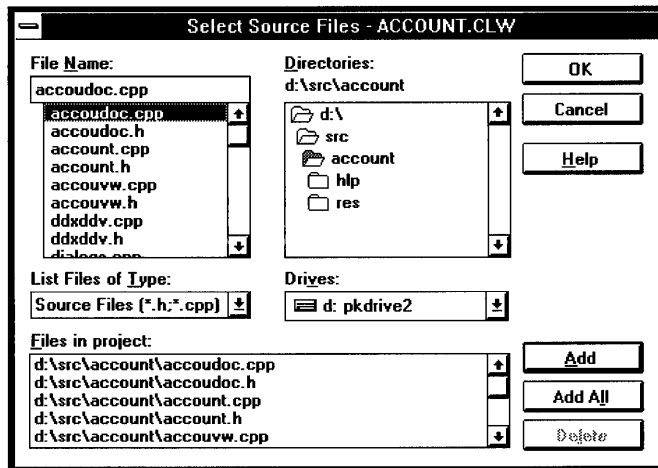


Figure 9.11 Select Files Dialog Box for Generating a New .CLW File

## Updating Existing Code for Use with ClassWizard

This section provides the basics for adding ClassWizard comments to existing code so that the code can be used with ClassWizard. Microsoft Foundation Class Library version 1 code is an example of code containing classes that need to be updated in this way. To update a class for use with ClassWizard, it must already have a message map defined.

For more detailed information on updating existing code, see Technical Note 19, which can be found in `MSVC\HELP\MFCNOTES.HLP`.

To use existing code with ClassWizard, place the special-format comments ClassWizard uses to locate message-map entries in both your header and implementation files.

In your header file, ClassWizard uses comments of the following form, placed before and after message-handler declarations:

```
//{{AFX_MSG(<classname>)
//message-handler declaration here
//}}AFX_MSG
```

The *classname* is the name of the class to which the message handler belongs. For example:

```
//{{AFX_MSG(CPasswordDlg)
afx_msg void OnChangePassword();
//}}AFX_MSG
```

In your implementation file, ClassWizard uses comments of the following form, placed before and after the message-map entries:

```
//{{AFX_MSG_MAP(<classname>)
// message map entries here
//}}AFX_MSG_MAP
```

For example:

```
BEGIN_MESSAGE_MAP(CPasswordDlg, CDialog)
//{{AFX_MSG_MAP(CPasswordDlg)
ON_EN_CHANGE(IDC_PASSWORD, OnChangePassword)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

You can also upgrade existing dialog classes for use with ClassWizard dialog data exchange and validation, although you may not need to do this if your existing dialog-box code already handles data exchange and validation to your satisfaction.

For an example of the AFX\_DATA, AFX\_DATA\_INIT, and AFX\_DATA\_MAP comments used by ClassWizard to maintain information for data exchange and validation, see the PENDING.CPP (and PENDING.H) module of the SCRIBBLE example program discussed in the *Class Library User's Guide*.

Once you have placed the necessary ClassWizard comments in your source files, load the project's resource script file into App Studio, invoke ClassWizard, and build the ClassWizard (.CLW) file according to the instructions in the previous section.

To learn how resource script files change when you upgrade from the previous version of the Microsoft Foundation Class Library, see Chapter 2, "Working with Files and Symbols."

P A R T 2

# App Studio Reference

Chapter 10	App Studio Quick Reference .....	137
Chapter 11	Property Page Reference .....	159



## CHAPTER 10




# App Studio Quick Reference

The first part of this chapter lists common App Studio tasks and shows you which menu items, toolbar buttons, and keystrokes you use to perform them. The second part describes all of App Studio's menus, toolbars and palettes, and accelerator keys.

## Task Reference

This section provides a quick guide to many of the common tasks you perform with App Studio's menus, accelerator keys, and toolbar buttons. Each brief task description includes a page reference to the *App Studio User's Guide* so that you can locate the passage that describes the task in detail.



## Managing Files

Task	Menu, Command	Key	Button	Comments	Page
Create a new resource script, resource file, bitmap, icon, or cursor	File, New...	CTRL+N		Opens the New dialog box, which prompts you to select the file type.	23
Create a new resource script				The resource browser window for the new resource script appears.	23
Open a resource script, resource, .EXE, .DLL, .DRV, bitmap, icon, or cursor file	File, Open...	CTRL+O		Opens the Open dialog box, which you use to navigate directories and choose a file.	24
Close the active resource script or resource file	File, Close			Or double-click the file's Control-menu box.	—
Save the active resource script or resource file	File, Save	CTRL+S		Saves the .RC file and any resource files that have changed.	22

**Managing Files** (*continued*)

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Rename the active resource script or resource file	<u>F</u> ile, Save <u>A</u> s...	F12		Opens the Save As dialog box, which prompts you for a filename.	22
Save all open resource scripts and resource files	<u>F</u> ile, Save <u>A</u> ll			Saves all resource scripts and resource files currently open.	—
Set names of included header and resource files	<u>F</u> ile, Set <u>I</u> ncludes...			Opens the Set Includes dialog box, where you enter the names of header files and compile-time resource scripts.	28
Exit App Studio	<u>F</u> ile, <u>E</u> xit	ALT+F4		Exits App Studio and prompts you to save any modified files.	—

**Editing**

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Undo an action	<u>E</u> dit, <u>U</u> ndo	CTRL+Z		By default, 10 operations can be undone. To change the number, see Appendix A.	8
Redo the last undo	<u>E</u> dit, <u>R</u> edo	CTRL+A		Disabled until an edit action has been undone.	8
Delete to the Clipboard	<u>E</u> dit, <u>C</u> u <sup>t</sup>	CTRL+X		Deletes the selection and overwrites the current Clipboard contents.	46
Copy to the Clipboard	<u>E</u> dit, <u>C</u> opy	CTRL+C		Overwrites the current Clipboard contents with the selection.	46
Paste from the Clipboard	<u>E</u> dit, <u>P</u> aste	CTRL+V		Does not overwrite selection in destination.	46
Delete selection	<u>E</u> dit, <u>D</u> elete	DEL		Removes the current selection; does not save to the Clipboard.	46



**Handling Resources**

This section lists general resource-handling tasks first, then tasks specific to particular resource types.

## General

Task	Menu, Command	Key	Button	Comments	Page
Create a new resource script, resource file, bitmap, icon, or cursor	<u>F</u> ile, <u>N</u> ew...	CTRL+N		Opens the New dialog box, which prompts you to select the file type.	23
Create a new resource	<u>R</u> esource, <u>N</u> ew...	CTRL+R		Opens the New Resource dialog box. Select a resource type and press ENTER.	6
Open an existing resource	<u>R</u> esource, <u>O</u> pen	ENTER		Enabled only when a resource is selected in the resource browser window.	7
Open a resource for binary editing	<u>R</u> esource, O <u>pen Binary</u> <u>D</u> ata	CTRL+B		Opens the binary editor window. Enabled only when a resource is selected in the resource browser window.	111
Change a resource's properties	<u>R</u> esource, <u>P</u> roperties...	ALT+ ENTER		Activates the Properties window.	10
Import a bitmap, cursor, or icon into your project	<u>R</u> esource, <u>I</u> mport...			Opens the Import Resource dialog box, where you enter the name of the file containing the bitmap, cursor, or icon resource.	28
Export a bitmap, cursor, or icon and save it in a separate file	<u>R</u> esource, <u>E</u> xport...			Opens the Export Resource dialog box, where you enter the name of the file in which to save the resource. Enabled only when a bitmap, cursor, or icon resource is selected or contained in the active editor window.	28

## Dialog Boxes — General





Task	Menu, Command	Key	Button	Comments	Page
Test a dialog box	<u>R</u> esource, <u>T</u> est	CTRL+T		Activates the dialog box for testing its controls. Menu command appears only when a dialog editor window is active.	59
Show or hide layout grid for current dialog		CTRL+G		Turns grid on or off for current dialog editor only.	51







**Dialog Boxes — General** *(continued)*

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Set order of TAB key movement among controls	<u>L</u> ayout, Set Tab <u>O</u> rders	CTRL+D		Default is order that controls were added to dialog box.	48
Show or hide the control palette	<u>W</u> indow, Show <u>C</u> ontrol Palette	F2		Appears as Hide Control Palette if palette is shown.	43




**Dialog Boxes — Alignment**

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Align controls along left edge	<u>L</u> ayout, <u>A</u> lign Controls, <u>L</u> eft	CTRL+ LEFT ARROW		Selected controls are aligned with edge of the dominant control, which is marked by filled sizing handles.	52
Align controls along right edge	<u>L</u> ayout, <u>A</u> lign Controls, <u>R</u> ight	CTRL+ RIGHT ARROW		Selected controls are aligned with edge of the dominant control, which is marked by filled sizing handles.	52
Align controls along top edge	<u>L</u> ayout, <u>A</u> lign Controls, <u>T</u> op	CTRL+ UP ARROW		Controls are aligned with edge of the dominant control, which is marked by filled sizing handles.	52
Align controls along bottom edge	<u>L</u> ayout, <u>A</u> lign Controls, <u>B</u> ottom	CTRL+ DOWN ARROW		Controls are aligned with edge of the dominant control, which is marked by filled sizing handles.	52
Align controls on vertical centerline	<u>L</u> ayout, <u>A</u> lign Controls, <u>V</u> ert. Center	F9		Controls are aligned on centerline of the dominant control, which is marked by filled sizing handles.	53
Align controls on vertical centerline	<u>L</u> ayout, <u>A</u> lign Controls, <u>H</u> oriz. Center	SHIFT+ F9		Controls are aligned on centerline of the dominant control, which is marked by filled sizing handles.	53


## Dialog Boxes — Spacing and Positioning

Task	Menu, Command	Key	Button	Comments	Page
Equalize horizontal spacing of three or more selected controls	<u>L</u> ayout, <u>S</u> pace Evenly, <u>A</u> cross	ALT+ RIGHT ARROW		Spacing between selected controls is made the same.	53
Equalize vertical spacing of three or more selected controls	<u>L</u> ayout, <u>S</u> pace Evenly, <u>D</u> own	ALT+ DOWN ARROW		Spacing between selected controls is made the same.	53
Center controls vertically in dialog box	<u>L</u> ayout, <u>C</u> enter in Dialog, <u>V</u> ertical	CTRL+F9		Horizontal position is not changed.	53
Center controls horizontally in dialog box	<u>L</u> ayout, <u>C</u> enter in Dialog, <u>H</u> orizontal	CTRL+ SHIFT+ F9		Vertical position is not changed.	53
Position selected buttons in upper-right corner of dialog box	<u>L</u> ayout, <u>A</u> rrange Buttons, <u>R</u> ight	CTRL+B		Buttons are positioned with standard spacing.	54
Position selected buttons at bottom of dialog box	<u>L</u> ayout, <u>A</u> rrange Buttons, <u>B</u> ottom	CTRL+ SHIFT+B		Buttons are positioned with standard spacing.	54

## Dialog Boxes — Sizing

Task	Menu, Command	Key	Button	Comments	Page
Make controls same size in vertical dimension	<u>L</u> ayout, <u>M</u> ake Same Size, <u>H</u> eight	CTRL+ BACK- SLASH		Other selected controls are sized to the dominant control, which is the last control selected.	54
Make controls same size in horizontal dimension	<u>L</u> ayout, <u>M</u> ake Same Size, <u>W</u> idth	CTRL+ MINUS		Other selected controls are sized to the dominant control, which is the last control selected.	54
Make controls same size in both dimensions	<u>L</u> ayout, <u>M</u> ake Same Size, <u>B</u> oth	CTRL+ EQUAL		Other selected controls are sized to the dominant control, which is the last control selected.	54



Dialog Boxes — Sizing *(continued)*

Task	Menu, Command	Key	Button	Comments	Page
Size control to fit text caption	<u>L</u> ayout, <u>S</u> ize to Content	F7		Does not apply to group boxes.	47
Set dimensions of layout grid for all dialogs	<u>L</u> ayout, <u>G</u> rid Settings...			Opens the Grid Settings dialog box.	52
Show or hide layout grid for current dialog		CTRL+G		Turns grid on or off for current dialog editor only.	51
Set order of TAB key movement among controls	<u>L</u> ayout, Set Tab <u>O</u> rder	CTRL+D		Default is order in which controls were added to dialog box.	48
Show or hide the control palette	<u>W</u> indow, Show <u>C</u> ontrol Palette	F2		Appears as Hide <u>C</u> ontrol Palette if palette is shown.	43


## Menus

Task	Menu, Command	Key	Button	Comments	Page
View a menu resource as a drop-down menu	<u>R</u> esource, <u>V</u> iew as Popup	CTRL+U		Displays the menu bar as a drop-down menu. Appears only when a menu editor window is active. Command is checked when in effect.	66


## Bitmaps, Icons, and Cursors — General

Task	Menu, Command	Key	Button	Comments	Page
Create new icon or cursor images	<u>R</u> esource, New <u>D</u> evice Image...	INS		Opens New Icon Image or New Cursor Image dialog box. Enabled only if resource does not include images for all known devices.	101
View and edit an icon or cursor image for a device	<u>R</u> esource, Open <u>D</u> evice Image			Opens the Open Icon Image or Open Cursor Image dialog box. Can also use the Device drop-down list on the editor toolbar.	102
Delete an icon or cursor image for a device	<u>R</u> esource, Delete <u>D</u> evice Image	DEL		Deletes the current device image.	102
Toggle between actual-size and magnified views	<u>I</u> mage, <u>Z</u> oom In/Out	M		Click zoom tool to select magnification factor.	98



**Bitmaps, Icons, and Cursors — General** *(continued)*

Task	Menu, Command	Key	Button	Comments	Page
Set a cursor's hotspot	<u>I</u> mage, Set <u>H</u> otspot			Shown only when a cursor is being edited.	104
Flip the selection along its vertical axis	<u>I</u> mage, Flip <u>V</u> ertical	Y			93
Flip the selection along its horizontal axis	<u>I</u> mage, Flip <u>H</u> orizontal	X			93
Toggle display of pixel grid	<u>I</u> mage, <u>G</u> rid Settings...	G			100
Toggle display of tile grid	<u>I</u> mage, <u>G</u> rid Settings...	CTRL+G		Tile grid appears only when pixel grid is also shown.	106
Show and hide pixel and tile grids; change grid dimensions	<u>I</u> mage, <u>G</u> rid Settings...			Opens the Grid Settings dialog box.	105
Show or hide the graphics palette	<u>W</u> indow, Show <u>G</u> raphics Palette	F2		Appears as Hide <u>G</u> raphics Palette if palette is shown.	86

**Bitmaps, Icons, and Cursors — Colors**

Task	Menu, Command	Key	Button	Comments	Page
Edit the current foreground color	<u>I</u> mage, <u>E</u> dit Foreground Color...			Opens the Color dialog box.	108
Pick up color from a bitmap, icon, or cursor	<u>I</u> mage, <u>P</u> ickup Color	COMMA		Preceding tool is selected after color pickup.	89
Load colors from a file	<u>I</u> mage, <u>G</u> et Palette Colors...			Opens the Get Palette Colors dialog box, which you use to navigate directories and choose a file.	110
Save colors to a file	<u>I</u> mage, <u>S</u> ave Palette Colors...			Opens the Save Palette Colors dialog box, which you use to navigate directories and choose a file.	110
Toggle opacity of the background color	<u>I</u> mage, <u>D</u> raw <u>O</u> paque	O		Works when moving and copying and when using custom brush.	92

**Bitmaps, Icons, and Cursors — Colors** *(continued)*

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Make background color opaque	<u>I</u> mage, Draw <u>O</u> paque	O		Works when moving and copying and when using custom brush.	92
Make background color transparent	<u>I</u> mage, Draw <u>O</u> paque	O		Works when moving and copying and when using custom brush.	92
Invert colors in the selection	<u>I</u> mage, <u>I</u> nv <u>e</u> rt Colors				93


**String Tables**

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Add a string to a string table	R <u>e</u> sour <u>c</u> e, N <u>e</u> w, S <u>t</u> ring	INS			78
Search for text	<u>E</u> dit, <u>F</u> ind String...	ALT+F3		Opens the Find String dialog box. Appears only when a string editor window is active.	77
Search for next occurrence of text	<u>E</u> dit, Find <u>N</u> ext String	F3		Appears only when a string editor window is active.	—


**Accelerator Tables**

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Add an accelerator to an accelerator table	R <u>e</u> sour <u>c</u> e, N <u>e</u> w, <u>A</u> ccelerator	INS			71

**ClassWizard**

<b>Task</b>	<b>Menu, Command</b>	<b>Key</b>	<b>Button</b>	<b>Comments</b>	<b>Page</b>
Connect user-interface objects to code	R <u>e</u> sour <u>c</u> e, Class <u>W</u> izard	CTRL+W		Invokes ClassWizard. Enabled for Microsoft Foundation Class Library resource scripts only.	119

## Managing Symbols

Task	Menu, Command	Key	Button	Comments	Page
View and change symbol names and values	<u>E</u> dit, S <u>ymbols</u> ...	CTRL+I		Opens the Symbol Browser dialog box.	34
Set names of included header and resource files	<u>F</u> ile, S <u>et Includes</u> ...			Opens the Set Includes dialog box, where you enter the names of header files and compile-time resource scripts.	29

## Installing VBX Controls

Task	Menu, Command	Key	Button	Comments	Page
Install and remove VBX controls	<u>F</u> ile, I <u>nstall</u> C <u>ontrols</u> ...			Opens the Install Controls dialog box, where you enter the name of the file containing the control.	55


## Managing App Studio

Task	Menu, Command	Key	Button	Comments	Page
Arrange windows in overlapping pattern	<u>W</u> indow, C <u>ascade</u>				—
Arrange windows one above the other	<u>W</u> indow, T <u>ile Horizontal</u>				—
Arrange windows side-by-side	<u>W</u> indow, T <u>ile Vertical</u>				—
Arrange icons along bottom of App Studio window	<u>W</u> indow, <u>A</u> rrange Icons				—
Show or hide the toolbar	<u>W</u> indow, T <u>oolbar</u>			Checked if toolbar is shown.	—
Show or hide the status bar	<u>W</u> indow, S <u>tatus Bar</u>			Checked if status bar is shown.	—
Show or hide the graphics editor graphics palette	<u>W</u> indow, S <u>how Graphics</u> P <u>alette</u>	F2		Appears as Hide <u>G</u> raphics Palette if palette is shown.	86
Show or hide the dialog editor control palette	<u>W</u> indow, S <u>how Control</u> P <u>alette</u>	F2		Appears as Hide <u>C</u> ontrol Palette if palette is shown.	43

## Managing App Studio *(continued)*

Task	Menu, Command	Key	Button	Comments	Page
Show or hide the Properties window	<u>W</u> indow, S <u>h</u> ow P <u>r</u> op <u>e</u> r <u>t</u> i <u>e</u> s	SHIFT+F2		Appears as Hide <u>P</u> roperties if Properties window is shown.	10
Exit App Studio	<u>F</u> ile, <u>E</u> xit	ALT+F4		Exits App Studio and prompts you to save any modified files.	—

## Getting Online Help

Task	Menu, Command	Key	Button	Comments	Page
Open the help system's Contents screen	<u>H</u> elp, <u>C</u> ontents	F1		Opens App Studio Help at the top-level Contents screen.	—
Search for a keyword in Help	<u>H</u> elp, S <u>e</u> arch...	SHIFT+F1		Opens the Search dialog box.	—
Get help on a part of the App Studio user interface				Click the button, then choose a menu or click the object you want information about.	—
Get information on product support	<u>H</u> elp, P <u>r</u> od <u>u</u> ct S <u>u</u> pp <u>o</u> rt			Explains kinds of support available from Microsoft Product Support.	—
Get information about your copy of App Studio	<u>H</u> elp, <u>A</u> bout App S <u>t</u> udio			Displays a message box that shows software version, registered owner, and serial number.	—

## Menu Reference

This section documents all the commands on App Studio's menus. Menus are described in the order of their appearance on the App Studio menu bar. Some menus appear only under certain circumstances. The Image menu, for example, appears only when an image editor window is active.

## Managing Files

<u>F</u> ile		Use To
<u>N</u> ew...	CTRL+N	Create a new resource script, resource file, bitmap, icon, or cursor
<u>O</u> pen...	CTRL+O	Open a resource script, resource, .EXE, .DLL, .DRV, bitmap, icon, or cursor file

**Managing Files** (*continued*)

<b>File</b>		<b>Use To</b>
<u>C</u> lose		Close the active resource script or resource file
<u>S</u> ave	CTRL+S	Save the active resource script or resource file
Save <u>A</u> s...		Rename the active resource script or resource file
Save <u>A</u> ll		Save all open resource scripts and resource files
Set <u>I</u> ncludes...		Set names of included header and resource files
Install <u>C</u> ontrols...		Install VBX controls
<u>E</u> xit		Exit App Studio

**Editing**

<b>Edit</b>		<b>Use To</b>
<u>U</u> ndo	CTRL+Z	Undo an action
<u>R</u> edo	CTRL+A	Redo the last undo
<u>C</u> ut	CTRL+X	Delete to the Clipboard
<u>C</u> opy	CTRL+C	Copy to the Clipboard
<u>P</u> aste	CTRL+V	Paste from the Clipboard
<u>D</u> elete	DEL	Delete selection
<u>S</u> ymbols...	CTRL+I	View and change symbol names and values
<u>F</u> ind String...	ALT+F3	Search for text
Find <u>N</u> ext String...	F3	Search for next occurrence of text

**Creating and Editing Resources**

<b>Resource</b>		<b>Use To</b>
<u>N</u> ew...	CTRL+R	Create a new resource
<u>O</u> pen		Open an existing resource
Open Binary <u>D</u> ata	CTRL+B	Open a resource for binary editing
<u>P</u> roperties...	ALT+ ENTER	Change a resource's properties
<u>I</u> mport...		Import a bitmap, cursor, or icon into your project
<u>E</u> xport		Export a bitmap, cursor, or icon and save it in a separate file
Class <u>W</u> izard	CTRL+W	Connect user-interface objects to code
<u>T</u> est	CTRL+T	Test a dialog box
<u>V</u> iew as Popup	CTRL+U	View a menu resource as a pop-up menu
New <u>D</u> evice Image...	INS	Create new icon or cursor images



### Creating and Editing Resources *(continued)*

<u>Resource</u>		<u>Use To</u>
Open Device Image...		View and edit an icon or cursor image for a device
Delete Device Image		Delete an icon or cursor image for a device
New <u>S</u> tring	INS	Add a string to a string table
New <u>A</u> ccelerator	INS	Add an accelerator to an accelerator table

## Laying Out Dialog Boxes

The Layout menu appears only when the dialog editor window is active.

<u>Layout</u>		<u>Use To</u>
<u>A</u> lign Controls		Align controls along one edge or center them on an axis
<u>S</u> pace Evenly		Equalize spacing between three or more aligned controls
<u>C</u> enter in Dialog		Center controls in a dialog box
Arrange <u>B</u> uttons		Align buttons at right edge or bottom of dialog box
<u>M</u> ake Same Size		Equalize dimensions of selected controls
<u>S</u> ize to Content	F7	Size a control to fit its contents
<u>G</u> rid Settings...		Set dimensions of layout grid
Set Tab <u>O</u> rder	CTRL+D	Set order of TAB key movement among controls

## Working with Images

The Image menu appears only when an image editor window (for editing bitmaps, icons, or cursors) is active.

<u>Image</u>		<u>Use To</u>
<u>Z</u> oom In/Out	M	Toggle between actual-size and magnified views
<u>P</u> ickup Color	COMMA	Pick up color from a bitmap, icon, or cursor
Set <u>H</u> otspot		Set a cursor's hotspot
<u>I</u> nvert Colors		Invert colors in the selection
Flip <u>V</u> ertical	Y	Flip the selection along its vertical axis
Flip <u>H</u> orizontal	X	Flip the selection along its horizontal axis
<u>E</u> dit Foreground Color...		Edit the current foreground color
<u>G</u> et Palette Colors...		Load colors from a file
<u>S</u> ave Palette Colors...		Save colors to a file

**Working with Images** *(continued)*

<u>I</u> mage		Use To
Draw <u>O</u> paque	O	Make background color opaque
<u>G</u> rid Settings...		Show and hide pixel and tile grids; change grid dimensions

**Managing App Studio**

<u>W</u> indow		Use To
Cascade		Arrange windows in overlapping pattern
<u>T</u> ile Horizontal		Arrange windows one above the other
Tile <u>V</u> ertical		Arrange windows side-by-side
<u>A</u> rrange Icons		Arrange icons along bottom of App Studio window
<u>T</u> oolbar		Show or hide the toolbar
<u>S</u> tatus Bar		Show or hide the status bar
Hide <u>G</u> raphics Palette	F2	Hide the graphics palette
Show <u>G</u> raphics Palette	F2	Show the graphics palette
Hide <u>C</u> ontrol Palette	F2	Hide the control palette
Show <u>C</u> ontrol Palette	F2	Show the control palette
Show <u>P</u> roperties	SHIFT+F2	Show or hide the Properties window

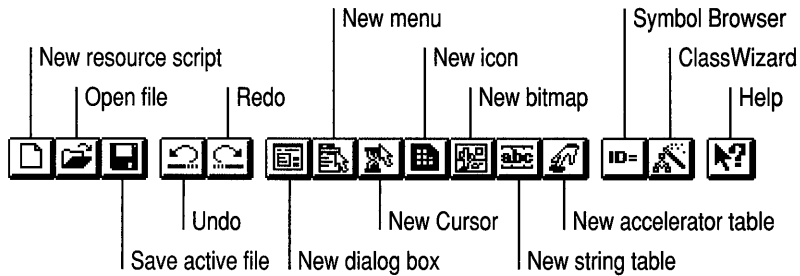
**Getting Online Help**

<u>H</u> elp		Use To
<u>C</u> ontents	F1	Open the Help Contents screen
<u>S</u> earch...		Search for a keyword in Help
<u>P</u> roduct Support		Get information on product support
<u>A</u> bout App Studio...		Get information about your copy of App Studio

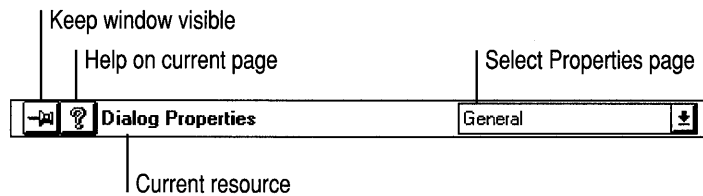
**Toolbar and Palette Reference**

This section describes the buttons and other controls on App Studio's toolbars, control bars, and palettes.

## The App Studio Toolbar

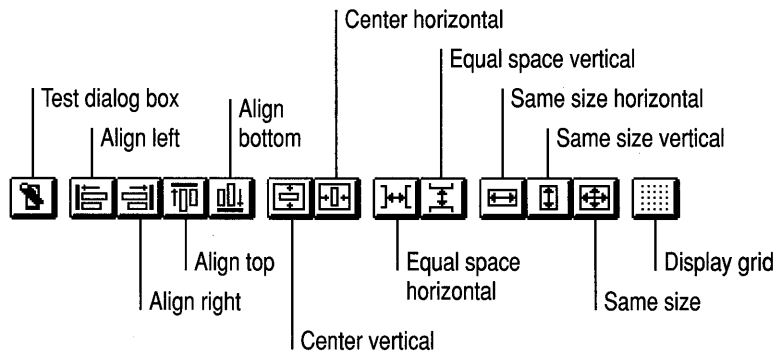


## The Properties Window Toolbar



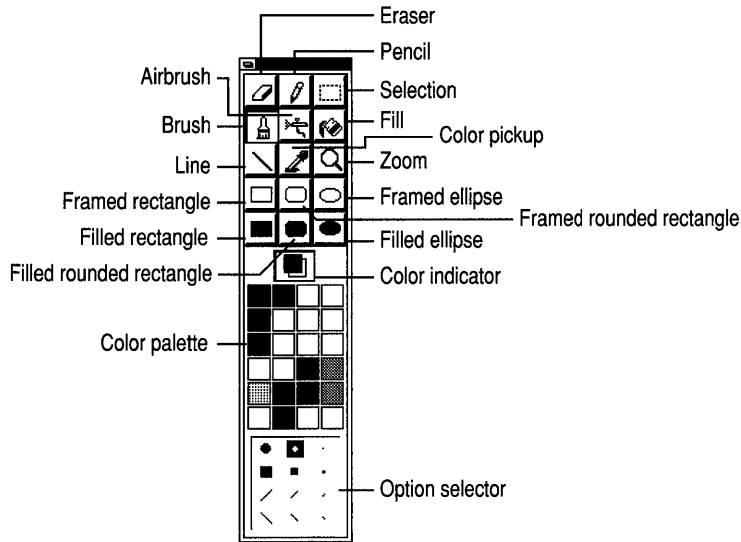
## The Dialog Editor Toolbar

This toolbar appears at the top of a dialog editor window.

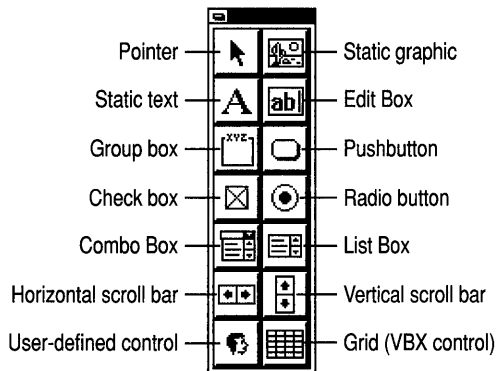


## The Graphics Editor Palette

This palette appears when an image editor window is active.



## The Control Palette



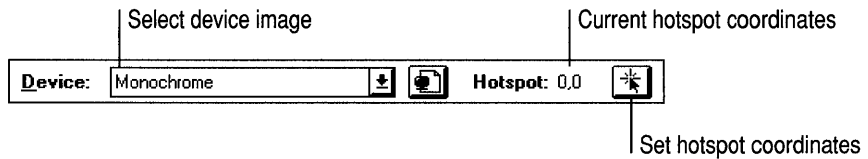
## The Icon Editing Toolbar

This toolbar appears at the top of a image editor window that contains an icon.



## The Cursor Editing Toolbar

This toolbar appears at the top of a image editor window that contains a cursor.



## Key Reference

This section lists accelerator keys for common tasks.

### Managing Files

To	Press
Create a new file	CTRL+N
Create a new resource script	F11
Open a file	CTRL+O
Save the current resource script	CTRL+S, SHIFT+F12
Save the resource script under a different name or as a different type	F12

### Editing

To	Press
Undo an action	CTRL+Z
Redo the last undo	CTRL+A
Delete to the Clipboard	CTRL+X, SHIFT+DEL
Copy to the Clipboard	CTRL+C, CTRL+INS
Paste from the Clipboard	CTRL+V, SHIFT+INS
Delete	DEL
View and change symbols	CTRL+I

## Creating and Editing Resources

To	Press
Create a new resource	CTRL+R
Create a new resource in resource browser window, string editor, accelerator editor, Symbol Browser	ALT+N
Create a resource of type currently highlighted in the resource browser window	INS
Create a new dialog box	CTRL+1
Create a new menu	CTRL+2
Create a new cursor	CTRL+3
Create a new icon	CTRL+4
Create a new bitmap	CTRL+5
Create a new string table	CTRL+6
Create a new accelerator table	CTRL+7
Delete a resource in resource browser window, string editor, accelerator editor, Symbol Browser	ALT+D
Open an existing resource selected in the resource browser window	ENTER, ALT+P
Open a resource for binary editing	CTRL+B
Activate a resource's Properties window	ALT+ENTER, ENTER
Connect a user-interface objects to code (invoke ClassWizard)	CTRL+W
Test a dialog box	CTRL+T
Open the Properties window and select Styles property page	CTRL+Y
Open the Properties window and select General property page	CTRL+E
Open the Properties window and select ID control	CTRL+Q

## Managing Windows

To	Press
Hide or show the graphics palette or control palette	F2
Hide or show the Properties window	SHIFT+F2
Activate next window	CTRL+F6, CTRL+TAB
Activate previous window	CTRL+SHIFT+F6, CTRL+SHIFT+TAB
Close current window	CTRL+F4
Activate the Properties window	ALT+ENTER, ALT+F6, ENTER (except in resource browser window)
Activate resource browser window	CTRL+F2
Scroll vertically	PAGE UP, PAGE DOWN
Scroll horizontally	CTRL+PAGE UP, CTRL+PAGE DOWN
Move resource browser window focus to the Resources box	RIGHT ARROW
Move resource browser window focus to the Type box	LEFT ARROW
Shift resource browser window focus between Resources and Type boxes	TAB, SHIFT+TAB

## Using the Properties Window

To	Press
Activate Properties window	ALT+ENTER
Select next page	PAGE DOWN
Select previous page	PAGE UP
Move to next field	TAB
Move to previous field	SHIFT+TAB
Hide or show the Properties window	SHIFT+F2
Open the Properties window and select Styles property page	CTRL+Y
Open the Properties window and select General property page	CTRL+E
Open the Properties window and select ID control	CTRL+Q
Accept changes	ENTER
Cancel change	ESC

## Editing Graphics

To	Press
Increase the magnification factor	> (SHIFT+PERIOD)
Decrease the magnification factor	< (SHIFT+COMMA)
Use framed-rectangle tool	R
Use filled-rectangle tool	SHIFT+R
Use framed rounded-rectangle tool	N
Use filled rounded-rectangle tool	SHIFT+N
Use framed-ellipse tool	E
Use filled-ellipse tool	SHIFT+E
Use pencil tool	P
Use eraser tool	SHIFT+P
Use brush tool	D
Use selection tool	S
Use fill tool	F
Use line tool	L
Use airbrush tool	A
Pick up color	COMMA
Increase brush size	PLUS, EQUAL SIGN
Decrease brush size	MINUS
Use single-pixel brush	PERIOD
Flip along horizontal axis	X
Flip along vertical axis	Y
Toggle background-color opacity	O
Outline custom brush with foreground color	SHIFT+O
Toggle pixel grid	G
Zoom under cursor	M
Use zoom tool	Z
Use custom brush	CTRL+B
Halve size of custom brush	MINUS
Double size of custom brush	PLUS
New device image for icons and cursors	INS
Move to next pane	TAB, F6
Move to previous pane	SHIFT+TAB, SHIFT+F6
Select previous foreground color	[



**Editing Graphics** (*continued*)

To	Press
Select next foreground color	]
Select previous background color	{
Select next background color	}
Show or hide pixel grid	G
Show or hide tile grid	CTRL+G

**Using the Dialog Editor**

To	Press
Move to the next control	TAB
Move to the previous control	SHIFT+TAB
Move control right one DLU	RIGHT ARROW
Move control left one DLU	LEFT ARROW
Move control up one DLU	UP ARROW
Move control down DLU	DOWN ARROW
Expand the selected control horizontally	SHIFT+RIGHT ARROW
Contract the selected control horizontally	SHIFT+LEFT ARROW
Expand the selected control vertically	SHIFT+DOWN ARROW
Contract the selected control vertically	SHIFT+UP ARROW
Align selected controls along left edge	CTRL+LEFT ARROW
Align selected controls along right edge	CTRL+RIGHT ARROW
Align selected controls along top edge	CTRL+UP ARROW
Align selected controls along bottom edge	CTRL+DOWN ARROW
Align selected controls horizontally on centerlines	F9
Align selected controls vertically on centerlines	SHIFT+F9
Center selected control vertically in dialog box	CTRL+F9
Center selected control horizontally in dialog box	CTRL+SHIFT+F9
Space selected controls evenly across	ALT+RIGHT ARROW
Space selected controls evenly down	ALT+DOWN ARROW
Align buttons at right edge of dialog box	CTRL+B
Align buttons at bottom of dialog box	CTRL+SHIFT+B

**Using the Dialog Editor** *(continued)*

To	Press
Make selected controls same width	CTRL+MINUS
Make selected controls same height	CTRL+BACKSLASH
Make selected controls same height and width	CTRL+EQUAL
Size control to content	F7
Show or hide alignment grid	CTRL+G
Set tab order	CTRL+D

**Using the Menu Editor**

To	Press
View menu as pop-up	CTRL+U
Insert empty menu at current location	INS
Delete menu item	DEL

**Using the String Editor**

To	Press
Find text	ALT+F3, CTRL+F
Find next occurrence of text	F3
Add new string	INS
Delete current string	DEL

**Editing Resources as Binary Data**

To	Press
Move selection right	RIGHT ARROW
Move selection left	LEFT ARROW
Move selection up	UP ARROW
Move selection down	DOWN ARROW
Move between hexadecimal and ASCII listings	TAB

**Getting Online Help**

To	Press
Open the Help Contents screen	F1



# Property Page Reference

This chapter provides detailed information about each of the App Studio property pages. Property pages are displayed in the App Studio Properties window and are used to control the appearance and behavior of Windows resources. For more information about the App Studio Properties window and how to use it, see Chapter 1, page 9.

## Resource Property Page

The following properties are available in the Properties window when a resource is selected in the resource browser window but the editing window for that resource is not open, or when the editing window is open but minimized:

### ID

The resource's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. (For string tables, the ID is always **STRINGTABLE** and cannot be modified.)  
Type: Symbol, Integer, or Quoted String.

### Preload

Determines if the resource is loaded when the application starts up (True) or is not loaded until it is called (False). Equivalent to the **PRELOAD** and **LOADONCALL** resource compiler statements. Type: Bool. Default: False.

### Moveable

Permits the resource to be moved to compact memory if necessary (True), or causes it to remain at a fixed memory location. Equivalent to the **MOVABLE** and **FIXED** resource compiler statements. Type: Bool. Default: True.

### Discardable

Allows the resource to be purged from memory if no longer needed. When the resource is needed again, it is loaded from disk. Type: Bool. Default: True.

**Pure**

Prevents the resource from being dynamically modified by the application at run time if set to True. If Pure is False, your application cannot have more than one instance running at a time. Has no effect for standard resource types in Microsoft Foundation Class Library applications. Made available for user resource types and backward compatibility with real-mode applications. Type: Bool. Default: True.

Bitmaps, icons, and cursors also have the following fields in the Resource property page:

**Filename**

The name of the file containing the bitmap, icon, or cursor resource.

**Preview**

A box showing what the bitmap, icon, or cursor looks like. Useful for browsing through graphics resources without opening them.

## Accel Table: Accel Properties — General

**ID**

The resource's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol or Integer.

**Key**

The accelerator key. Can be one of the following (see Chapter 5, page 73, for more information):

- Integer—Range 0 to 255. Interpreted as ASCII or virtual-key value, depending on the Type property. Single digits are interpreted as a key value. To enter an ASCII value from 0 to 9, precede the number with two zeros (for example, 006).
- Character—Single character optionally preceded by ^ to signify a control character.
- Virtual key identifier—Any one of the virtual-key identifiers in the drop-down list.

**Modifiers**

Indicate whether the accelerator is a combination formed with Ctrl, Alt, or Shift. When the key is an ASCII key, Ctrl and Shift are not available. Type: Bool. Defaults: Ctrl is True, Alt and Shift are False.

**Type**

Specifies whether the Key property is an ASCII value or a virtual key (VirtKey) value.

### Next Key Typed

When you choose this command, the next key combination typed changes the Key and Modifiers values appropriately. The key is always interpreted as a virtual key if possible.

## Bitmap Properties — General

### ID

The resource's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol, Integer, or Quoted String.

### Filename

The name of the file containing the bitmap resources.

### Width

Image width in pixels. Type: Integer. Default: 48.

### Height

Image height in pixels. Type: Integer. Default: 48.

### Colors

Monochrome (2) or 16. The number of colors in a bitmap is determined by the current display device.

### Save Compressed

Saves the image in compressed format to save space; useful for large bitmaps. Only color bitmaps can be compressed. Type: Bool. Default: False.

## Cursor Properties — General

### ID

The resource's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol, Integer, or Quoted String.

### Filename

The name of the file containing the cursor resource.

The following properties are displayed only and cannot be modified on the property page:

### Width

Image width in pixels. This property is determined by the target-device definition selected by the user.

### Height

Image height in pixels. This property is determined by the target-device definition selected by the user.

**Hotspot**

Location of the cursor's active area. In pixels, relative to the upper-left corner (0,0). Set with the Set Hotspot command on the Image menu.

## Dialog Properties — General

**ID**

The resource or resource object's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol, Integer, or Quoted String.

**Caption**

The text that appears as part of the dialog box to label it. Default: A name based on the type of resource (in this case, "Dialog") plus a number based on the resource identifier assigned by App Studio.

**Font Name**

The typeface of the font that will be used in all the controls in the dialog box. The bold version of the typeface is always used. Change this value by choosing the Font command in the lower-left corner of the property page. Default: MS Sans Serif.

**Font Size**

The point size of the font that will be used in all the controls in the dialog box. Default: 8 points. Change this value by choosing the Font command in the lower-left corner of the property page.

**Font**

Choose the Font command to change the typeface or size of the dialog-box font.

**Menu**

Contains the resource identifier of the menu used in the dialog box, if any. Type: Resource identifier. Default: None.

**X Pos**

The x-coordinate, in dialog box units (DLUs), of the upper-left corner of the dialog box. Type: Integer.

**Y Pos**

The y-coordinate, in DLUs, of the upper-left corner of the dialog box. Type: Integer.

**Registered Class**

Identifier of a registered dialog class (a Windows operating system window class, not to be confused with a C++ class). Provided to support C programming. If you are using a resource file with Microsoft Foundation Class Library support, this option is disabled. Type: Integer or String. String must be in double quotes. Default: None.

# Dialog Properties — Styles

## Style

One of the following:

- **Overlapped**—Creates an overlapped window. An overlapped window is always a top-level window and should have a caption and a border.
- **Popup (Default)**—Creates a pop-up window.
- **Child**—Creates a child window.

## Border

One of the following:

- **None**—No border. A title bar is not available.
- **Thin**—A thin border.
- **Resizing**—Creates a thick border that can be used to resize the dialog box.
- **Dialog Frame (Default)**—A dialog-box border.

## Minimize Box

Creates a minimize box for the dialog box. Disabled if there is no title bar.

Type: Bool. Default: False.

## Maximize Box

Creates a maximize box for the dialog box. Disabled if there is no title bar.

Type: Bool. Default: False.

## Titlebar

Creates a title bar for the dialog box. Disabled if the dialog box has no border.

Type: Bool. Default: True.

## System Menu

Creates a system menu for the dialog box. Disabled if there is no title bar. Type:

Bool. Default: True.

## Horiz. Scroll

Creates a horizontal scroll bar for the dialog box. Type: Bool. Default: False.

## Vert. Scroll

Creates a vertical scroll bar for the dialog box. Type: Bool. Default: False.

## Clip Siblings

Clips child windows relative to each other; that is, when a particular child window is repainted, this style clips all other top-level child windows out of the region of the child window to be updated. If **Clip Siblings** is **False** and child windows overlap, it is possible, when drawing in the client area of a child window, to draw in the client area of a neighboring child window. For use with child windows only. Type: Bool. Default: False.



**Clip Children**

Excludes the area occupied by child windows when drawing within the parent window. Used when creating the parent window. Do not use this style if your dialog box contains a group box. Type: Bool. Default: False.

**System Modal**

Creates a system-modal dialog box. A system-modal dialog box prohibits switching to another window or program while the dialog box is active. Type: Bool. Default: False.

**Absolute Align**

Determines whether the dialog box is aligned relative to the screen or relative to its parent window. If True, the dialog is displayed at coordinates relative to the upper-left corner of the screen. Type: Bool. Default: False.

**NoIdleMsg**

Suppresses the **WM\_ENTERIDLE** message ordinarily sent to a dialog box's owner when no more messages are waiting in its message queue. Type: Bool. Default: False.

**LocalEdit**

Specifies that edit-box controls in the dialog box will use memory in the application's data segment. Normally all edit-box controls in dialog boxes use memory outside the application's data segment. Type: Bool. Default: False.

**Visible**

Specifies that the dialog box is visible when first displayed. Set this property to False for form views and dialog-bar template resources. Type: Bool. Default: True.

**Disabled**

Creates a dialog box that is initially disabled. Type: Bool. Default: False.

## Dialog: Check Box Properties — General

**ID**

The resource's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol or Integer.

**Caption**

The text that appears as part of the control to label it. To make one of the letters in the caption of a control the mnemonic key, precede it with an ampersand (&). Default: A name based on the type of control (for example, "Check") plus a number based on the resource identifier assigned by App Studio.

**Visible**

Determines whether or not the control is visible when the application is first run. Type: Bool. Default: True.

**Disabled**

Determines if the resource is displayed as disabled when the dialog box is created. Type: Bool. Default: False.

**Group**

Specifies the first control of a group of controls in which the user can move from one control to the next by using the arrow keys. All controls in the tab order after the first control with the Group property set to False belong to the same group. The next control in the tab order with Group set to True ends the first group of controls and starts the next group. Type: Bool. Default: False.

**Tabstop**

Specifies that the user can move to this control with the TAB key. Type: Bool. Default: True.

**Auto**

Creates a check box that, when selected, automatically toggles between checked and unchecked states. You must set this property to True if you are using a group of check boxes with Dialog Data Exchange (see page 121). Type: Bool. Default: True.

**Left Text**

Positions the check box's caption text to the left instead of to the right. Type: Bool. Default: False.

**Tri-State**

Creates a three-state check box. A three-state check box can be grayed as well as checked or not checked. A grayed check box indicates that the state represented by the control is undetermined. Type: Bool. Default: False.

## Dialog: Combo Box Properties — General

For a description of the following properties see Dialog: Check Box Properties—General on page 164:

**ID****Caption****Visible****Disabled****Group****Tabstop**

Combo boxes have an additional property:

**Enter list choices**

Contains the choices you want to appear in the combo box when the dialog is created. Press CTRL+ENTER at the end of each item in the list to move to the next line. This property is only available in resource files with Microsoft Foundation Class Library support.

## Dialog: Combo Box Properties — Styles

### Type

Specifies the type of combo box. Can have one of the following values:

- **Simple**—Creates a simple combo box that combines an edit-box control with a list control. The list is displayed at all times, and the current selection in the list is displayed in the edit-box control.
- **Dropdown (Default)**—Creates a drop-down combo box. Same as a simple combo box, except the list is not displayed unless the user clicks a drop-down arrow at the right of the edit-box control portion of the combo box.
- **Drop List**—Similar to the drop-down style, but the edit-box control is replaced by a static-text item that displays the current selection in the list.

For a description of the following styles see Dialog: Edit Box Properties — Styles (following) and Dialog: List Box Properties — Styles (page 169):

Owner Draw

Has Strings

Sort

Vert. Scroll

No Integral Height

Auto HScroll

Disable No Scroll

OEM Convert

## Dialog: Edit Box Properties — General

For a description of the following properties see Dialog: Check Box Properties — General on page 164:

ID

Visible

Disabled

Group

Tabstop

## Dialog: Edit Box Properties — Styles

### Align Text

Text aligns left, centered, or right. In a multiline edit box, text is always aligned left. Default: Left.

### Multiline

Creates a multiline edit-box control.

When the multiline edit-box control is in a dialog box, the default response to pressing the ENTER key is to activate the default button.

If AutoHScroll is selected, the multiline edit-box control automatically scrolls horizontally when the caret goes past the right edge of the control. To start a new line, the user must press ENTER. If AutoHScroll is not selected, the control automatically wraps words to the beginning of the next line when necessary. A new line is also started if the user presses ENTER, providing the Want Return property is set. The position of the wordwrap is determined by the window size. If the window size changes, the wordwrap position changes and the text is redisplayed.

Multiline edit-box controls can have scroll bars. An edit-box control with scroll bars processes its own scroll-bar messages. Edit-box controls without scroll bars scroll as described in the previous paragraph. They also process any scroll messages sent by the parent window. Type: BOOL. Default: False.

#### Horiz. Scroll

Provides a horizontal scroll bar for a multiline control. Type: Bool. Default: False.

#### Auto HScroll

Automatically scrolls text to the right when the user types a character at the end of the line. Type: Bool. Default: True.

#### Vert. Scroll

Provides a vertical scroll bar for a multiline edit-box control. Type: Bool. Default: False.

#### Auto VScroll

In a multiline control, automatically scrolls text up one line when the user presses ENTER on the last line. Type: Bool. Default: False.

#### Password

Displays all characters as an asterisk (\*) as they are typed into the edit-box control. Not available in multiline controls. Type: Bool. Default: False.

#### No Hide Sel

Changes the way text is displayed when an edit box loses and regains focus. Normally, existing text in an edit box is hidden when the control loses focus and is displayed as inverted text when the control regains focus. If NoHideSel is set to True, selected text in a edit box is displayed as selected at all times. Type: Bool. Default: False.

#### OEM Convert

Converts text entered in the edit-box control from the Windows character set to the OEM character set and then back to the Windows set. This ensures proper character conversion when the application calls the AnsiToOem function to convert a Windows string in the edit-box control to OEM characters. This style is most useful for edit-box controls that contain filenames. Type: Bool. Default: False.

**Want Return**

Specifies that a carriage return be inserted when the user presses the ENTER key while entering text into a multiline edit-box control in a dialog box. If this style is not specified, pressing the ENTER key has the same effect as pressing the dialog box's default pushbutton. This style has no effect on a single-line edit-box control. Type: Bool. Default: False

**Border**

Creates a border around the edit box. Type: Bool. Default: True.

**Uppercase**

Converts all characters to uppercase as they are typed into the edit box. Type: Bool. Default: False.

**Lowercase**

Converts all characters to lowercase as they are typed into the edit box. Type: Bool. Default: False.

**Read Only**

Prevents the user from typing or editing text in the edit box. Type: Bool. Default: False.

## Dialog: Group Box Properties — General

For a description of the following properties see Dialog: Check Box Properties — General on page 164:

ID

Caption

Visible

Disabled

Group

Tabstop (Default: False)

## Dialog: List Box Properties — General

For a description of the following properties see Dialog: Check Box Properties — General on page 164:

ID

Visible

Disabled

Group

Tabstop

# Dialog: List Box Properties — Styles

## Selection

Determines how items in a list box can be selected. Possible values are as follows:

- **Single (default)**—Only one item in a list box can be selected at a time.
- **Multiple**—More than one list-box item can be selected, but the **SHIFT** and **CTRL** keys have no effect. Clicking or double-clicking an unselected item selects it. Clicking or double-clicking a selected item deselects it.
- **Extended**—The **SHIFT** and **CTRL** keys can be used together with the mouse to select and deselect list-box items, select groups of items, and select non-adjacent items.

## Owner Draw

Controls the owner-draw characteristics of the list box. Can be one of the following values:

- **No (default)**—Turns off the owner-draw style. The list box contains strings.
- **Fixed**—Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are the same height.

**CWnd::OnMeasureItem** is called when the list box is created and **CWnd::OnDrawItem** is called when a visual aspect of the list box has changed.

- **Variable**—Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are variable in height.

**CWnd::OnMeasureItem** is called for each item in the list when the list box is created and **CWnd::OnDrawItem** is called when a visual aspect of the list box has changed.

## Has Strings

Specifies that an owner-draw list box contains items consisting of strings. The list box maintains the memory and pointers for the strings so the application can use the **LB\_GETTEXT** message to retrieve the text for a particular item. By default, all list boxes except owner-draw list boxes have this style. An application can create an owner-draw list box either with or without this style.

This style is only available if the Owner Draw property is set to either **Fixed** or **Variable**. If Owner Draw is set to **No**, the list box contains strings by default.

Type: **Bool**. Default: **True**.

## Border

Creates a list box with a border. Type: **Bool**. Default: **True**.

## Sort

Sorts the contents of the list box alphabetically. Type: **Bool**. Default: **True**.

**Notify**

Notifies the parent window if a list item has been clicked or double-clicked. Type: Bool. Default: True.

**Multicolumn**

Specifies a multicolumn list box that is scrolled horizontally. The **LB\_SETCOLUMNWIDTH** message sets the width of the columns. Type: Bool. Default: False.

**Horiz. Scroll**

Creates a list box with a horizontal scroll bar. Type: Bool. Default: False.

**Vert. Scroll**

Creates a list box with a vertical scroll bar. Type: Bool. Default: True.

**No Redraw**

Specifies that the list box's appearance is not updated when changes are made. This style can be changed at any time by sending a **WM\_SETREDRAW** message or by calling **CWnd::SetRedraw**. Type: Bool. Default: False.

**Use Tabstops**

Allows a list box to recognize and expand tab characters when drawing its strings. The default tab positions are 32 dialog box units. Type: Bool. Default: False.

**Want Key Input**

Specifies that the owner of the list box receives **WM\_VKEYTOITEM** or **WM\_CHAROITEM** messages whenever the user presses a key and the list box has the input focus. This allows an application to perform special processing on the keyboard input. If a list box uses the Has Strings style, the list box receives **WM\_VKEYTOITEM** messages. If a list box does not use the Has Strings style, it receives **WM\_CHAROITEM** messages. Type: Bool. Default: False.

**Disable No Scroll**

Shows a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll. Without this style, the scroll bar is hidden when the list box does not contain enough items. Type: Bool. Default: False.

**No Integral Height**

Specifies that the size of the list box is exactly the size specified by the application when it created the list box. Normally, Windows sizes a list box so that the list box does not display partial items. Type: Bool. Default: True.

## Dialog: Picture Properties — General

For a description of the following properties see Dialog: Check Box Properties—General on page 164:

**ID**

Visible

Disabled  
Group  
Tabstop (Default: False)

Static graphics have additional general properties:

Type

Sets the type of static graphic to display. One of the following:

- **Frame (Default)**—Displays a frame. You set the color of the frame in the Color box. Use a frame to visually group controls.
- **Icon**—Displays an icon in the dialog box. Use the Icon box to specify the identifier of the icon you want to display.
- **Rectangle**—Displays a filled-in rectangle. You set the color of the rectangle in the Color box.

Icon

Sets the resource identifier of the icon to be displayed.

Color

Sets the color of a frame or rectangle to black, white, or gray. This property is not available when the picture type is icon. Default: Black.

## Dialog: Pushbutton Properties — General

For a description of the following properties see Dialog: Check Box Properties—General on page 164:

ID  
Caption  
Visible  
Disabled  
Group  
Tabstop

Pushbuttons have additional general properties:

Default Button

If True, the control is the default button in the dialog box. The default button is selected (drawn with a heavy black border) when the dialog box first appears and is executed if the user presses ENTER without choosing another command in the dialog box. Windows allows only one default button in a dialog box. Type: Bool. Default: False.



**Owner Draw**

Creates an owner-draw button. Use an owner-draw button when you need to customize the appearance of a control by providing your own **OnDrawItem** message handler in the owner-window procedure (usually a dialog-box procedure or class derived from the Microsoft Foundation classes **CDialog** or **CFormView**). You can also derive your own class from **CButton** and override **CButton::DrawItem**. See **CWnd::OnDrawItem** and **CButton::OnDraw** in the *Class Library Reference* for more information.

This property must be set to define a bitmap button in a dialog box using the **CBitmapButton** class. Type: Bool. Default: False

## Dialog: Radio Button Properties — General

For a description of the following properties see the Dialog: Check Box Properties —General on page 164:

**ID****Caption****Visible****Disabled****Group****Tabstop (Default: False)**

Radio buttons also have additional general properties:

**Auto**

When the user selects a radio button with this property, the radio button is automatically selected and any other radio buttons in the same group are cleared (deselected). You must set this property to True if you are using a group of check boxes with Dialog Data Exchange (see page 121). Type: Bool. Default: True.

**Left Text**

Places the radio button's caption text on the left rather than the right. Type: Bool. Default: False.

## Dialog: Scrollbar Properties — General

For a description of the following properties see the Dialog: Check Box Properties —General on page 164:

**ID****Visible****Disabled****Group****Tabstop (Default: False)**

Scroll bars have an additional general property:

#### Align

One of the following values:

- None (Default)—No special alignment is performed. The size of the scroll bar is the size specified in App Studio.
- Top/Left—Aligns the upper-left corner of the scroll bar with the upper-left corner of the containing window specified in App Studio.
- Bottom/Right—Aligns the lower-right corner of the scroll bar with the lower-right corner of the containing window specified in App Studio.

## Dialog: Text Properties — General

For a description of the following properties see Check Box Properties—General on page 164:

ID

Caption

Visible

Disabled

Group (Default: True)

Tabstop (Default: False)

Static text controls have additional general properties:

#### No Prefix

Prevents ampersands (&) in the control's text from being interpreted as the mnemonic character. Normally a string containing an ampersand is displayed with the ampersand removed and the next character in the string underlined. The No Prefix style is most often used when filenames or other strings that may contain an ampersand need to be displayed.

#### No Wrap

Displays text left-aligned. Tabs are expanded but words are not wrapped. Text that extends past the end of a line is clipped. Type: Bool. Default: False.

#### Simple

Disables No Wrap and Text Align. Text in static text controls with this property set does not wrap and is not clipped. In addition, setting this property means that overriding **WM\_CTLCOLOR** in the parent window has no effect on the control. Type: Bool. Default: False.

#### Text Align

Controls how text is aligned in the static-text control. Possible values are Left, Center, and Right. Set to Left when No Wrap is selected. Default: Left.

## Dialog: User Control Properties — General

For a description of the following properties see Dialog: Check Box Properties—General on page 164:

ID  
Caption  
Visible  
Disabled  
Group  
Tabstop

User-defined controls have additional general properties:

Class

The name of the control's Windows class. This class must be registered before the dialog box containing the control is created.

Style

A 32-bit hexadecimal value specifying the control's style, primarily used to edit the lower 16 bits that make up a user control's sub-style.

## Dialog: VBX Control Properties — General

For a description of the following properties see Dialog: Check Box Properties—General on page 164:

ID  
Caption  
Visible  
Disabled  
Group  
Tabstop

## Dialog: VBX Control Properties — Styles

See page 57 for a description of how to use the VBX control Styles property page. For information about specific VBX control styles, see the documentation for that VBX control.

## Icon Properties — General

### ID

The resource or resource object's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol, Integer, or Quoted String.

### Filename

The name of the file containing the icon resource.

The following properties are displayed only and cannot be modified on the property page:

### Width

Image width in pixels. This property is determined by the currently selected target-device definition.

### Height

Image height in pixels. This property is determined by the currently selected target-device definition.

### Colors

Monochrome (2) or 16. This property is determined by the currently selected target-device definition.

## Menu Properties — General

### ID

The resource or resource object's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol, Integer, or Quoted String.

## Menu: Menu Item — General

### ID

The resource's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol or Integer.

### Caption

The text that appears as part of the menu item to label it. To make one of the letters in the caption of a menu item the mnemonic key, precede it with an ampersand (&).

### Separator

If True, the menu item is a separator. Type: Bool. Default: False.

### Checked

If True, the menu item is initially checked. Type: Bool. Default: False.

**Popup**

If True, the menu item is a pop-up menu (a submenu). Type: Bool. Default: True for top-level menu items in a menu bar; otherwise False.

**Grayed**

If True, the menu item is initially grayed and inactive. Type: Bool. Default: False.

**Inactive**

If the Grayed property is True, then the Inactive property is always True. Otherwise Inactive determines whether the menu item is initially inactive. Type: Bool. Default: False.

**Help**

Right justifies the menu item in the menu bar at run time. Type: Bool. Default: False.

**Break**

Can be one of these values:

- None (Default)—No break
- Column—For static menu-bar items, places the item on a new line. For pop-up menus, places the item in a new column with no dividing line between the columns. Setting this property affects the appearance of the menu only at run time, not in the menu editor.
- Bar—Same as Column except, for pop-up menus, separates the new column from the old column with a vertical line. Setting this property affects the appearance of the menu only at run time, not in the menu editor.

**Prompt**

Contains text to appear in the status bar when this menu item is highlighted. The text entered here is placed in the string table with the same identifier as the menu item. Available only in resource files with Microsoft Foundation Class Library support.

## String Editor: String Properties — General

**ID**

The resource or resource object's identifier. Usually the resource ID is a symbol supplied by App Studio and defined in the .H file that App Studio creates as part of your project. Type: Symbol or Integer.

**Caption**

A string of up to 255 bytes (single characters, escape sequences, or ASCII values). See page 79 for more information about entering escape sequences and ASCII values as part of the string. Type: Text.

# Appendix

Appendix A	APPSTUDIO.INI Settings . . . . .	179
------------	----------------------------------	-----



# APSTUDIO.INI Settings

APSTUDIO.INI, App Studio's initialization file, is a text file containing information that App Studio retains from one session to another. Most of the settings are "read-write": App Studio reads them at startup and writes them at the end of the session if they have changed. Other settings are "read-only": App Studio reads them at startup but never writes them. You edit APSTUDIO.INI by hand to create or change these settings. This appendix describes the read-only settings you can use to customize App Studio's behavior. Section labels divide APSTUDIO.INI into sections. Each section contains one or more entries consisting of an entry name, an equal sign, and a string:

```
[Section Label]
EntryName=String
AnotherEntry=AnotherString
```

Each part of this appendix shows both the section label and the entry.

## Setting the Default .RC File Type

```
[General]
UseMFC=Bool
```

This entry is equivalent to the Use Microsoft Foundation Classes check box in the New dialog box. If *Bool* is 1, the check box is checked by default; if *Bool* is 0, the check box is cleared by default. If the check box is checked, App Studio creates a resource script (.RC) file that supports Microsoft Foundation Class Library features such as ClassWizard and VBX controls. If the check box is cleared, App Studio creates a standard resource script (.RC) file.

## Setting the Number of Undo Levels

```
[General]
UndoLevels=levels
```



The number of undo operations is limited to the number specified by *levels*. If this entry does not appear in APSTUDIO.INI, the default of 10 is used.

## Using Default Dialog-Box Buttons

[Dialog Editor]  
InitialButtons=*Bool*

If *Bool* is 1, dialog-box templates are created with the OK and Cancel buttons. If *Bool* is 0, they are not.

## Setting the Default Magnification Factor

[Graphics Editor]  
DefaultZoom=*int*

Sets the default value for the ratio of magnified and actual-size views in the image editor window. The range is 2 to 10. If this entry does not appear in APSTUDIO.INI, the default of 6 is used.

## Describing Cursor Devices

[Cursor Devices]  
*name=colors[,width[,height]*

Specifies the names of display devices and the attributes of their corresponding cursor images. The name of the device is specified by *name*. It appears in the New Cursor Image dialog box when you add an image to a cursor. The *colors*, *width*, and *height* entries specify the number of colors supported by the device (subject to App Studio's limit of 16 colors) and the image size in pixels. Windows currently supports only monochrome cursors.

## Describing Icon Devices

[Icon Devices]  
*name=colors[,width[,height]*

Specifies the names of display devices and the attributes of their corresponding icon images. The name of the device is specified by *name*. It appears in the New Cursor Image dialog box when you add an image to a cursor. The *colors*, *width*, and *height* entries specify the number of colors supported by the device (subject to App Studio's limit of 16 colors) and the image size in pixels.

# Index

## A

Absolute Align (Dialog Box Styles properties) 164

Accel Table, Accel Properties 160

Accelerator keys

*See also* Accelerator table

adding 71

copying 72

deleting 72

key name in menu caption 67, 74

menu item, associating with 73–74

moving 72

properties

defining 72–73

editing 72–73

values 73

Accelerator keys, App Studio (Quick Reference)

binary editing 157

creating and editing resources 153

editing dialog boxes 156

graphics 155

menus 157

string tables 157

managing

App Studio windows 154

files 152

online help 157

Properties window 154

Accelerator table *See also* Accelerator Keys

creating new 69

defined 17, 69

editing 70–72

editor window

described 70

illustrated 12,71

empty tables deleted 70

General properties 160

opening 70

Quick reference to tasks 144

*See also* Accelerator Keys

Accelerator table properties

ID 160

Key 160

Modifiers 160

Next Key Typed 161

Accelerator table properties (*continued*)

property page reference 160

Type 160

Add Member Variable dialog box *See* ClassWizard

Adding

accelerator keys 71

class, with ClassWizard 114–116

controls 42–44

custom colors 108–109

strings in string table 77–78

VBX controls to control palette 55

Airbrush tool

changing size of 87

drawing with 86–87

Align (Scrollbar properties) 173

Align Text (Edit Box Styles properties) 166

Aligning controls

on center 53

overview 52

procedure 52–54

App Studio

basic concepts 4

binary data, editing 111

converting files to 24–25

opening resource files from 24

overview 3

Properties window 9

Quick reference to tasks 145

starting 4

App Studio toolbar

ClassWizard 119

New Accelerator Table 69

New Bitmap 84

New Cursor 84

New File 23

New Icon 84

New String Table 76

new-resource buttons 6

Quick reference 150

Redo 9

Symbol Browser 34

Undo 9

.APS file

defined 22

APSTUDIO.INI

CursorDevices section 102–103

IconDevices section 102–103

overview 179

UndoRedo section 8

APSTUDIO.INI (*continued*)

- UseMfc section 23
- VBX control information 55
- Areas, filling 88
- Arranging
  - Controls 51–54
  - pushbuttons 54
- ASCII characters, adding to a string 79
- Auto
  - Check Box properties 165
  - Radio Button properties 172
- Auto HScroll
  - Combo Box Styles properties 166
  - Edit Box Styles properties 167
- Auto VScroll, Edit Box Styles properties 167

**B**

- Background color (graphics) 86
- Backgrounds, opaque or transparent (graphics) 92–93
- Binary data
  - opening a resource for editing 111
  - overview 111
  - using the binary data editor window 112
- Bitmap properties
  - Colors 161
  - Filename 161
  - Height 161
  - ID 161
  - property page reference 161
  - Save Compressed 161
  - Width 161
- Bitmap resource, defined 18
- Bitmaps
  - See also* Graphics
  - Colors (Quick Reference) 143
  - editing 142
  - General properties 161
- Bitmaps, toolbar
  - editing 104–107
  - overview 104
- .BMP file
  - exporting 28
  - importing 28
- Border
  - Dialog Box Styles properties 163
  - Edit Box Styles properties 168
  - List Box Styles properties 169
- Break (Menu Item properties) 176
- Browser, Symbol *See* Symbol Browser
- Browsing
  - graphics, with resource browser window 84
  - resources 5–6

- Brush tool
  - changing size of 87
  - drawing with 86–87
  - single pixel 87
- Brush, custom
  - creating 93
  - using 94
- Buttons, toolbar
  - editing 106–107
  - overview 104–105

**C**

- Calculated symbols, including 29–30
- Caption
  - Check Box properties 164
  - Combo Box General properties 165
  - Dialog Box General properties 162
  - Group Box properties 168
  - Menu item properties 175
  - Pushbutton properties 171
  - Radio Button properties 172
  - String properties 176
  - Text properties 173
  - User Control properties 174
  - VBX Control General properties 174
- Cascading menu, creating 64
- CDialog class 115
- Centering controls in a dialog box 53
- CFormView class 115
- Changed source code
  - ClassWizard, updating for 131
  - rebuilding ClassWizard file 132–133
- Changing
  - color palette 108
  - graphics properties 85
  - name of RESOURCE.H 29
  - symbol header file name 29
  - symbol numerical value 33–34
  - symbol or symbol name 32
  - unassigned symbols 35
- Characters, adding ASCII or special characters to a string 79
- Check Box properties
  - Auto 165
  - Caption 164
  - Disabled 165
  - Group 165
  - ID 164
  - Left Text 165
  - property page reference 164
  - Tabstop 165
  - Tri-State 165
  - Visible 164

- Checked (Menu Item properties) 175
- Class (User Control properties) 174
- Class Library version 1 programs, updating for use with ClassWizard 133
- Classes
  - CDialog 115
  - CFormView 115
  - ClassWizard, types created in 114
  - creating new in ClassWizard 115–116
  - deleting in ClassWizard 131
  - importing in ClassWizard 117
  - moving in ClassWizard 132
  - renaming in ClassWizard 132
- ClassWizard
  - .CLW file
    - defined 22,131
    - rebuilding 132
  - Add Member Variable dialog box
    - illustrated 122
    - Property box 122
    - Variable Type box 122
  - adding a new class 114–116
  - App Studio toolbar button 115
  - classes, types created in 114
  - comments 116, 133
  - defined 113
  - Edit Code command 120
  - Edit Member Variables dialog box 122
  - example 125
  - how it works 113
  - importing a class 117
  - overview 113–114
  - Repair Class Information dialog box 131
  - updating 131
  - updating code for use with 133
  - what it modifies 116
  - what it's used for 113
- Clearing selected area of graphic 91
- Clip Children (Dialog Box Styles properties) 164
- Clip Siblings (Dialog Box Styles properties) 163
- Clipboard, pasting from, graphics editor 91
- Closed figures, drawing 87–88
- .CLW file 131
  - defined 22
  - rebuilding 132
- Code
  - editing from ClassWizard 120–121
  - updating for use with ClassWizard 133–134
- Color
  - adding custom 108–109
  - changing 108
    - changing number of, graphics editor 96
  - editing 107–109
  - inverse
    - Color (*continued*)
      - inverse (*continued*)
        - changing 104
        - drawing with 103–104
        - using 93
      - palettes, saving or loading 110
      - picking up 89
      - screen
        - changing 104
        - drawing with 103–104
        - selecting 86
        - transferring 89
  - Color (Picture properties) 171
- Colors
  - Bitmap properties 161
  - Icon properties 175
- Combo box
  - drop-down arrow 48
  - drop-downs, sizing 47–48
- Combo Box General properties
  - Caption 165
  - Disabled 165
  - Enter List Choices 165
  - Group 165
  - ID 165
  - property page reference 165
  - Tabstop 165
  - Visible 165
- Combo Box Styles properties
  - Auto HScroll 166
  - Combo Box Styles properties 166
  - Disable No Scroll 166
  - Has Strings 166
  - No Integral Height 166
  - OEM Convert 166
  - Owner Draw 166
  - property page reference 166
  - Sort 166
  - Type 166
  - Vert. Scroll 166
- Comments
  - in .RC files 25
  - used by ClassWizard 116, 133
- COMMON.RES
  - described 19
  - using 19
- Compiler directives
  - including 31
  - limits on 25
  - using 30–31
- Conditional compilation statements, limits on 25
- Control bar quick reference 149
- Control, dominant
  - changing 51

- Control, dominant (*continued*)
  - defined 51
  - effect on aligning 52–54
- Control palette
  - hiding 43
  - Quick reference 151
  - showing 43
- Controls
  - adding
    - alternate methods 43
    - by drawing 44
    - introduction 42
    - with drag and drop 43
    - with point and click 43
  - aligning
    - on center 53
    - overview 52
    - procedure 52–54
  - arranging 51–54
  - centering in dialog box 53
  - copying 46
  - deleting 46
  - making the same size, height, or width 54
  - mnemonic
    - defining 50
    - for controls with a caption 50
    - for controls without a caption 50
  - moving
    - between dialog boxes 46
    - one DLU at a time 46
    - overview 46
    - within a dialog box 46
  - placing
    - by drawing 44
    - with drag and drop 43
    - with point and click 43
  - repositioning, with drag and drop 43
  - selected
    - changing 46
    - described 45
  - selecting 45
  - sizing
    - one DLU at a time 47
    - to fit caption 47
  - spacing evenly 53
- Coordinates, dialog box 42
- Copying
  - accelerator keys 72
  - controls 46
  - graphics, selected area of 92
  - menus and menu items
    - overview 65
    - using drag and drop 66
    - using the Edit menu 66
- Copying (*continued*)
  - resources between files 26–27
- Creating
  - cascading (hierarchical) menu 64
  - drop-down menu 63
  - form view 59
  - graphics
    - as part of a resource script file 83
    - as a stand-alone file 84
  - menu item 63–64
  - new
    - accelerator table 69
    - class in ClassWizard 115
    - menu resource 62
    - resource script file 22–23
  - resources 5
  - string table 75
  - symbols 35
- Cropping, graphics 95
- CToolbar class 104
- .CUR file
  - exporting 28
  - importing 28
- Cursor
  - device image, creating new 101
  - hotspot, setting 104
- Cursor devices, defining 178
- Cursor Editing toolbar, Quick reference 152
- Cursor properties
  - Filename 161
  - Height 161
  - Hotspot 162
  - ID 161
  - property page reference 161
  - Width 161
- Cursor resource
  - defined 19
- Cursors
  - See also* Graphics
  - See also* Images
  - Colors, Quick reference to tasks 143
  - editing 100
  - Quick reference to tasks 142
- Custom
  - brush
    - creating 93
    - using 94
  - colors, adding 108–109
  - controls, defined 54
  - resources
    - creating 111
    - editing in App Studio 111
    - including 31
- Cutting, graphics 91

## D

- Data resource, creating 111
- DDV *See* Dialog Data Validation
- DDX *See* Dialog Data Exchange
- Default .RC file type, setting 177
- Default Button (Pushbutton properties) 171
- Defining, accelerator keys 72–73
- Deleting
  - accelerator keys 72
  - classes using ClassWizard 131
  - controls 46
  - graphic, selected area of 91
  - resources 5
  - strings in string table 78
  - symbols, unassigned 36
  - VBX controls 56
- Device image
  - cursor
    - creating new 101
    - overview 101
  - deleting 102
  - description, editing 102–103
  - icon
    - creating new 101
    - overview 101
  - selecting 102
- Dialog
  - Check Box Properties 164
  - Combo Box Properties
    - General 165
    - Styles 166
  - Edit Box Properties
    - General 166
    - Styles 166
  - Group Box Properties 168
  - List Box Properties
    - General 168
    - Styles 169
  - Picture Properties 170
  - Pushbutton Properties 171
  - Radio Button Properties 172
  - Scrollbar Properties 172
  - Text Properties 173
  - User Control Properties 174
  - VBX Control Properties
    - General 174
    - Styles 174
- Dialog box
  - controls 39
  - coordinates 42
  - creating
    - new 40
    - overview 40
  - Dialog box (*continued*)
    - creating (*continued*)
      - with ClassWizard (example) 125
    - defined 16
    - editing 41
    - font 42
    - keyboard access, defining 50
    - mnemonics
      - defining 50
      - for controls with a caption 50
      - for controls without a caption 50
    - opening 41
    - Quick reference
      - Aligning controls 140
      - General tasks 139
      - Sizing controls 141
      - Spacing and positioning controls 141
    - symbol, changing name or value of 41
    - tab order
      - changing 48–49
      - defined 48
      - setting 48–49
    - testing 59–60
    - units 42
- Dialog box class, defining with ClassWizard (example) 126
- Dialog box controls *See* Controls
- Dialog Box Data, overview 121
- Dialog Box General properties
  - Caption 162
  - Font 162
  - Font Name 162
  - Font Size 162
  - ID 162
  - Menu 162
  - Registered Class 162
  - X Pos 162
  - Y Pos 162
- Dialog Box Styles properties
  - Absolute Align 164
  - Border 163
  - Clip Children 164
  - Clip Siblings 163
  - Disabled 164
  - Horiz. Scroll 163
  - LocalEdit 164
  - Maximize Box 163
  - Minimize Box 163
  - NoIdleMsg 164
  - Style 163
  - System Menu 163
  - System Modal 164
  - Titlebar 163
  - Vert. Scroll 163
  - Visible 164

- Dialog box template *See* Dialog box
  - Dialog box units 42
  - Dialog Data Exchange
    - Control variables 122, 130
    - defined 121
    - overview 121
    - setting up with ClassWizard 122
    - Value variables 122
    - variables
      - creating 122
      - defining, example 128
      - setting initial value, example 124
      - using 124
  - Dialog Data Validation
    - custom 125
    - defined 124
    - overview 121
    - using, example 128
  - Dialog editor
    - overview 39
    - toolbar, Quick reference 150
  - Dialog Properties
    - General 162
    - Styles 163
  - Directives, compiler
    - including 31
    - using 30–31
  - Disable No Scroll
    - Combo Box Styles properties 166
    - List Box Styles properties 170
  - Disabled
    - Check Box properties 165
    - Combo Box General properties 165
    - Dialog Box Styles properties 164
    - Edit Box General properties 166
    - Group Box properties 168
    - List Box General properties 168
    - Picture properties 170
    - Pushbutton properties 171
    - Radio Button properties 172
    - Scrollbar properties 172
    - Text properties 173
    - User Control properties 174
    - VBX Control General properties 174
  - Discardable (Resource properties) 159
  - Display devices
    - default in image editor 100
    - types supported 101
  - Displaying, graphics palette 86
  - .DLL files 22
  - DLUs *See* Dialog box units
  - Dominant control
    - alignment, effect on 52
    - changing 51
  - Dominant control (*continued*)
    - defined 51
    - described 45
    - effect on aligning 53–54
    - setting 45
  - Drag and drop
    - controls 43
    - illustrated 8
    - using 7–8
  - Drawing
    - figure, closed 88
    - figures, closed 87
    - freehand 86
    - lines 87–88
    - tools
      - changing size of 87
      - using 86–87
      - with selection 92
  - Drop-down
    - arrow 48
    - combo box 47–48
    - menu
      - creating 63
      - viewing the menu resource as 66
  - Dynamic-link library 22
- ## E
- Edit Box General properties
    - Disabled 166
    - Group 166
    - ID 166
    - property page reference 166
    - Tabstop 166
    - Visible 166
  - Edit Box Styles properties
    - Align Text 166
    - Auto HScroll 167
    - Auto VScroll 167
    - Border 168
    - Horiz. Scroll 167
    - Lowercase 168
    - Multiline 166
    - No Hide Sel 167
    - OEM Convert 167
    - Password 167
    - property page reference 166
    - Read Only 168
    - Uppercase 168
    - Vert. Scroll 167
    - Want Return 168
  - Edit menu
    - Copy command
      - accelerators 72

**Edit menu** (*continued*)Copy command (*continued*)

- controls 46
- graphics 91
- menu items 66

## Cut command

- accelerators 72
- controls 46
- graphics 91
- menu items 66
- strings 78

## Delete command

- accelerators 72
- controls 46
- strings 78

## Paste command

- accelerators 72
- controls 46
- graphics 91
- strings 78

## Quick reference 147

## Redo command 9

## Symbols 34

## Undo command 9

**Edit Variables dialog box** (ClassWizard) 122**Editing**

## accelerator

- properties 72–73
- table 70–72

## binary data

- opening a resource for editing 111
- using the binary data editor window 112

## bitmaps 104–107

## code, in ClassWizard 120

## cursors 100

## dialog box 41

## dialog box resource 42

## from Symbol Browser 36

## graphics

- as a stand-alone resource 84
- overview 83, 91
- part of a resource script file 84
- properties 85

## icons 100

## menu resource 62

## resources 5, 7

## resources at binary level 111

## session, beginning 4

## string table 77

## user-defined control properties 58

## VBX control properties 56–57

## windows, overview 10–14

**Editor**

## accelerator table

- described 70
- illustrated 71

## dialog box 41

## graphics

- overview 81
- view, adjusting 97

## image

- described 82
- illustrated 81

## string table 75

**Enter List Choices** (Combo Box General properties) 165**Eraser tool**

- drawing with 86–87
- single pixel 87

**Erasing, freehand** 86**Escape sequences, adding to a string** 79**Even spacing between controls** 53**.EXE files** 22**Executable file** 22**Existing resource files, opening** 24**Exporting graphics files** 28**Extending graphics** 95**F****Factor, magnification**

- changing 99–100
- defined 99

**Figure, closed, drawing** 87–88**File management, Quick reference** 137**File menu**

- Install Controls command 56
- Install controls command 55
- New command 23, 84
- Quick reference 146
- Set Includes command 29–31

**Filename**

- Bitmap property page 161
- Cursor property page 161
- Icon properties 175
- Resource property page (bitmaps, icons, cursors) 160

**Files**

- .APS 22
- .CLW 22
- .DLL 22
- .EXE 22
- .RC 22
- .RES 22
- App Studio, described 22
- executable 22
- graphics
  - .BMP, .DIB, .ICO, .CUR 22



Files (*continued*)graphics (*continued*)

exporting 28

importing 28

opening, file types listed 22

other resource files, including 30–31

## resource

consequences of updating to App Studio format 25

opening existing 24

## resource script

overview 15–16

working with 21–22

## RESOURCE.H 22

saving, file types listed 22

symbol header file, changing the name of 29

## Fill tool 88

## Filling areas (graphics editor) 88

## Filter, message (ClassWizard) 119

## Finding

graphics, with resource browser window 84

strings in string table 77

## Flipping graphics 93

## Font

dialog box 42

Dialog Box General properties 162

## Font Name, Dialog Box General properties 162

## Font Size, Dialog Box General properties 162

## Foreground color, selecting 86

## Form view

controls 39

creating 59

defined 59

using 59

## Formatting, adding to a string 79

## Freehand drawing and erasing 86

## Functions, message handling

creating 117–120

deleting 120

**G**Graphic, static, properties *See* Picture propertiesGraphical resources *See* Graphics

## Graphics

*See also* Bitmaps*See also* Cursors*See also* Icons

backgrounds, setting 92

browsing, using resource browser window 84

changing number of colors in 96

copying, selected area of 92

## creating

as part of a resource script file 83

as a stand-alone file 84

Graphics (*continued*)

## cropping 95

cutting 91

## editing

as a stand-alone resource 84

overview 83, 91

part of a resource script file 84

## extending 95

## files

exporting 28

importing 28

loading 28

saving 28

types 22

finding, using resource browser window 84

## flipping 93

moving selected area of 92

## opening

as a stand-alone resource 84

part of a resource script file 84

pasting in the graphics editor 91

## properties

changing 85

setting 85

## Quick reference to tasks 142

## resizing

overview 94

using Properties window 96

## Rotating 93

## selecting

entire 91

parts of 89

## shrinking 96

## sizing

overview 94

using Properties window 96

## stretching 96

## toolbar

editing 106–107

overview 104

## Graphics editor

graphics palette (quick reference) 151

## grid, pixel

hiding 100

showing 100

overview 81

view, adjusting 97

workspace, managing 97

## Graphics palette

described 82

displaying 86

hiding 86

illustrated 81

inverse-color selector 104

Graphics palette (*continued*)  
  screen-color selector 104  
  showing 86  
Grayed (Menu item properties) 176  
Grid Settings command 52  
Grid  
  layout (dialog editor) *See* Grid, Snap to  
  pixel (graphics editor)  
    hiding 100  
    showing 100  
  Snap to  
    defined 51  
    using 51–52  
Group  
  Check Box properties 165  
  Combo Box General properties 165  
  Edit Box General properties 166  
  Group Box properties 168  
  List Box General properties 168  
  Picture properties 170  
  Pushbutton properties 171  
  Radio Button properties 172  
  Scrollbar properties 172  
  Text properties 173  
  User Control properties 174  
  VBX Control General properties 174  
Group Box properties  
  Caption 168  
  Disabled 168  
  Group 168  
  ID 168  
  property page reference 168  
  Tabstop 168  
  Visible 168  
Guides, insertion 66

## H

Handlers, message  
  creating with ClassWizard 117–120  
  example 127  
Handles, sizing 47  
Has Strings  
  Combo Box Styles properties 166  
  List Box Styles properties 169  
Height  
  Bitmap property page 161  
  Cursor property page 161  
  Icon properties 175  
Help  
  App Studio  
    Quick reference 149  
    tasks, Quick reference to 146  
  getting help on Windows messages 119

Help (*continued*)  
  Menu item properties 176  
Hiding  
  control palette 43  
  graphics palette 86  
  Properties window 10  
Hierarchical menu, creating 64  
Horiz. Scroll  
  Dialog Box Styles properties 163  
  Edit Box Styles properties 167  
  List Box Styles properties 170  
Hotspot  
  (Cursor properties) 162  
  cursor 104

## I

.ICO file  
  exporting 28  
  importing 28  
Icon (Picture properties) 171  
Icon devices 178  
Icon Editing toolbar (quick reference) 151  
Icon properties  
  Colors 175  
  Filename 175  
  Height 175  
  ID 175  
  property page reference 175  
  Width 175  
Icon resource, defined *See* Icons  
Icons  
  *See also* Graphics  
  *See also* Images  
  colors (quick reference to tasks) 143  
  device image 101  
  editing 100  
  quick reference to tasks 142

## ID

Accelerator table property page 160  
Bitmap property page 161  
Check Box properties 164  
Combo Box General properties 165  
Cursor property page 161  
Dialog Box General properties 162  
Edit Box General properties 166  
Group Box properties 168  
Icon properties 175  
List Box General properties 168  
Menu item properties 175  
Menu Properties 175  
Picture properties 170  
Pushbutton properties 171  
Radio Button properties 172

ID (*continued*)

- Resource property page 159
  - Scrollbar properties 172
  - String properties 176
  - Text properties 173
  - User Control properties 174
  - VBX Control General properties 174
- Identifier *See* Resource identifier
- Image editor
- See also* Graphics editor
  - illustrated 81–82
  - panes 97
  - view, adjusting 97
- Image menu
- color, editing foreground 108
  - described 83
  - Flip Horizontal command 93
  - Flip Vertical command 93
  - Quick reference 148
  - Set Cursor Hotspot 104
  - Zoom command 98
- Image, device
- creating new 101
  - deleting 102
  - description, editing 102–103
  - selecting 102
  - types supported 101
- Importing
- class, with ClassWizard 114, 117
  - graphics files 28
- Inactive (Menu item properties) 176
- Included resource script files 25
- Including
- calculated symbols 29–30
  - compiler directives 31
  - read-only symbols 29–30
  - resource-file macros 31
  - resources from other files 30–31
  - shared symbols 29–30
- Initialization, APSTUDIO.INI file 177
- Inserting, strings in string table 77–78
- Insertion guides 66
- Installing VBX controls 55
- Inverse color
- changing 104
  - drawing with 103–104
- Inverting, colors 93
- K**
- Key (Accelerator table properties) 160
  - Key reference 152
  - Keyboard access, dialog box 50

**L**

- Layout menu
- Align Horiz. Center command 53
  - Align Vert. Center command 53
  - Arrange Buttons command 54
  - Grid Settings command 52
  - Quick reference 148
  - Set Tab Order command 49
  - Size to Content command 47
- Left Text
- Check Box properties 165
  - Radio Button properties 172
- Lines, drawing 87–88
- List Box General properties
- Disabled 168
  - Group 168
  - ID 168
  - property page reference 168
  - Tabstop 168
  - Visible 168
- List Box Styles properties
- Border 169
  - Disable No Scroll 170
  - Has Strings 169
  - Horiz. Scroll 170
  - Multi column 170
  - No Integral Height 170
  - No Redraw 170
  - Notify 170
  - Owner Draw 169
  - property page reference 169
  - Selection 169
  - Sort 169
  - Use Tabstops 170
  - Vert Scroll 170
  - Want Key Input 170
- Loading graphics files 28
- LocalEdit, Dialog Box Styles properties 164
- Lowercase, Edit Box Styles properties 168
- M**
- Macros in resource file, including 31
  - Magnification factor
    - changing 99–100
    - defined 99
    - setting default 178
  - Making spacing even between controls 53
  - Managing graphics editor workspace 97
  - Map, message, defined 113
  - Mapping messages to functions 117–120
  - Maximize Box (Dialog Box Styles properties) 163
  - Member variables, Dialog Data Exchange 122

- Menu
    - App Studio, quick reference 146
    - Dialog Box General properties 162
  - Menu editor terms, defined 61
  - Menu item and accelerator, associating 66–67
  - Menu item properties
    - Break 176
    - Caption 175
    - Checked 175
    - Grayed 176
    - Help 176
    - ID 175
    - Inactive 176
    - Popup 176
    - Prompt 176
    - property page reference 175
    - Separator 175
  - Menu items
    - accelerator, associating with 73–74
    - creating 63–64
    - copying
      - options 65
      - using drag and drop 66
      - using the Edit menu 66
    - moving
      - options 65
      - using drag and drop 66
      - using the Edit menu 66
    - selecting 64–65
    - styles 64
  - Menu Properties
    - ID 175
    - property page reference 175
  - Menu resource
    - creating new 62
    - editing 62
    - viewing as drop-down 66
    - viewing as drop-down menu 66
  - Menus
    - copying
      - options 65
      - using drag and drop 66
      - using the Edit menu 66
    - defined 16
    - Menu Item Properties 175
    - moving
      - options 65
      - using drag and drop 66
      - using the Edit menu 66
    - Quick reference to tasks 142
    - selecting 64–65
    - single item, creating 63
    - terms 61
  - Message filter, ClassWizard 119
  - Message handlers
    - creating with ClassWizard (example) 127
    - deleting 120
  - Message map, defined 113
  - Messages
    - getting help on 119
    - mapping to functions 117–120
  - Microsoft Foundation Class Library
    - resource files
      - adding framework support to 26
      - features supported in 26
    - support for 23
    - version 1, updating for ClassWizard 133
  - Minimize Box (Dialog Box Styles properties) 163
  - Mnemonic
    - control
      - defining 50
      - for controls with a caption 50
      - for controls without a caption 50
    - dialog box
      - defining 50
      - for controls with a caption 50
      - for controls without a caption 50
    - menu, defining 63
    - menu item, defining 64
  - Modifiers (Accelerator table properties) 160
  - Moveable (Resource properties) 159
  - Moving
    - accelerator keys 72
    - between property pages 10
    - classes, in ClassWizard 132
    - controls
      - between dialog boxes 46
      - one DLU at a time 46
      - within a dialog box 46
    - graphic, selected area of 92
    - menus and menu items
      - options 65
      - using drag and drop 66
      - using the Edit menu 66
    - resources 5
    - strings
      - between files 78
      - in string table 78
  - Multicolumn (List Box Styles properties) 170
  - Multiline (Edit Box Styles properties) 166
- ## N
- Name restrictions, symbol 36–37
  - New resource
    - creating 6
    - script file 22–23
  - New-item box 63

- Next Key Typed (Accelerator table properties) 161
- No Hide Sel (Edit Box Styles properties) 167
- No Integral Height
  - Combo Box Styles properties 166
  - List Box Styles properties 170
- No Prefix (Text properties) 173
- No Redraw (List Box Styles properties) 170
- No Wrap (Text properties) 173
- NoIdleMsg (Dialog Box Styles properties) 164
- Notify (List Box Styles properties) 170

## O

- Objects, manipulating user-interface 7
- OEM Convert (Edit Box Styles properties) 167
- Online help (App Studio) 146
- Opaque background in graphics 92
- Opening
  - accelerator table 70
  - binary data editor 111
  - dialog box editor 41
  - files 22
  - graphics
    - as a stand-alone resource 84
    - part of a resource script file 84
  - resource files
    - from App Studio 24
    - from Visual Workbench 23–24
  - resource script file 4
  - string table 77
- Overview, App Studio 3
- Owner Draw
  - Combo Box Styles properties 166
  - List Box Styles properties 169
  - Pushbutton properties 172

## P

- Paint bucket tool 88
- Palette
  - colors
    - adding custom color to 108–109
    - changing 108
    - editing 107
    - saving or loading 110
  - graphics
    - described 82
    - illustrated 81
    - inverse-color selector 104
    - screen-color selector 104
- Panes, adjusting in image editor 97
- Password (Edit Box Styles properties) 167
- Pasting from clipboard to graphics 91

- Pencil tool
  - changing size of 87
  - drawing with 86–87
- Picking up colors 89
- Picture properties
  - Color 171
  - Disabled 170
  - Group 170
  - Icon 171
  - ID 170
  - property page reference 170
  - Tabstop 170
  - Type 171
  - Visible 170
- Pixel grid (graphics editor)
  - hiding 100
  - showing 100
- Placing controls 42
- Popup
  - Menu item properties 176
  - option, using 63–64
- Preferences, user
  - In APSTUDIO.INI file 177
- Preload (Resource properties) 159
- Preview (Resource properties) 160
- Prompt
  - Menu item properties 176
  - string, menu item 64
- Properties
  - accelerator 72–73
  - graphics
    - editing 85
    - setting 85
  - resource 9
- Properties window 41
  - See also* Property pages
  - controlling 10
  - hiding 10
  - overview 9
  - property page reference 159
  - pushpin 10
  - toolbar quick reference 150
  - using 9
  - viewing 10
- Property page reference
  - Accelerator table properties 160
  - Bitmap Properties 161
  - Check Box properties 164
  - Combo Box General properties 165
  - Combo Box Styles properties 166
  - Cursor properties 161
  - Dialog General properties 162
  - Dialog Styles properties 163
  - Edit Box General properties 166

Property page reference (*continued*)

- Edit Box Styles properties 166
- Group Box properties 168
- Icon properties 175
- List Box General properties 168
- List Box Styles properties 169
- Menu item properties 175
- Menu Properties 175
- overview 159
- Picture properties 170
- Pushbutton properties 171
- Radio Button properties 172
- Resource properties 159
- Scrollbar properties 172
- String properties 176
- Text properties 173
- User Control properties 174
- VBX Control General properties 174
- VBX Control Styles properties 174

Property page, Resource 6

Property pages

- See also* Properties window
- moving between 10
- viewing 10

Pure (Resource properties) 160

Pushbutton properties

- Caption 171
- Default Button 171
- Disabled 171
- Group 171
- ID 171
- Owner Draw 172
- property page reference 171
- Tabstop 171
- Visible 171

Pushbuttons

- arranging 54
- in new dialogs 178

Pushpin, properties window 10

## Q

Quick reference

- Accelerator keys (App Studio)
  - binary editing 157
  - creating and editing resources 153
  - editing 152, 156–157
  - introduction 137
  - managing 152, 154
  - online help 157
  - overview 152
  - Properties window 154
- Accelerator tables 144

Quick reference (*continued*)

- App Studio
  - toolbar 150
  - windows 145
- Bitmaps, icons, and cursors
  - colors 143
  - general 142
- ClassWizard 144
- common App Studio tasks 137
- Control palette (dialog editor) 151
- Cursor Editing toolbar 152
- dialog box tasks
  - aligning controls 140
  - general 139
  - sizing controls 141
  - spacing and positioning controls 141
- Dialog editor toolbar 150
- Edit menu 147
- file management tasks 137
- File menu 146
- Graphics Editor palette 151
- Help menu 149
- Icon Editing toolbar 151
- Image menu 148
- introduction 137
- Layout menu 148
- menu commands 146
- menu resources 142
- menus 137
- online help 146
- Properties Window toolbar 150
- resource editing tasks 138–139
- Resource menu 147
- resource tasks, general 139
- String tables 144
- Symbol editor 145
- Toolbar buttons 137
- Toolbars and palettes 149
- VBX controls 145
- Window menu 149

## R

Radio Button properties

- Auto 172
- Caption 172
- Disabled 172
- Group 172
- ID 172
- Left Text 172
- property page reference 172
- Tabstop 172
- Visible 172

RCDATA resources, including 31

- .RC file type
    - setting default 177
  - .RC files 22
    - included 25
  - Read Only (Edit Box Styles properties) 168
  - Read-only symbols
    - included 25
    - Including 30
    - including 29
  - Reading in graphics files 28
  - Rebuilding ClassWizard (.CLW) file 132
  - Redo command 8–9
  - Reference, property page 159
  - Registered Class (Dialog Box General properties) 162
  - Removing VBX controls 56
  - Renaming classes in ClassWizard 132
  - Repair Class Information dialog box 131
  - .RES files 22
  - Resizing
    - See also* Sizing
    - controls 54
  - Resource, dialog box ??
  - Resource browser window
    - creating new resource with 6
    - using 5, 7, 41–42, 62, 70, 77
  - Resource files
    - advanced techniques, overview of 28–29
    - comments 25
    - consequences of updating to App Studio format 25
    - creating new 22–23
    - including resources from other files 30–31
    - macros, including 31
    - opening
      - from App Studio 24
      - from Visual Workbench 23–24
      - not an App Studio or AppWizard file 24
    - support for Microsoft Foundation Class Library
      - adding 26
      - described 26
  - Resource identifier, strings, changing 79
  - Resource menu
    - ClassWizard button 115
    - ClassWizard command 119
    - New command 40, 62, 69, 76, 84
    - Open command 62
    - Quick reference 147
    - Test command 60
  - Resource properties
    - bitmaps, icons, cursors
      - Filename 160
      - Preview 160
    - property page reference 159
  - Resource property page
    - Discardable 159
  - Resource property page (*continued*)
    - ID 159
    - Moveable 159
    - overview 6
    - Preload 159
    - Pure 160
  - Resource script file
    - creating new 22–23
    - defined 15
    - included 25
    - opening 4
  - RESOURCE.H
    - changing the name of 29
    - defined 22
  - Resources
    - accelerator table, defined 17
    - bitmap, defined 18
    - browsing 5–6
    - copying between files 26–27
    - creating 5, 138
    - creating new 6
    - cursor, defined 19
    - custom
      - creating 111
      - including 31
    - data, creating 111
    - deleting 5
    - dialog box, defined 16
    - editing 5, 7, 36, 138
    - handling 138
    - icon 18
    - menu 16
    - moving 5
    - open existing 42
    - properties 9
    - quick reference to tasks 139
    - RCDATA 31
    - sample 19
    - shared 31
    - string table 18
    - viewing 5–6
    - working with 21
  - Resources, graphical *See* Graphics
  - Reversing color 93
  - Rotating graphics 93
- ## S
- Sample resources 19
  - Save Compressed (Bitmap property page) 161
  - Saving
    - files 22
    - graphics files 28

- Screen color
  - changing 104
  - drawing with 103–104
- Script file *See* Resource script file
- Scrollbar properties
  - Align 173
  - Disabled 172
  - Group 172
  - ID 172
  - property page reference 172
  - Tabstop 172
  - Visible 172
- Searching in string table 77
- Select Source Files dialog box (ClassWizard) 133
- Selected control 45
- Selecting
  - colors, foreground and background 86
  - controls 45
  - copying selected area (graphics editor) 92
  - cutting selected area (graphics editor) 91
  - deleting selected area (graphics editor) 91
  - drawing with selection (graphics editor) 92
  - entire graphics 91
  - inverting color of selection (graphics editor) 93
  - menus or menu items 64–65
  - moving selected area (graphics editor) 92
  - parts of graphics 89
- Selection *See* Selecting
- Selection (List Box Styles properties) 169
- Separator (Menu Item properties) 175
- Setting
  - cursor hotspot 104
  - graphics properties 85
- Shared
  - resources 31
  - symbols 29–30
- Show control palette 43
- Showing graphics palette 86
- Shrinking, graphics 96
- Simple (Text properties) 173
- Sizing
  - See also* Resizing
  - combo box drop-downs 47–48
  - controls 51
  - individual controls
    - one DLU at a time 47
    - to fit caption 47
- Sizing handles 47
- Snap to Grid
  - defined 51
  - placing controls, affect on 43
  - using 51–52
- Sort
  - Combo Box Styles properties 166
  - Sort (*continued*)
    - List Box Styles properties 169
  - Source code, updating ClassWizard 131–132
  - Starting, App Studio 4
  - Static graphic properties *See* Picture properties
  - Static text properties *See* Text properties
  - Status bar
    - image editor 83
    - position and size indicators 42
    - prompt string 64
  - Stretching graphics 96
  - String editor 75
  - String properties
    - Caption 176
    - ID 176
    - property page reference 176
    - String Editor 176
  - String table
    - See also* Strings
    - creating new 75
    - defined 18, 75
    - editing 77
    - editor, overview 75
    - empty deleted 76
    - opening 77
    - Quick reference to tasks 144
  - Strings
    - See also* String Table
    - identifier 79
    - moving, between files 78
    - string table
      - adding 77–78
      - changing 79
      - deleting 78
      - finding 77
      - inserting 77–78
      - moving 78
      - searching for 77
  - Style
    - Dialog Box Styles properties 163
    - User Control properties 174
  - Styles
    - menu item 64
    - Windows *See* Properties, resource
  - Symbol Browser
    - opening 34
    - using 34
    - View Use command 36
  - Symbol header file, changing the name of 29
  - Symbols
    - calculated
      - including 30
      - overview 29
    - changing 32–34



Symbols (*continued*)

- creating new 35
  - defined as expressions 25
  - deleting unassigned 36
  - included as read-only 25
  - name restrictions 36–37
  - new, creating 35
  - quick reference to tasks 145
  - read-only
    - including 30
    - overview 29
  - shared
    - including 30
    - overview 29
  - value
    - changing 33–35
    - restrictions 36–38
  - working with
    - introduction 31–32
    - overview 21
- System Menu (Dialog Box Styles properties) 163
- System Modal (Dialog Box Styles properties) 164

**T**

## Tab order, dialog box

- changing 49
  - defined 48
  - setting 48–49
- Table, accelerator *See* Accelerator table
- Table, string *See* String tables
- Tabstop
  - Check Box properties 165
  - Combo Box General properties 165
  - Edit Box General properties 166
  - Group Box properties 168
  - List Box General properties 168
  - Picture properties 170
  - property, using 48
  - Pushbutton properties 171
  - Radio Button properties 172
  - Scrollbar properties 172
  - Text properties 173
  - User Control properties 174
  - VBX Control General properties 174
- Task quick reference 137
- Template, dialog box *See* Dialog box
- Terms, menu editor 61
- Testing
  - dialog box 59–60
  - graphic with inverted colors 93
- Text Align (Text properties) 173
- Text properties
  - Caption 173

Text properties (*continued*)

- Disabled 173
  - Group 173
  - ID 173
  - No Prefix 173
  - No Wrap 173
  - property page reference 173
  - Simple 173
  - Tabstop 173
  - Text Align 173
  - Visible 173
- Tile grid
  - defined 105
  - displaying or hiding 106
  - setting dimensions 105
- Titlebar (Dialog Box Styles properties) 163
- Toolbar
  - App Studio
    - ClassWizard 115, 119
    - New Accelerator Table 69
    - New Bitmap 84
    - New Cursor 84
    - new dialog box 40
    - New File 23
    - New Icon 84
    - New Menu 62
    - New String Table 76
    - new-resource buttons 6
    - Redo 9
    - Symbol Browser 34
    - Undo 9
  - bitmaps
    - editing 104–107
    - overview 104
  - dialog editor
    - Align Bottom 52
    - Align Left 52
    - Align Right 52
    - Align Top 52
    - Center Horizontal 53
    - Center Vertical 53
    - Make Same Height 54
    - Make Same Size 54
    - Make Same Width 54
    - Snap to Grid 51
    - Space Evenly Across 53
    - Space Evenly Down 53
    - Test 60
  - graphics
    - editing 106–107
    - overview 104–105
  - quick reference
    - introduction 149
    - overview 137

Toolbar (*continued*)

- tile grid
  - defined 105
  - displaying or hiding 106
  - setting dimensions 105

## Toolbox, graphics editor, Zoom tool 98–99

## Tools

- closed figure 88
- drawing 86
  - changing size of 87
  - using 87
- fill 88
- line 87– 88
- selecting 86

## Transferring colors 89

## Transparent background (graphics) 92

## Tri-State (Check Box properties) 165

## Type

- Accelerator table property page 160
- Combo Box Styles properties 166
- Picture properties 171

## U

## Undo 8–9

## Undo levels 8, 178

## Updating

- ClassWizard 131
- code for ClassWizard 133

## Uppercase (Edit Box Styles properties) 168

## Use Tabstops (List Box Styles properties) 170

## UseMfc (APSTUDIO.INI) 23

## User Control properties

- Caption 174
- Class 174
- Disabled 174
- Group 174
- ID 174
- Style 174
- Tabstop 174
- Visible 174

## User preferences in APSTUDIO.INI 177

## User-defined controls

- differences between VBX controls and 58
- overview 55
- properties 58
- using 54, 57

## User-interface objects, manipulating 7

## Using

- Properties window 9
- redo 8
- Symbol Browser, overview 34
- undo 8
- user-defined controls 57

## V

## Values

- accelerator 73
- symbol
  - changing 33
  - restrictions 36–38

## Variables, DDX

- Control variables
  - example 130
  - using 122
- creating
  - example 128
  - using 122
- setting initial value (example) 129
- using 124
- Value 122

## VBX Control General properties

- Caption 174
- Disabled 174
- Group 174
- ID 174
- property page reference 174
- Tabstop 174
- Visible 174

## VBX Control Styles properties, property page reference 174

## VBX controls

- adding to control palette 55
- advantages 54
- defined 54
- deleting 56
- deleting from control palette 56
- differences between user-defined controls and 58
- Grid control 39
- installing 55
- overview 54
- properties 56–57
- quick reference to tasks 145
- using 55

## Vert. Scroll

- Combo Box Styles properties 166
- Dialog Box Styles properties 163
- Edit Box Styles properties 167
- List Box Styles properties 170

## View, graphics editor

- adjusting 97
- zooming in
  - on location under cursor 99
  - or out 98

## Viewing

- Properties window 10
- property pages 10
- resources 5–6

## Visible

- Check Box properties 164
- Combo Box General properties 165
- Dialog Box Styles properties 164
- Edit Box General properties 166
- Group Box properties 168
- List Box General properties 168
- Picture properties 170
- Pushbutton properties 171
- Radio Button properties 172
- Scrollbar properties 172
- Text properties 173
- User Control properties 174
- VBX Control General properties 174

Visual Workbench, opening resource files from 23–24

**W**

Want Key Input (List Box Styles properties) 170

Want Return (Edit Box Styles properties) 168

## Width

- Bitmap property page 161
- Cursor property page 161
- Icon properties 175

## Window

- Properties 41
- resource browser 41–42

## Window menu

- Hide Control Palette 43
- Quick reference 149
- Show Control Palette 43
- Show Properties command 10

## Windows resources

- basics 15
- defined 15
- overview 15

## Windows, App Studio

- overview 10–14
- quick reference to tasks 145

Working with symbols 31–32

**X**

## X Pos

- Dialog Box General properties 162

**Y**

## Y Pos

- Dialog Box General properties 162

**Z**

## Zooming

- graphics editor view 98, 99
- See also* Magnification Factor





# Documentation Feedback—Microsoft® Visual C++™ Version 1.0

We need your feedback to improve our documentation. When you become familiar with this product, please complete and return this form. Use the back of the form to make comments and suggestions, noting errors and special strengths in areas such as online help, programming examples, indexes, and organization. Note that your comments and suggestions become the property of Microsoft Corporation.

## Programming Background

1. Years of programming experience:  
All languages \_\_\_\_\_ C \_\_\_\_\_ C++ \_\_\_\_\_
2. Which programming language do you use for most of your software development tasks?  
\_\_\_ C \_\_\_ C++ \_\_\_ Other \_\_\_\_\_
3. How long have you used this product? \_\_\_\_\_ months

## Presenting Visual C++

4. Did you read *Presenting Visual C++*? \_\_\_ Yes \_\_\_ No
5. Did it give you a useful overview of the product?  
\_\_\_ Yes \_\_\_ No
6. Did the roadmaps help orient you to the documentation?  
\_\_\_ Yes \_\_\_ No

## Help

7. How often do you use help? (Check one.)  
\_\_\_ Never \_\_\_ Occasionally \_\_\_ Often
8. I would use help more if it were easier to use.  
\_\_\_ Yes \_\_\_ No
9. Rank the following methods for accessing help from 1 (use most often) to 5 (use least often):  
\_\_\_\_ Highlight a language element in a source file, then press F1.  
\_\_\_\_ Open a help file from the help menu.  
\_\_\_\_ Open a help file from the product group.  
\_\_\_\_ Press F1 in a dialog box.  
\_\_\_\_ Other \_\_\_\_\_
10. May we call you about help?  
\_\_\_ Yes \_\_\_ No Phone number \_\_\_\_\_

## Indexes

11. Do the individual book indexes help you find the information you need?  
\_\_\_ Always \_\_\_ Most of the time \_\_\_ Sometimes \_\_\_ Seldom
12. Does the comprehensive index help you find the information you need?  
\_\_\_ Always \_\_\_ Most of the time \_\_\_ Sometimes \_\_\_ Seldom
13. When more than one book is contained in one cover, indicate which option would be more helpful:  
\_\_\_ One index for the entire volume  
\_\_\_ One index for each document in the volume

## Visual Workbench User's Guide

14. Does the book provide enough information for you to use Visual Workbench effectively? \_\_\_ Yes \_\_\_ No—please explain: \_\_\_\_\_

15. Indicate which topics we need to improve:  
\_\_\_ Editor \_\_\_ Browser \_\_\_ Debugger  
\_\_\_ Projects and Build Options \_\_\_ Environment Options  
\_\_\_ Other \_\_\_\_\_

16. Do you use the Fast Track tables in Chapter 5?  
\_\_\_ Yes \_\_\_ No—please explain: \_\_\_\_\_

17. Indicate which areas we need to improve:  
\_\_\_ Organization \_\_\_ Procedures \_\_\_ Concepts  
\_\_\_ Reference material \_\_\_ Examples \_\_\_ Illustrations  
\_\_\_ Other \_\_\_\_\_

## App Studio User's Guide

18. Does the book help you complete your resource editing tasks?  
\_\_\_ Yes \_\_\_ No—please explain: \_\_\_\_\_

19. Indicate which areas we need to improve:  
\_\_\_ Organization \_\_\_ Procedures \_\_\_ Concepts  
\_\_\_ Reference material \_\_\_ Examples \_\_\_ Illustrations  
\_\_\_ Other \_\_\_\_\_

20. Have you used the App Studio Reference in Chapter 10?  
\_\_\_ Yes \_\_\_ No

21. Rank the usefulness of the following App Studio Reference topics from 1 (use most often) to 4 (use least often):  
\_\_\_\_ Task Reference \_\_\_\_ Menu Reference  
\_\_\_\_ Toolbar and Palette Reference \_\_\_\_ Key Reference

## Class Library User's Guide

22. Did you perform the steps in the Scribble tutorial?  
\_\_\_ Yes \_\_\_ No Your reaction: \_\_\_\_\_

23. Indicate which topics in the tutorial we need to improve:  
\_\_\_ Documents \_\_\_ Views \_\_\_ App Studio \_\_\_ Printing  
\_\_\_ Commands \_\_\_ Dialogs \_\_\_ Scrolling/Splitting  
\_\_\_ Help \_\_\_ Other \_\_\_\_\_

24. Have you used VBX controls in your programs?  
\_\_\_ Yes \_\_\_ No

## Overall Evaluation

25. Does the entire document set help you find the information you need?  
\_\_\_ Always \_\_\_ Most of the time \_\_\_ Sometimes \_\_\_ Seldom
26. Does the help system help you find the information you need?  
\_\_\_ Always \_\_\_ Most of the time \_\_\_ Sometimes \_\_\_ Seldom

Name \_\_\_\_\_

Address \_\_\_\_\_

City/State/Zip \_\_\_\_\_

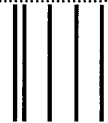
Home Phone (\_\_\_\_) \_\_\_\_\_

Work Phone (\_\_\_\_) \_\_\_\_\_

May we contact you for additional information about your comments?  Yes  No

*Additional comments:*

Fold.....



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

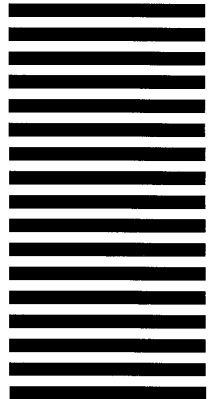
**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 108 REDMOND WA

POSTAGE WILL BE PAID BY ADDRESSEE

**Microsoft®**

Microsoft Corporation  
Languages—Visual C++™  
One Microsoft Way  
Redmond WA 98052-9953



04107-125-2529015





\* 6 1 1 5 3 \*



Recyclable

**Microsoft®**