

CodeWarrior® IDE User Guide

Windows® | Macintosh® | Solaris™ | Linux

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 11/19/99 mds



Metrowerks CodeWarrior copyright ©1993–1999 by Metrowerks, Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks, Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, PowerPlant, and Metrowerks University are registered trademarks of Metrowerks Inc. CodeWarrior Constructor, Geekware, PowerParts, and Discover Programming are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT IN THE CD BOOKLET.

How to Contact Metrowerks:

U.S.A. and International	Metrowerks Corporation 9801 Metric Blvd., Suite #100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite #300 Ville St-Laurent, QC Canada H4L 5G6
World Wide Web	http://www.metrowerks.com
Registration Information	http://www.metrowerks.com/register mailto:register@metrowerks.com
Desktop Technical Support	http://www.metrowerks.com/support/desktop/ mailto:cw_support@metrowerks.com
Embedded Technical Support	http://www.metrowerks.com/support/embedded/ mailto:cw_emb_support@metrowerks.com
Sales, Marketing, & Licensing	mailto:sales@metrowerks.com
Ordering	Voice: (800) 377-5416 Fax: (512) 873-4901
Intl. Sales, Mkt & Licensing	mailto:intlsls@metrowerks.com
International Ordering	Voice: +1 512 873 4724 Fax: +1 512 873 4901

Table of Contents



1 Introduction	17
About the CodeWarrior IDE	17
IDE User Guide Overview	19
Read the Release Notes!	20
Manual Conventions	20
Typographical Conventions	20
Host Conventions	21
Figure Conventions.	22
Keyboard Conventions	22
CodeWarrior Year 2000 Compliance.	24
New in This Release.	24
Rapid Application Development Tools	24
New Command	25
IDE Customization	25
Key Binding Improvements	25
XML Project Import/Export	26
Debugger Auto-Variables View.	26
User-Defined Trees in Access Paths	26
Project Window Improvements.	26
Where To Go From Here	26
QuickStart and Tutorials	27
Targeting Documentation	27
 2 Getting Started	 29
System Requirements	29
Windows	30
Mac OS	30
Solaris	31
Linux.	31
CodeWarrior IDE Installation.	31
Programming Concepts	31
Creating Input Files	32
Generating the Software.	34
Debugging and Refining	34
An Introduction to the CodeWarrior IDE.	35

Table of Contents

Projects and Targets	35
Source Code Editing and Browsing	36
Compiling and Linking	37
Project Debugging	37
Rapid Application Development	37
Version Control	38
Scripting	38
Customizing the IDE	38
Working with Third-Party Tools	39
3 Working with Projects	41
Guided Tour of the Project Window	42
Navigating the Project Window	43
Project Window Toolbar.	43
Files View	44
Design View	50
Link Order View	51
Targets View.	51
Creating a New Project	52
Types of Project Files	52
Choosing a Project Stationery File	53
Naming Your New Project.	55
Using the New Project dialog box	58
Modifying Your New Project.	59
Building Your New Project	59
Working with Project Stationery	60
About Project Stationery	60
About the Project Stationery Folder.	61
Creating Your Own Project Stationery.	61
Importing Makefiles into Projects (Windows).	65
Using the Makefile Importer Wizard	65
Makefile Importer Options	66
Opening an Existing Project	68
Using the Open Command	69
Using the Open Recent Command	69
Using the Project Window to Open Subprojects.	69
Opening Project Files Created on Other Host Platforms	70

Opening Project Files from Earlier IDE Versions	71
Saving a Project	72
Items Saved with Your Project	73
Saving a Copy of Your Project	73
Closing a Project	74
Choosing a Default Project	74
Managing Files in a Project	75
Expanding and Collapsing Groups	75
Selecting Files and Groups	77
Adding Files	78
Moving Files and Groups	86
Creating Groups	87
Removing Files and Groups	87
Renaming Groups	88
Touching and Untouching Files	89
Working with Complex Projects	91
What is a Build Target?	91
What is a Subproject?	92
What is a Design?	93
Strategy for Creating Complex Projects	94
Creating a New Build Target	94
Changing a Build Target Name	95
Changing the Build Target Settings	96
Setting the Current Build Target	96
Creating Build Target Dependencies	97
Assigning Files to Build Targets	98
Creating Subprojects Within Projects	99
Examining Project Information	100
Moving a Project	102
Importing and Exporting a Project	103
Controlling Debugging in a Project	104
Activating Debugging for a Project	104
Activating Debugging for a File	104
Adding Preprocessor Symbols to a Project	106
4 Working with Files	109
Creating a New File	109

Table of Contents

Opening an Existing File	110
Opening Files from the File Menu	110
Opening Files from the Project Window	112
Opening Files from an Editor Window	115
Opening a Related File	116
Saving a File	117
Closing a File	123
Closing One File	123
Closing All Files	125
Printing a File	125
Setting Print Options	125
Printing a Window	126
Reverting to a Previously-Saved File	127
Comparing and Merging Files & Folders.	127
File Comparison and Merge Overview	128
Choosing Files To Compare	130
Examining and Applying Differences	131
Choosing Folders To Compare	132

5 Editing Source Code **137**

Guided Tour of the Editor Window	137
Text Editing Area.	139
Interface Pop-Up Menu	139
Routine Pop-Up Menu	140
Marker Pop-Up Menu	141
Options Pop-Up Menu	142
VCS Pop-Up Menu	143
File Path Caption.	143
Dirty File Marker.	143
Pane Splitter Controls.	144
Line Number Button	144
Toolbar Disclosure Button	144
Path Pop-Up Menu (Mac OS)	145
Editor Window Configuration	145
Setting Text Size and Font	145
Seeing Window Controls	146
Splitting the Window into Panes	147

Saving Editor Window Settings	148
Basic Text Editing	149
Basic Editor Window Navigation	149
Adding Text	151
Deleting Text	151
Selecting Text	152
Moving Text (Drag and Drop)	154
Using Cut, Copy, Paste, and Clear	154
Balancing Punctuation	154
Shifting Text Left and Right	155
Undoing Changes	156
Controlling Color	157
Navigating the Text	157
Finding a Routine	158
Adding, Removing, and Selecting a Marker	158
Opening a Related File	160
Going to a Particular Line	161
Using Go Back and Go Forward	161
Configuring Editor Commands	162
Opening the Directory of a File (Mac OS)	162
Online References	162
Finding Symbol Definitions	163
WinHelp (Windows)	164
QuickHelp (Mac OS)	164
QuickView (Mac OS)	165
THINK Reference (Mac OS)	166
Inserting Routine (Reference) Templates (Mac OS)	166
6 Searching and Replacing Text	169
Guided Tour of the Find Dialog Box	169
Find and Replace Section	170
Multi-File Search Section	175
Searching for Selected Text	180
Finding and Replacing Text in a Single File	182
Finding Search Text	182
Controlling Search Range in a Single File	184
Controlling Search Parameters	184

Table of Contents

Searching with Special Characters	185
Replacing Found Text.	185
Using Batch Searches	187
Finding and Replacing Text in Multiple Files	189
Activating Multi-File Search	189
Choosing Files to be Searched	190
Saving a File Set	194
Removing a File Set.	195
Controlling Search Range in Multiple Files.	196
Using Regular Expressions (grep)	198
Special Operators	198
Using Regular Expressions	200

7 Browsing Source Code **207**

Activating the Browser	207
Understanding the Browser Strategy	208
Contents View	209
Browser View	210
Hierarchy View	211
Guided Tour of the Browser	212
Browser Contextual Menu.	213
Contents Window	215
Browser Window.	216
Multi-Class Hierarchy Window	224
Single-Class Hierarchy Window	226
Symbol Window	227
Browser Menu	229
Browser Wizards	230
New Class Wizard	231
New Member Function Wizard.	237
New Data Member Wizard	241
Using the Browser	244
Setting Browser Options	245
Identifying Symbols in the Browser Database	245
Navigating Code in the Browser	246
Browsing Across Subprojects.	247
Completing Symbols	247

Opening a Source File	248
Seeing a Declaration	249
Seeing a Routine Definition	249
Editing Code in the Browser	250
Analyzing Inheritance	250
Finding Functions That Are Overrides	250
Viewing MFC and PowerPlant Classes	251
Saving a Default Browser	251
8 Configuring IDE Options	253
Preferences Guided Tour	253
Preference Panels	254
Dialog Box Buttons	254
Choosing Preferences	256
General Preferences	256
Editor Preferences	272
Debugger Preferences	283
Customizing the IDE	296
Dialog Box Buttons	296
Customizing Commands	298
Customizing Key Bindings	304
Customizing Toolbars	313
9 Configuring Target Options	321
Target Settings Guided Tour	321
Settings Panels	322
Dialog Box Buttons	323
Choosing Target Settings	324
Target Configurations	326
Code Generation Configurations	352
Editor Configurations	356
Debugger Configurations	357
10 Compiling and Linking	361
Choosing a Compiler	362
Understanding Plug-in Compilers	362
Setting a File Extension	362
Compiling and Linking a Project	363

Table of Contents

Compiling Files	364
Setting Link Order	365
Updating a Project	366
Making a Project	366
Enabling Debugging	367
Running a Project	367
Debugging a Project	368
Generating a Link Map	369
Synchronizing Modification Dates	369
Removing Object Code	369
Advanced Compile Options	371
Using Precompiled or Preprocessed Headers	371
Creating Precompiled Headers	372
Defining Symbols For C/C++	375
Defining Symbols For Pascal	377
Preprocessing Source Code	378
Disassembling Source Code	379
Guided Tour of the Message Window	379
Using the Message Window	382
Seeing Errors and Warnings	383
Stepping Through Messages	384
Correcting Compiler Errors and Warnings	385
Correcting Linker Errors	386
Correcting Pascal Circular References	389
Saving and Printing the Message Window	390
Locating Errors in Modified Files	391
Special Library Options (Mac OS)	391
Import Weak	392
Initialize Before	393
Merge Into Output	393

11 Debugging Source Code 395

Symbolics Files	396
Preparing for Debugging	397
Setting Up a Build Target for Debugging	397
Setting Up a File for Debugging	398
Generating Symbolics Information	399

Using the Debugger	400
Guided Tour of the Debugger	401
Stack Crawl Window	402
Source Code Browser Window	410
Symbol Hint	415
Debugger Contextual Menu	416
Processes Window	417
Expressions Window	420
Global Variables Window	421
Breakpoints Window	423
Watchpoints Window	424
Register Details Window	424
Register Windows	430
Log Window	434
Variable Window	435
Array Window	436
Memory Window	437
Basic Debugging	439
Starting Up	440
Running, Stepping, and Stopping Code	443
Navigating Code	452
Breakpoints	457
Watchpoints	465
Viewing and Changing Data	469
Editing Source Code	481
Expressions	482
How Expressions are Interpreted	482
Using Expressions	485
Example Expressions	487
Expression Syntax	489
Troubleshooting	493
General Problems	494
Problems Launching the Debugger	494
Problems Running/Crashing the Debugger	496
Problems with Breakpoints	496
Problems with Variables	498
Problems with Source Files	502

Debugger Error Messages	504
12 RAD Designs and Layouts	509
About RAD	509
CodeWarrior RAD Tools	510
Class Authoring	510
Component Model	510
Creating RAD Projects	511
RAD Wizards	515
Java Applet Wizard	516
Java Application Wizard	523
Java Bean Wizard	527
Working with Designs	531
Adding Designs to a Project	531
Design View in the Project Window	533
Designs in the Targets View	533
Working with Layouts	534
13 RAD Layout Editing	537
Layout Editor	537
Creating a Layout	537
Modifying a Layout	538
Layout Wizards	544
Java Frame Wizard	545
Component Palette	546
Component Palette Toolbar	548
Catalog Pop-up Menu	548
Component Tools	548
Component Catalog window	550
Component Catalog Toolbar	551
Catalog Pane	552
Component Pane	553
Component Catalog Contextual Menu	557
Creating Component Catalogs	558
Object Inspector	562
Object Pop-up Menu	563
Properties Tab	564


Events Tab	565
Object Inspector Contextual Menu	566
14 RAD Components	569
Java AWT Components	569
Java Swing Components.	570
Component Editors	572
Menu Editor.	573
Pop-up Menu Editor	577
15 RAD Browsing	579
Browser Window RAD Features	579
Tab Control	579
Properties View	581
Methods View	582
Events View	583
RAD Windows	584
New Property Window	584
New Method Window	586
New Event Set Window	589
New Event Set wizard	591
New Event Window	597
16 Using the CodeWarrior IDE with Version Control Systems	601
Using Version Control Systems	601
Commercially Available VCS Plug-ins.	601
Activating VCS Operations	603
Installing VCS Software	603
Activating VCS Software	603
Accessing VCS Features	606
VCS Menu	606
Version Control Login Window	609
VCS Pop-up	609
Project Window	612
VCS Messages Window	613
Common VCS Operations	613
Getting a File's Status	613
Modifying a Checked In File.	614

Dealing with 'ckid' Resources (Mac OS)	615
Other Operations.	615
17 CodeWarrior Reference	617
IDE Menu Reference	617
File Menu	619
Edit Menu.	623
Search Menu.	626
Project Menu	632
Debug Menu	638
Data Menu	642
Browser Menu	644
Window Menu.	645
Layout Menu	650
Version Control System (VCS) Menu	651
Help Menu	652
Java Exceptions Submenu	653
Toolbar Submenu	653
Apple Menu (Mac OS)	655
Tools Menu (Mac OS).	655
Scripts Menu (Mac OS)	657
Editor Extensions Menu (Mac OS)	658
Info Menu (Solaris).	659
IDE Default Key Bindings	660
Mac OS, Solaris, and Linux Modifier Key Legend.	661
File Menu	662
Edit Menu.	663
Search Menu.	664
Project Menu	666
Debug Menu	667
Data Menu	668
Browser Menu	670
Miscellaneous	670
Editor Commands	671
Catalog	674
Layout Menu	674
Window Menu.	675

A Solaris and Linux Utilities	679
Copy Files Accessory 679
Source File/Directory Selection 680
Currently Selected Files/Directories 681
Destination Directory Selection. 682
File Systems Facility. 682
The File Systems Dialog Box 683
The File System Options Dialog Box 684
Keyboard Preferences Dialog Box 687
 B Perl Scripts	 691
Installing the Perl Software Plug-ins. 691
Windows 691
Mac OS 692
Configuring the Perl Target Settings Panel 692
Perl Scripting 693
Special Considerations 695
StdIn Usage 695
Avoiding Link Errors 695
 Index	 697

Table of Contents

Introduction



This manual describes the CodeWarrior Integrated Development Environment (IDE) in detail. You can apply the adept capabilities of the IDE to develop software on different operating systems using programming languages like C, C++, and Java™.

NOTE On occasion a CodeWarrior product ships with an earlier version of the IDE than reflected in this user guide. In that case, your IDE will not have the new features described in this manual. You can identify new features by referring to [“New in This Release” on page 24](#). For more information on available patches and updates for CodeWarrior tools, visit the following web address:

<http://www.metrowerks.com/>

This chapter introduces the CodeWarrior IDE. The sections in this chapter are:

- [About the CodeWarrior IDE](#)
- [IDE User Guide Overview](#)
- [Read the Release Notes!](#)
- [Manual Conventions](#)
- [CodeWarrior Year 2000 Compliance](#)
- [New in This Release](#)
- [Where To Go From Here](#)

About the CodeWarrior IDE

The CodeWarrior IDE is an application that provides a simple, versatile graphical user interface and tools for developing computer software. Using the IDE, you can develop a program, plug-in,

library, or other executable code to run on a wide variety of computer systems using different programming languages.

The IDE permits a software developer to quickly assemble source code files (for example, a file written in the C++ computer language), resource files, library files, other files, and configuration settings into a “project,” without writing a complicated build script (or “makefile”). Source code files may be added or deleted from a project using simple mouse and keyboard operations instead of tediously editing a build script.

A single project allows you to create and manage several configurations of settings for use on various computer platforms. The platform on which you run CodeWarrior is called the “host.” From that host, you use the IDE to develop code to “target” various platforms and operating systems. There are two distinct meanings of “target” in CodeWarrior terminology:

- **Platform target**—the operating system, processor, or microcontroller for which you write code. For example, if you choose to write code for a particular processor running a desktop operating system, you are creating the code for a platform target.
- **Build target**—the collection of settings and files that determines the contents of your code, as well as the process in which this code is compiled and linked into the final output.

CodeWarrior allows you to specify multiple build targets for a particular platform target. For example, you can compile a debugging version (build target) and an optimized version (build target) of your program for Windows (platform target). The build targets can share files in the same project while using their own settings. After debugging the program, generating a final version is as simple as changing the project’s build target and using a single [Make](#) command.

The CodeWarrior IDE includes compilers, linkers, a debugger, a source code browser, an editor, and rapid application development tools. You can edit, navigate, examine, compile, link, and debug code until you have a running application. Options for code generation, debugging, and navigation of your project are all configurable in the IDE.

The CodeWarrior IDE and its tools have everything you need to develop code!

IDE User Guide Overview

There are several chapters in this User Guide to explain how to use the IDE. Each chapter begins with an overview of topics. The chapters are:

- [Introduction](#)—(this chapter) an overview of the CodeWarrior IDE and documentation
- [Getting Started](#)—system requirements and installation
- [Working with Projects](#)—creating, configuring, and managing projects
- [Working with Files](#)—opening, saving, backing up, comparing, and printing files
- [Editing Source Code](#)—editing and navigating text and source code
- [Searching and Replacing Text](#)—finding and replacing text in single files and multiple files
- [Browsing Source Code](#)—analyzing and navigating a project from various views
- [Configuring IDE Options](#)—customizing the IDE Preferences window and using toolbars
- [Configuring Target Options](#)—setting up and customizing a project and its build targets
- [Compiling and Linking](#)—compiling, linking, running, updating, preprocessing, and precompiling a project's build target and its files
- [Debugging Source Code](#)—using the CodeWarrior debugger to examine your code
- [RAD Designs and Layouts](#)—understanding rapid application development (RAD) and its implementation in CodeWarrior
- [RAD Layout Editing](#)—using CodeWarrior's RAD tools to create and manage project designs and layouts
- [RAD Components](#)—a description of the components provided with the RAD tools

- [RAD Browsing](#)—using the Class Browser and Object Inspector to examine RAD designs
- [Using the CodeWarrior IDE with Version Control Systems](#)—using revision control systems with the CodeWarrior IDE
- [CodeWarrior Reference](#)—convenient reference information for menu commands and default key bindings

Read the Release Notes!

Please read the release notes. They contain important information about new features, bug fixes, and incompatibilities that may not have made it into the documentation due to release deadlines. You can find the release notes on the CodeWarrior CDs in the Release Notes folder.

Manual Conventions

The following sections describe the different conventions used in the CodeWarrior IDE manual:

- [Typographical Conventions](#)
- [Host Conventions](#)
- [Figure Conventions](#)
- [Keyboard Conventions](#)

Typographical Conventions

The IDE manual uses some style conventions to make it easier to read and find specific information:

Notes, warnings, tips, and beginner's hints

An advisory statement or **NOTE** may restate an important fact, or call your attention to a fact which may not be obvious.

A **WARNING** given in the text may call attention to something such as an operation that, if performed, could be irreversible, or flag a possible error that may occur.

A **TIP** can help you become more productive with the CodeWarrior IDE. Impress your friends with your knowledge of little-known facts that can only be learned by actually reading the fabulous manual!

A **For Beginners** note may help you better understand the terminology or concepts if you are new to programming.

Typeface conventions

If you see some text that appears in a different typeface (as the word `different` does in this sentence), you are reading file or folder names, source code, keyboard input, or programming items.

Text **formatted like this** means that the text refers to an item on the screen, such as a **menu command** or **control** in a dialog box.

When a menu command is part of a submenu in CodeWarrior's menu hierarchy, all of the intermediate commands are listed and separated with a > character. For example, the [Reset Window Toolbar](#) menu command is part of the Toolbar submenu, which is in the Window menu. When you read **Window > Toolbar > Reset Window Toolbar**, follow the hierarchy to select the command. In this case, you would highlight **Window** in CodeWarrior's menu bar, then highlight **Toolbar**, and finally choose the **Reset Window Toolbar** command.

If you are using an online viewing application that supports hypertext navigation, such as Adobe Acrobat, you can click on underlined and colored text to view another topic or related information. For example, clicking the text ["IDE User Guide Overview"](#) in Adobe Acrobat takes you to a section that gives you an overview of the entire IDE User Guide.

Host Conventions

CodeWarrior runs on the host platforms and operating systems listed below. Throughout this manual, a generic platform identifier is used to identify the host platform, regardless of operating system.

The specific versions of the operating systems that host CodeWarrior are:

- **Windows**—desktop versions of the Windows operating system that are Win32 compliant, such as Windows 95, Windows 98, Windows 2000, and Windows NT.
- **Mac OS**—desktop versions of the Mac OS, System 7.1 or later.
- **Solaris**—Solaris version 2.5.1 or later.
- **Linux**—Red Hat Linux or Souse Linux.

Figure Conventions

The visual interfaces of the platforms that host CodeWarrior are nearly identical in all significant respects. When discussing a particular interface element, such as a dialog box or window, the screenshot of that element can come from a variety of possible hosts. You should have no difficulty understanding the picture, even if you are using CodeWarrior on a different host than the one shown in the screenshot.

However, there are occasions when dialog boxes or windows are unique to a particular host. For example, a particular dialog box may appear dramatically different on a Windows host and on a Mac OS host. In that case, a screenshot from each unique host will be clearly identified so that you can see how CodeWarrior works on your preferred host.

Keyboard Conventions

Most of CodeWarrior's default keyboard shortcuts are very similar across multiple platforms. However, some keys have different names, depending on the host. For example, a typical keyboard for a Windows machine has an Alt key, but that same key is called the Option key on a typical keyboard for a Mac OS computer.

To handle these kinds of situations, CodeWarrior documentation identifies and uses the following paired terms in the text:

- **Enter/Return**—the “carriage return” or “end of line” key. This is not the numeric keypad's Enter key, although in almost all cases that works the same way.
- **Backspace/Delete**—the Windows Backspace key and the Mac OS Delete key. In text editing, pressing Backspace/Delete erases the character before the insertion point. (Note that

Backspace/Delete is not the same as Delete/Del, the “forward delete” key.)

- Ctrl/Command/Meta—the Windows Ctrl (control) key, the Mac OS Command key (⌘), and the Solaris Meta key.
- Alt/Option—the Windows Alt key and the Mac OS Option key.

For example, you may encounter instructions such as “Press Enter/Return to proceed,” or “Alt/Option click the Function pop-up menu to see the functions in alphabetical order.” The first instruction tells you to press Enter on a Windows host or Return on a Mac OS host in order to proceed. The same idea applies to the second instruction: Alt-click on a Windows host or Option-click on a Mac OS host. In general, use the appropriate key as it is labeled on your keyboard.

Some combinations of key strokes require multiple modifier keys. In those cases, key combinations are shown connected with hyphens. For example, if you read “Shift-Alt/Option-Enter/Return,” you would press the Shift, Alt, and Enter keys on a Windows host and the Shift, Option, and Return keys on a Mac OS host.

Sometimes the cross-platform variation in keyboard shortcuts is more complex. In those cases, you will see more detailed instructions on how to use a keyboard shortcut for your host platform. In all cases, the host and shortcut will be clearly identified.

Special note for Solaris users

The Solaris-hosted CodeWarrior IDE uses the same modifier key names as used for the Mac OS (Shift, Command, Option, and Control). Likewise, the Key Bindings preference panel uses Mac OS symbols to represent modifier keys. [Table 17.1 on page 662](#) shows the default modifier key mappings and the symbols used to represent them. On Solaris machines, modifier keys can be mapped to any key on the keyboard. See [“Keyboard Preferences Dialog Box” on page 687](#) for more information on changing the default modifier key mappings. When reading this manual, you will need to keep in mind your modifier key mappings.

CodeWarrior Year 2000 Compliance

The Products provided by Metrowerks under the License agreement process dates only to the extent that the Products use date data provided by the host or target operating system for date representations used in internal processes, such as file modifications. Any Year 2000 Compliance issues resulting from the operation of the Products are therefore necessarily subject to the Year 2000 Compliance of the relevant host or target operating system. Metrowerks directs you to the relevant statements of Microsoft Corporation, Sun Microsystems, Inc., Apple Computer, Inc., and other host or target operating systems relating to the Year 2000 Compliance of their operating systems. Except as expressly described above, the Products, in themselves, do not process date data and therefore do not implicate Year 2000 Compliance issues.

For additional information, visit the following website:

<http://www.metrowerks.com/about/y2k.html>

New in This Release

The CodeWarrior IDE is now at version 4.0. Among its new features, the following are the most notable:

- [Rapid Application Development Tools](#)
- [New Command](#)
- [IDE Customization](#)
- [Key Binding Improvements](#)
- [XML Project Import/Export](#)
- [Debugger Auto-Variables View](#)
- [User-Defined Trees in Access Paths](#)
- [Project Window Improvements](#)

Rapid Application Development Tools

This release of CodeWarrior introduces rapid application development (RAD) tools for Java. You can use these RAD tools to quickly create Java applets, applications, and frameworks. The new

tools include a layout editor, RAD components, a catalog window, and an object inspector. CodeWarrior wizards guide you through the process of creating RAD “designs.” A new Design tab in the project window allows you to manage various designs for a given project.

For more information, see these chapters: [“RAD Designs and Layouts” on page 509](#), [“RAD Layout Editing” on page 537](#), [“RAD Components” on page 569](#), and [“RAD Browsing” on page 579](#).

New Command

The File menu contains an updated **New** command. Using this command, you can create files, projects, RAD designs, and objects. The **New** command also lets you access CodeWarrior’s Makefile Importer and RAD wizards.

For more information, refer to [“Creating a New Project” on page 52](#), [“Importing Makefiles into Projects \(Windows\)” on page 65](#), [“Creating a New File” on page 109](#), and [“Creating RAD Projects” on page 511](#).

IDE Customization

CodeWarrior can be customized to suit your personal programming needs. You can create and modify menu commands, toolbar buttons, key bindings, and contextual menus. The IDE lets you associate these new items with a command line (Windows) or a script (Mac OS). The customized menu commands have access to IDE information, such as the current editor selection, the frontmost window, the current project, and the output file of the current project.

Refer to [“Customizing the IDE” on page 296](#) for more information.

Key Binding Improvements

New keypad key bindings compliment the existing keyboard key bindings. Imported and exported key binding files are now cross-platform. The user interface for modifying key bindings has been improved.

For more information, refer to [“Customizing Key Bindings” on page 304.](#)

XML Project Import/Export

CodeWarrior’s project manager now lets you import and export a description of your project in extensible markup language (XML).

To learn more, see [“Importing and Exporting a Project” on page 103.](#)

Debugger Auto-Variables View

The debugger features a new Auto-variables view. This view automatically displays the variables referenced from a particular line of source code.

For more information, read [“Symbol Hint” on page 415.](#)

User-Defined Trees in Access Paths

In addition to the current {Project}, {Compiler}, and {System} access path definitions, you can now define your own source trees. Your source trees can be specified at the global- or target-level of the project. They can be defined via an absolute path, environment variable (Windows and Solaris), or a registry key (Windows). The source trees can then be used to form access paths.

To learn how to create your own source trees and corresponding access paths, see [“Source Trees” on page 266.](#)

Project Window Improvements

In the improved project window, you can sort file lists according to any of the columns displayed in that list.

For more information, see [“Sorting items” on page 47.](#)

Where To Go From Here

If you are trying to get started quickly with a new platform or if you are new to CodeWarrior, see [“QuickStart and Tutorials” on page 27.](#)

If you are an experienced CodeWarrior IDE user, review [“New in This Release” on page 24](#) for an overview of new features.

To get started quickly with a new target operating system, see [“Targeting Documentation” on page 27](#).

When you are ready to debug your code, be sure to read [“Debugging Source Code” on page 395](#).

The following sections will give you a better idea of how to proceed:

- [QuickStart and Tutorials](#)
- [Targeting Documentation](#)

QuickStart and Tutorials

You will find all the manuals mentioned in this section in the CodeWarrior Documentation folder on the CodeWarrior CD. For some products, the manuals will be on the CodeWarrior Reference CD.

If you are new to the CodeWarrior IDE, check out the following resources:

- The *CodeWarrior QuickStart Guide* provides an overview of CodeWarrior and points to the references available to you. The *QuickStart Guide* is on your CD, so you can use it regardless of whether you purchased any printed documentation.
- [“An Introduction to the CodeWarrior IDE” on page 35](#) provides a quick overview of the CodeWarrior IDE user interface.
- The CW Core Tutorials folder on the CodeWarrior CD contains some sample projects that will help you quickly become productive with the CodeWarrior IDE.
- Instructions for using CodeWarrior’s online documentation viewers are located in the CodeWarrior Documentation folder on the CodeWarrior CD.

Targeting Documentation

This manual describes how to use the cross-platform features of the IDE. However, you should read the appropriate CodeWarrior targeting documentation to learn how to develop software for a specific platform target.

[Table 1.1](#) identifies the documentation that you should consult when developing software for specific platform targets.

NOTE The CodeWarrior product you are using may not generate software for all the platform targets listed in [Table 1.1](#).

Table 1.1 Targeting Manuals for various platform targets

Platform Target	Targeting Manual
Java Virtual Machine	<i>Targeting the Java VM</i>
Mac OS	<i>Targeting Mac OS</i>
MIPS	<i>Targeting MIPS</i>
Nucleus	<i>Targeting Nucleus</i>
Palm OS	<i>Targeting Palm OS</i>
PlayStation	<i>Targeting PlayStation OS</i>
PowerPC Embedded	<i>Targeting PowerPC for Embedded Systems</i>
Solaris	<i>Targeting Solaris</i>
Win32/x86	<i>Targeting Win32</i>

Getting Started



This chapter helps you get started using the CodeWarrior IDE (Integrated Development Environment). You can find information about system requirements, installing IDE software, and an introduction to the IDE's user interface and capabilities.

The sections in this chapter are:

- [System Requirements](#)
- [CodeWarrior IDE Installation](#)
- [Programming Concepts](#)
- [An Introduction to the CodeWarrior IDE](#)

NOTE This manual describes how to use the cross-platform features of the IDE. However, you should read additional CodeWarrior documentation to learn how to develop software for a particular platform target. For information on developing software for specific platform targets, see [“Targeting Documentation” on page 27](#).

TIP For a quick look at the IDE's features, see [“An Introduction to the CodeWarrior IDE” on page 35](#). The tour gives you your first glimpse of the CodeWarrior IDE.

System Requirements

The system requirements to operate the CodeWarrior IDE are specified by platform. These platforms include:

- [Windows](#)
- [Mac OS](#)
- [Solaris](#)

- [Linux](#)

Windows

The Windows-hosted version of the CodeWarrior IDE requires the following:

- a 486DX processor or greater (an Intel Pentium™-class processor or AMD K6-class microprocessor is recommended)
- at least 32 megabytes of RAM
- approximately 90 megabytes of free hard disk space for a minimal installation
- approximately 450 megabytes of free hard disk space for a full installation
- Microsoft Windows operating system (Windows 95, Windows 98, or Windows NT 4.0 service pack 3)
- a CD-ROM drive to install the software.

Mac OS

The Mac OS-hosted version of the CodeWarrior IDE requires the following:

- a Motorola MC68040 processor or greater (a PowerPC 601 or greater processor is recommended)
- at least 32 megabytes of RAM
- approximately 120 megabytes of free hard disk space for a minimal installation
- approximately 400 megabytes of free hard disk space for a full installation
- Mac OS 7.6.1 or later
- a CD-ROM drive to install the software

NOTE When running on 68k machines, rapid application development (RAD) is not supported, and CFM 68K Enabler 4.0 is required (provided by the CodeWarrior installer).

Solaris

The Solaris-hosted version of the CodeWarrior IDE requires the following:

- a Sun SparcStation or Sparc-based machine
- at least 64 megabytes of RAM
- approximately 80 megabytes of free hard disk space
- Solaris 2.5.1 or later, an X11-R5 display server, and Motif 1.2 or later (CDE recommended)
- a CD-ROM drive to install the software.

Linux

The Linux-hosted version of the CodeWarrior IDE requires the following:

- an Intel Pentium™-class processor
- at least 64 megabytes of RAM
- approximately 80 megabytes of free hard disk space
- Red Hat Linux 5.2 and an X11-R5 display server
- a CD-ROM drive to install the software

CodeWarrior IDE Installation

To learn how to install the CodeWarrior IDE, read the QuickStart guide. The CodeWarrior installation software places the CodeWarrior IDE, compilers, linkers, tools, and debugger on your hard disk.

Programming Concepts

If you are new to programming computers, read this section to learn about the terms used in this manual.

There are three important tasks involved in developing software:

- [Creating Input Files](#)
- [Generating the Software](#)

- [Debugging and Refining](#)

First, you create files that contain statements using a computer language such as C, C++, Java, Pascal, or assembly language. You can also create files that contain resources, which are descriptions of user interface objects such as windows, dialog boxes, and menus.

Next, you apply tools to help turn your “source” materials into an output file that executes on your target computer. These tools include compilers, linkers, other tools, and maybe even assembly-language assemblers. You use these tools to create many different types of software, depending on the tools used and the target computer. Some of the types of software you can create include applications (executables), dynamic (shared) libraries, and static libraries.

After you create a piece of software, you can use a debugger to examine that software. The debugger helps you determine whether the software executes as you intended. If the software does not run properly, you make changes to your source material so that the software behaves correctly. You continue to revise and debug the source material until the software works appropriately.

Creating Input Files

The types of files you create and use to make a program are source code files, resource files, interface or header files, library files, and project files.

Source code file

A source code file is a text file containing program statements written in a language such as C, C++, Java, Pascal, or assembly language.

Resource file

A resource file contains descriptions of user interface items, such as window definitions, dialog box layouts, and text strings. A resource file may be a binary file linked into your software product, or it may be a text file translated by a special resource compiler before being linked. Placing resources in separate files makes it convenient to

tweak and customize those resources without having to recompile other parts of a program.

Interface or header file

An interface file, also called a header or include file, is a text file referred to by source code files. Typically, interface files give access to objects, variables, data structures, routines, and other items in libraries or source code files.

Library file

A library file contains objects, variables, routines, and other items that have already been compiled. There are two kinds of libraries: static and dynamic.

The IDE builds a static library right into a program. A static library cannot change (hence the term “static”) unless you rebuild the program. Furthermore, you cannot share static libraries with other programs.

Although a program might refer to a dynamic library, the IDE does not build it into a program. Instead, the program hooks up with the library dynamically, that is, while the program is actually running. Often, more than one program shares the same dynamic library. Also, a dynamic library can be replaced with a newer library without affecting the programs that refer to it, as long as the newer library provides the same operations as its predecessor.

NOTE Most platform targets support static libraries, but not all platform targets support dynamic libraries. For information on creating and using libraries for a specific platform target, see [“Targeting Documentation” on page 27](#).

Project file

A project file contains one or more build targets. Each build target contains a list of source code files, resources, interfaces, library files, and even other project files. Each build target also contains its own settings that tell the IDE how to create your software. For more information about these build targets, see [“Projects and Targets” on page 35](#).

Generating the Software

The types of tools the IDE uses to build your software include compilers, assemblers, and linkers. Using information from files and settings in a project's build target, the IDE automatically chooses the appropriate tools to create your software.

Compilers

A compiler translates a source code file, such as a C, C++, Pascal, or Java file, into binary machine code (also called object code) that will be used by linkers in a later build stage. A compiler is one of the first tools that the IDE invokes to build your program.

Assemblers

An assembler translates an assembly language source code file into object code. An assembler is really just a compiler, only it translates assembly language source code rather than high-level language source code like C, C++, or Pascal.

Linkers

A linker combines the object code in your project's build target with the object code produced by the compilers and assemblers in order to produce a piece of software.

Output file

The output file is the piece of software that the linker generates. There are many different types of software, such as applications and libraries. For in-depth information about the kinds of software you can develop for a platform target, see the CodeWarrior targeting documentation appropriate for your platform target. For information on which *Targeting* manual to read, see [“Targeting Documentation” on page 27](#).

Debugging and Refining

A debugger is a piece of software that facilitates controlled program execution and enables you to find errors in your code. With it, you can stop at points in your program's execution and see variable contents. You can also execute one line of code at a time.

All you need to do to enable the debugger is produce special data that the debugger uses to control your program. The IDE creates *debugging information* or *symbolics* files, which contain the information the debugger needs to display and control statements, variables, objects, and data types in your source code.

The IDE includes a debugger as part of the programming environment. In addition, the IDE supports third-party debuggers on some platforms. See the CodeWarrior targeting documentation associated with your platform target for additional information.

For information on using the CodeWarrior debugger, see [“Debugging Source Code” on page 395](#).

An Introduction to the CodeWarrior IDE

This section describes the concepts, tasks, and operation of the CodeWarrior IDE. The topics in this section are:

- [Projects and Targets](#)
- [Source Code Editing and Browsing](#)
- [Compiling and Linking](#)
- [Project Debugging](#)
- [Rapid Application Development](#)
- [Version Control](#)
- [Scripting](#)
- [Customizing the IDE](#)
- [Working with Third-Party Tools](#)

For information on the terms used in this section, see [“About the CodeWarrior IDE” on page 17](#) and [“Programming Concepts” on page 31](#).

Projects and Targets

The IDE uses projects and build targets to organize the files and settings used to create a program. A project is a file that contains one or more build targets. A build target is a collection of source code files, resource files, libraries, settings, even other projects, that

describe how to create a piece of software for a particular processor or operating system. Build targets within a project may share the same files, but each build target has its own settings.

The CodeWarrior IDE also has pre-configured stationery projects. Creating a new project is as easy as deciding on a platform target and a programming language and then choosing the corresponding stationery project.

You can set options to choose the platform target for which you are developing code, customize compiler optimizations and other object code details, configure source code translation, and specify the kinds of files that may be added to a build target. Depending on the platform target, you can set additional options.

For more information on working with projects and build targets, see [“Targeting Documentation” on page 27](#), [“Working with Projects” on page 41](#), and [“Configuring Target Options” on page 321](#).

Source Code Editing and Browsing

The IDE has a powerful and flexible text editor for editing source code and text files. Besides regular text-editing features, the IDE also provides drag-and-drop editing, text file manipulation in various file formats, and marker navigation in a text file.

The editor also has many practical features for programmers. Advanced programming aids include auto-indenting, syntax coloring, routine and interface pop-up menus, and automatic balancing for braces, brackets, and parentheses. The editor works closely with the class browser to make editing, viewing, and navigating among routines, data structures, variables, and objects intuitive and quick.

The [Find](#) dialog box and the [Search Menu](#) have commands and features to search and replace text in one file or in a group of files. You can search and replace normal text or regular expressions.

A file comparison and merging command displays two text files side-by-side and lets you easily compare differences between the two files. In addition, you can compare the contents of two folders to look for differences between files.

For more information on working with text and source code files, see [“Working with Files” on page 109](#), [“Editing Source Code” on page 137](#), [“Searching and Replacing Text” on page 169](#), and [“Browsing Source Code” on page 207](#).

Compiling and Linking

The IDE has commands to preprocess, precompile, compile, update, link, run, disassemble, and check syntax. The IDE automatically chooses the appropriate compilers and linkers and automatically determines which files to operate on when you issue a compile or link command. The IDE uses the settings in a project’s build targets to instruct its compilers and linkers how to process files and data. All you have to do is use one of the commands in the [Project Menu](#).

For more information on setting options, compiling, linking, and other software generation operations, see [“Configuring Target Options” on page 321](#) and [“Compiling and Linking” on page 361](#).

Project Debugging

The CodeWarrior debugger provides a seamless interaction between the programming and debugging of your source code. The debugger fully supports x86, PowerPC, 68K, and Java debugging.

After you enable the debugger, you simply **Run** the project to use debugging features. You can pause the program at any time to set breakpoints and watchpoints, view variables or memory, step into or out of routines, and perform many other debugging tasks.

NOTE Some versions of the CodeWarrior IDE do not ship with a debugger. In those cases, an external debugging application or other third-party debugger provides debugging support. For additional information, see the CodeWarrior targeting documentation for your particular platform.

Rapid Application Development

CodeWarrior includes rapid application development (RAD) tools for helping you quickly create new applications. You can use the

RAD tools to generate a graphical user interface and associate actions with that interface. CodeWarrior manages the software framework and generates basic source code automatically. In addition, you can revise the generated source code or write your own source code to handle tasks more efficiently.

For more information about CodeWarrior's RAD tools, see [“RAD Designs and Layouts” on page 509](#).

Version Control

You can configure the IDE to work with your preferred version control system (VCS). You can log onto a file server, retrieve files, store files, and do other revision control tasks with a VCS software package that supports the CodeWarrior IDE.

For information on using the IDE with VCS software, see [“VCS Pop-Up Menu” on page 143](#) and [“Using the CodeWarrior IDE with Version Control Systems” on page 601](#).

Scripting

The IDE supports scripting via Perl scripts, batch files (Windows), and AppleScript (Macintosh). When you script the IDE, you can automate repetitive, time-consuming, or complex tasks.

For more information on scripting the IDE, see [“Perl Scripts” on page 691](#), and [“Scripts Menu \(Mac OS\)” on page 657](#).

Customizing the IDE

The IDE has many user-configurable options. You can use the IDE's [Preferences](#) window to customize features, set the colors and fonts for viewing and editing source code, and assign keyboard shortcuts for commands. The IDE also has conveniently-placed, easily customizable toolbars that give quick access to commands and information by simply clicking a button.

For more information on customizing the IDE, see [“Configuring IDE Options” on page 253](#).

Working with Third-Party Tools

The IDE works with other third-party text editors, debuggers, and development tools.

For information on using CodeWarrior with third-party tools, see [“IDE Extras” on page 259](#), [“Debugging a Project” on page 368](#), and [“Using the CodeWarrior IDE with Version Control Systems” on page 601](#).

Getting Started

An Introduction to the CodeWarrior IDE

Working with Projects



This chapter introduces the CodeWarrior IDE Project window and shows how to create, configure, and work with projects.

A project contains one or more build targets. Each build target in a project contains a collection of files that the IDE uses to build an output file. Build targets within a project may share some or all of their files. Some examples of an output file include an application, static library, or dynamic library.

Each build target within a project has its own options that customize how the IDE builds the output file. There are a wide variety of options that control code optimization, browsing, debugging, compiler warnings, and much more.

Finally, build targets within a project can be configured to depend on other build targets in the project. This feature makes it possible to build software that, for example, combines the output files for different platform targets into a single output file.

This chapter discusses many of the basic tasks involving projects, such as creating projects, opening projects, adding files, and saving projects. It also describes advanced operations such as moving files in the Project window, marking files for debugging, creating nested projects and build targets, and dividing the Project window into groups of files.

The topics in this chapter are:

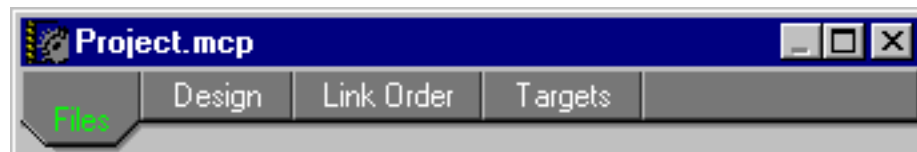
- [Guided Tour of the Project Window](#)
- [Creating a New Project](#)
- [Working with Project Stationery](#)
- [Importing Makefiles into Projects \(Windows\)](#)
- [Opening an Existing Project](#)

- [Saving a Project](#)
- [Closing a Project](#)
- [Choosing a Default Project](#)
- [Managing Files in a Project](#)
- [Working with Complex Projects](#)
- [Examining Project Information](#)
- [Moving a Project](#)
- [Importing and Exporting a Project](#)
- [Controlling Debugging in a Project](#)
- [Adding Preprocessor Symbols to a Project](#)

Guided Tour of the Project Window

The Project window shows information about the various files and build targets in a project file. The Project window uses four different views: the Files view, Design view, Link Order view (sometimes called Segments view or Overlays view), and Targets view. To choose a view, click its tab at the top of the Project window, as shown in [Figure 3.1](#).

Figure 3.1 View tabs at the top of the Project window



The Files view shows a list of all the files in a project. Items in this view may be organized into hierarchical groups that you create and arrange.

The Design view shows the rapid application development (RAD) designs in the project.

The Link Order view (sometimes called Segments view or Overlays view) shows information about how the IDE will compile or link the final output file for the project's current build target.

The Targets view shows information about the active build target, build targets associated with RAD designs, build target dependencies, and link-compatible build targets.

The topics that explain the Project window in detail are:

- [Navigating the Project Window](#)
- [Project Window Toolbar](#)
- [Files View](#)
- [Design View](#)
- [Link Order View](#)
- [Targets View](#)

To learn more about debugging information, see [“Controlling Debugging in a Project” on page 104](#).

Navigating the Project Window

To navigate the Project window, use the vertical scroll bar on the right side of the window, or the Up and Down Arrow keys on your keyboard. If the Project window contains many files, use the Home key to jump to the first file in the first segment or group, or use the End key to jump to the last file in the last segment or group.

Use the Page Up and Page Down keys to scroll one page up or one page down the Project window.

To learn about a technique for selecting files as you type, refer to [“Selection by keyboard” on page 78](#).

Project Window Toolbar

The toolbar in the Project window has buttons and other items to provide shortcuts to commands and information about the project. You can choose the items to display on the toolbar, and the order in which those items are displayed. You can even choose to hide or display the toolbar itself. To learn more about toolbars in the CodeWarrior IDE and how to configure them, refer to [“Customizing the IDE” on page 296](#).

Files View

The Project window's Files view shows the files for all build targets in the project. The files can be arranged into hierarchical groups without affecting the way the IDE handles a build target. This view also shows information about modification status, file access paths, code size, data size, current build target, debugging status, and other information.

The following topics describe the parts of the Files view:

- [File column](#)
- [Code column](#)
- [Data column](#)
- [Debug column](#)
- [Target column](#)
- [Touch column](#)
- [Sorting items](#)
- [Interface pop-up menu](#)
- [File Control pop-up menu](#)
- [Checkout Status column](#)
- [Project Checkout Status icon](#)

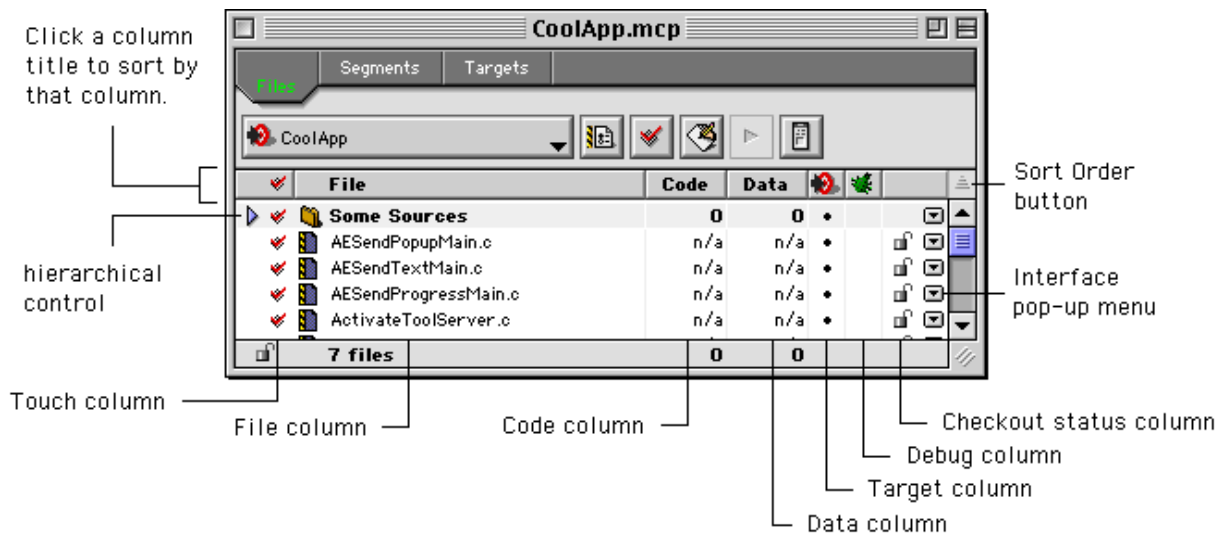
File column

The File column lists a project's files and groups in a user-configurable hierarchical view. A group may contain files and other groups.

Double-clicking a file's name in the File column will open that file. For information on opening files from the File column, refer to ["Opening Files from the Project Window" on page 112.](#)

Use the hierarchical control to display and hide the contents of groups in the File column.

Figure 3.2 Project window



To see the path along which the IDE accesses a file:

Windows Right-click the file's name in the Project window, then choose **Open in Windows Explorer** from the pop-up menu.

Mac OS Control-click the file's name in the Project window, then choose **File Path** from the pop-up menu.

Solaris Click and hold on the file's name in the Project window, then choose **File Path** from the pop-up menu.

For information on opening files from the File column, refer to ["Opening Files from the Project Window" on page 112.](#)

Code column

The Code column shows the size, in bytes or kilobytes, of the compiled executable object code for files and groups. If "0" is displayed in the Code column, it means that your file has not yet been compiled. If "n/a" is displayed, the file has no executable code that was compiled by the IDE, or the file is not in the current build target.

The values in this column do not reflect the amount of object code that will be added to the final output file. The linker may not use all of a file's object code when creating the final output. Instead, the

linker may ignore data and code that other files in the project do not reference (known as “deadstripping” code).

For more information on how the linker works, see [“Compiling and Linking a Project” on page 363](#).

Data column

The Data column shows the size, in bytes or kilobytes, of non-executable data in the object code for files in the project. A “0” in the Data column means that your file has not yet been compiled, or that the file does not contain a data section in its object code. If “n/a” is displayed, the file has no object code data.

Like the Code column, the data values in the Data column do not reflect the amount of data that will be added to the final output file.

The displayed numeric values are only for items in the current build target. Values for items not in the current build target are displayed in gray.

For more information on how the linker works, see [“Compiling and Linking a Project” on page 363](#).

Debug column

The Debug column indicates whether debugging information will be generated for the files in the project. A black marker in this column means that the IDE will generate debugging information for the corresponding item. A gray marker indicates that the IDE will generate debugging information for only some of the files in the group.

To generate debugging information for a:

- **File**—click in the Debug column next to the file.
- **Group**—click in the Debug column next to the group.
- **Project**—Alt/Option click in the Debug column.

To learn more about file debugging information, see [“Activating Debugging for a File” on page 104](#).

Target column

The Target column indicates whether an item is in the project's current build target. The IDE displays this column if a project has more than one build target. A dark marker in this column means that the corresponding item is in the current build target. A gray marker indicates that only some of the files in that group are in the current build target.

To assign or unassign a current build target for a:

- **File**—click in the Target column next to the file.
- **Group**—click in the Target column next to the group.
- **Project**—Alt/Option click in the Target column.

For information on adding or removing a file to or from a build target using the Target column, see [“Assigning Files to Build Targets” on page 98.](#)

Touch column

The Touch column indicates whether a file needs to be compiled the next time a target is built. A marker in this column means that the corresponding item will be re-compiled for the next [Bring Up To Date](#), [Make](#), [Run](#), or [Debug](#) command. A gray marker indicates that only some of the files in that group are set for re-compiling.

To enable or disable re-compiling for a:

- **File**—click in the Touch column next to the file.
- **Group**—click in the Touch column next to the group.
- **Project**—Alt/Option click in the Touch column.

To learn more about this feature, see [“Synchronizing modification dates” on page 90.](#)

Sorting items

You can sort the items listed in the Files view of the Project window. The title of each column behaves like a button. Click a particular column button to sort the listed items according to that column's title. The button is displayed differently from the other buttons to

indicate that it is currently active. Click the column button a second time to stop sorting the items according to that column's title.

For example, clicking the File button as shown in [Figure 3.3](#) sorts the listed items alphabetically by file name. Clicking the button again stops sorting according to file name criteria. Note that you can sort the items only one column at a time.

To change the sort order, click the Sort Order button:



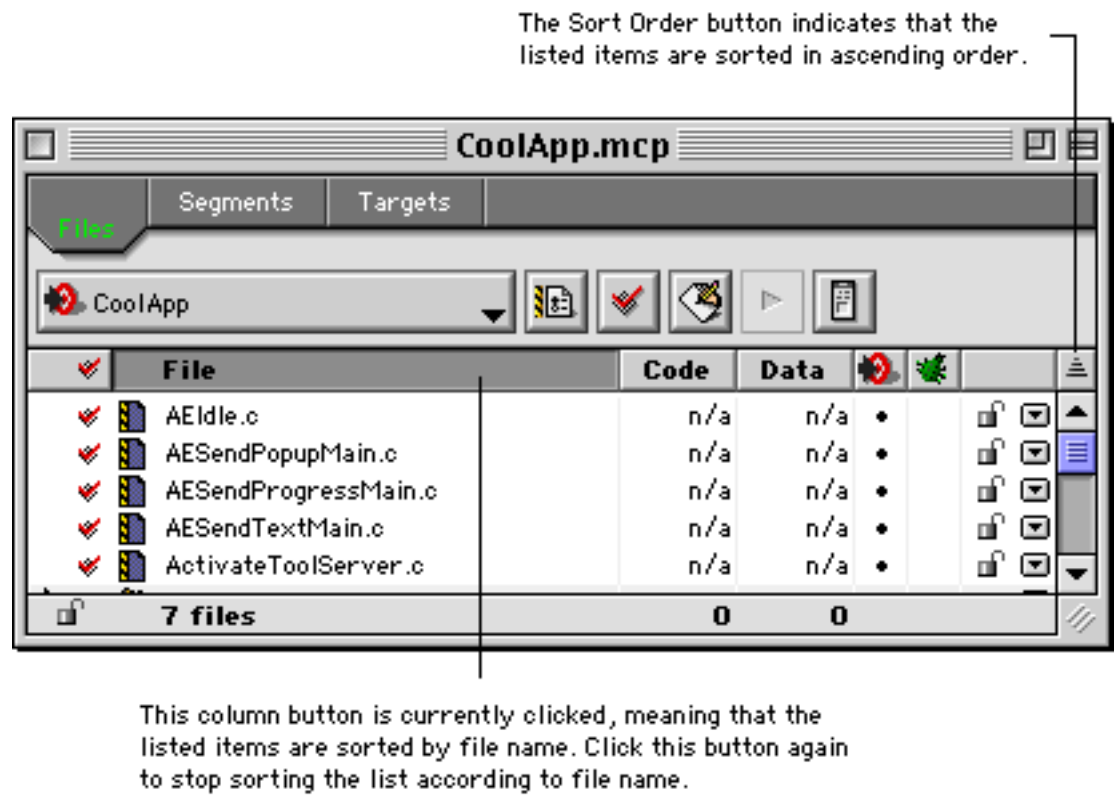
-  Items are currently sorted in ascending order.
-  Items are currently sorted in descending order.

Figure 3.3 **Sorting listed items in the Files view**



Interface pop-up menu

The Interface pop-up menu lists and opens interface and header files for your project source files. This pop-up menu also lets you touch or untouch the selected item and set other options. The additional options depend on the current build target.

For groups, the Interface pop-up menu lists the files within the group. Choosing a file in the pop-up menu will open that file.

For more information about opening interface and header files, see [“Interfaces pop-up menu” on page 114](#).

File Control pop-up menu

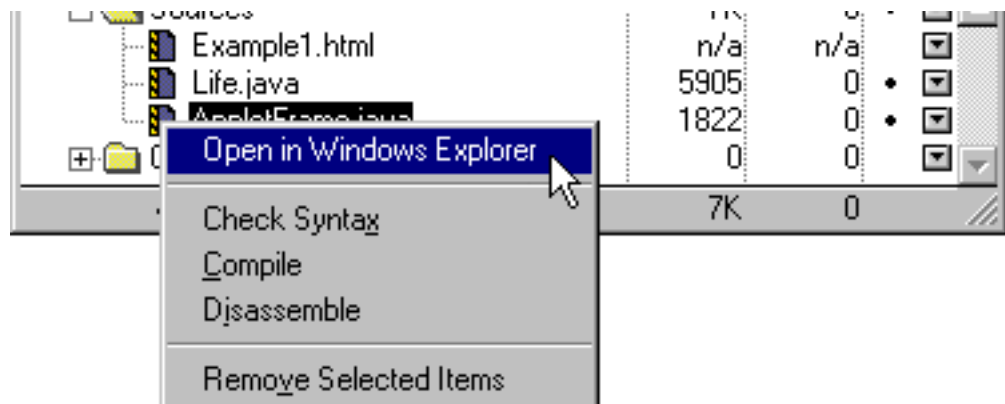
The File Control pop-up menu is shown in [Figure 3.4](#). To display this pop-up menu:

Windows Right-click an item’s icon in the Project window.

Mac OS Control-click an item’s icon in the Project window.

Solaris Click and hold on an item’s icon in the Project window.

Figure 3.4 File Control pop-up menu in the Project window



From the File Control pop-up menu, you can choose a command to operate on the item. The available commands depend on the selected item. The following examples show some of the pop-up menu commands:

Windows To see the path along which the IDE accesses a file, select **Open in Windows Explorer** from the pop-up menu.

Mac OS To see the path along which the IDE accesses a file, highlight **File Path** in the pop-up menu. To view the file and its enclosing folder from the Finder, select **Reveal in Finder** from the pop-up menu.

Checkout Status column

The Checkout Status column indicates whether files are checked in or checked out of a version control system (VCS). Use this column to track changes to your code, particularly when more than one person is working on your software project. The Checkout Status column, shown in [Figure 3.2 on page 45](#), only appears when you configure your CodeWarrior project to use a source code revision control system. For more information about using a particular revision control system, refer to its documentation.

Project Checkout Status icon

The Project Checkout Status icon shows whether a project is writable, as well as the file access permissions for that project. A revision control system assigns these permissions when you check in or check out a project file.

To learn more about the Project Checkout Status icon and how the icon relates to the access permissions assigned to your source code files, refer to [“Getting a File’s Status” on page 613](#). To learn more about using your particular revision control system, refer to its documentation.

Design View

The Design view shows the rapid application development (RAD) designs in your project. This view shows the objects that belong to particular designs and the names of those objects. The Design view is a convenient starting point for inspecting object names, properties, and design hierarchies.

When your project includes RAD designs, you can use the Project window’s Targets view to see the hierarchy of designs and build targets for the entire project. A single project file lets you manage multiple designs. Within each design, you can specify multiple build targets.

For more information about RAD designs, refer to [“RAD Designs and Layouts” on page 509](#).

Link Order View

The CodeWarrior IDE compiles a project’s files in the order shown in the Link Order view. This view is useful for explicitly specifying a custom build order. For example, if one file depends on information from a second file that has not been compiled yet, the IDE might fail to compile the project. You can avoid this problem by changing the ordering of the files in the Link Order view.

Changing the order of files in the Link Order view affects the final binary code produced by your project file. Note that items in the Link Order view can only be nested one level deep.

For more information, see [“Setting Link Order” on page 365](#). For more discussion about groups, refer to [“Managing Files in a Project” on page 75](#).

Mac OS The Link Order view might be named Segments for some build targets. Refer to [“Targeting Documentation” on page 27](#) for more information.

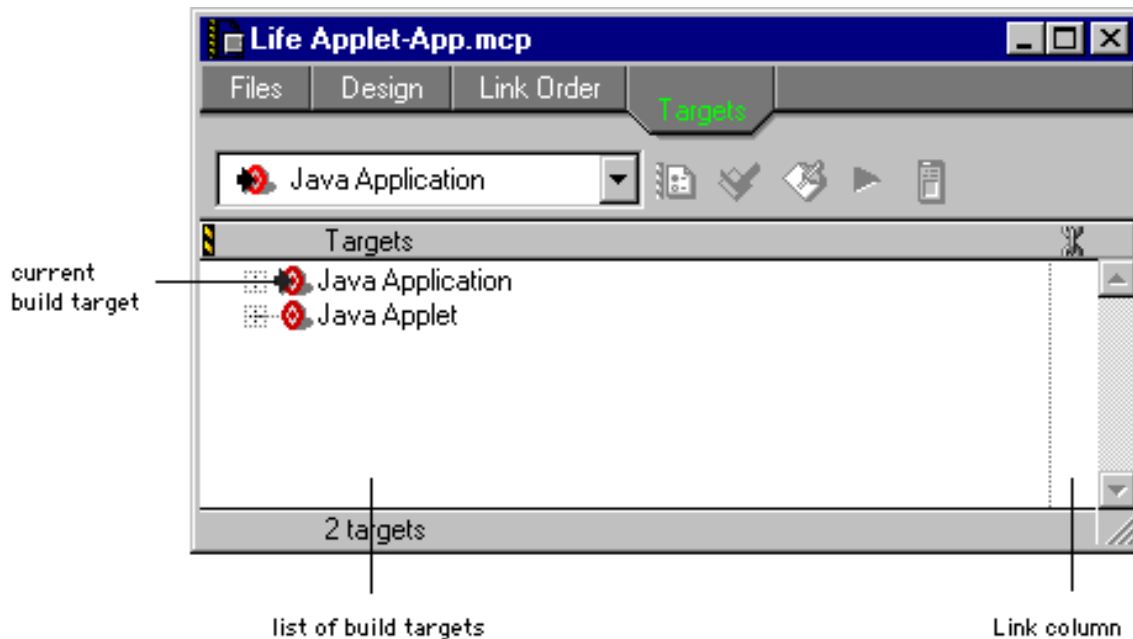
Embedded The Link Order view might be named Overlays for some build targets. Refer to [“Targeting Documentation” on page 27](#) for more information.

Targets View

The Targets view shows a list of the build targets in the project. This view also shows the objects that the build targets depend on to create a final output file. [Figure 3.5](#) shows an example Targets view.

If your project includes RAD designs, the Targets view shows the hierarchy of build targets within each design. For more information on working with build targets, see [“Working with Complex Projects” on page 91](#). To learn more about designs, see [“RAD Designs and Layouts” on page 509](#).

Figure 3.5 Targets view in the Project window



Creating a New Project

Creating a new CodeWarrior project file involves the following concepts:

- [Types of Project Files](#)
- [Choosing a Project Stationery File](#)
- [Naming Your New Project](#)
- [Using the New Project dialog box](#)
- [Modifying Your New Project](#)
- [Building Your New Project](#)

Types of Project Files

CodeWarrior uses these types of files to create new projects:

- **Project Stationery**—Project stationery contains pre-configured libraries, source code placeholders, and resource file placeholders. This kind of project file is useful for quickly creating new projects. Novice programmers and programmers

that want the convenience of pre-configured settings should use project stationery.

- **RAD Wizards**—Rapid application development (RAD) wizards present a series of dialog boxes that guide you through the process of creating a RAD project. To learn how to use these wizards, refer to [“Creating RAD Projects” on page 511](#).
- **Empty Project**—Empty project files do not contain any files whatsoever. This kind of project file is useful for custom configuration of compiler and linker settings. Advanced users can take advantage of empty projects to create fully customized applications.
- **Makefile Importer Wizard (Windows)**—The Makefile Importer wizard parses a Visual C nmake file or GNU make file and creates a CodeWarrior project. For more information about using the Makefile Importer, refer to [“Importing Makefiles into Projects \(Windows\)” on page 65](#).

The rest of this section discusses creating new projects using project stationery. For more detailed information, see [“Working with Project Stationery” on page 60](#).

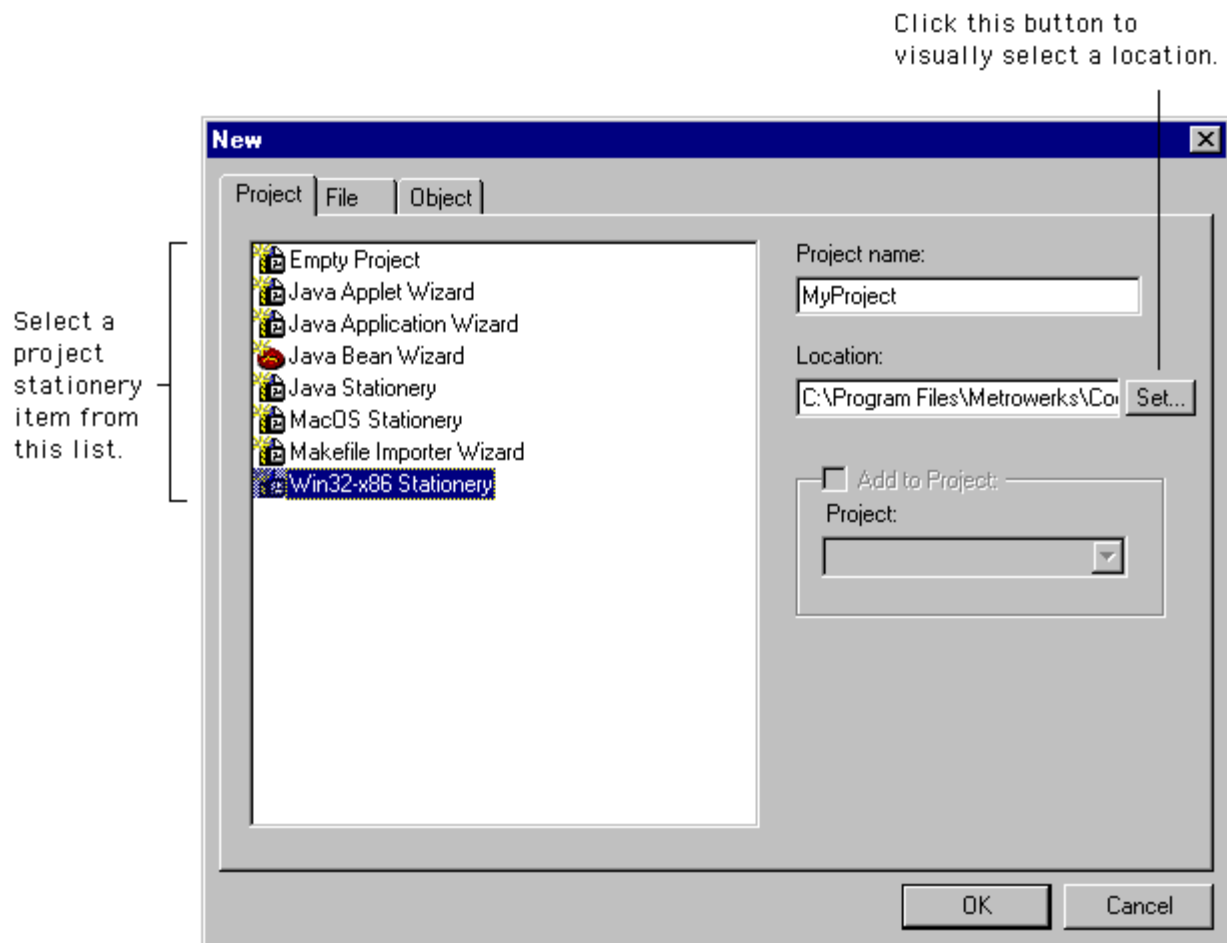
NOTE Advanced users who are interested in using empty projects should read [“Configuring Target Options” on page 321](#). That chapter provides information about configuring compiler and linker settings for use with projects.

Choosing a Project Stationery File

To use a project stationery file to create a new project, choose [New](#) from the [File Menu](#). The CodeWarrior IDE displays the New window, shown in [Figure 3.6](#) (Windows) and [Figure 3.7](#) (Mac OS).

Click the **Project** tab at the top of the New window to display the Project panel. The items listed in the Project panel represent various types of project files. To create new projects from stationery, you progress through a hierarchy of choices. First, you choose stationery for a particular platform target, such as Windows or Mac OS. Next, you specify a specific operating system, language, or programming framework. After you choose these more general options, you can choose from several pre-configured project stationery files.

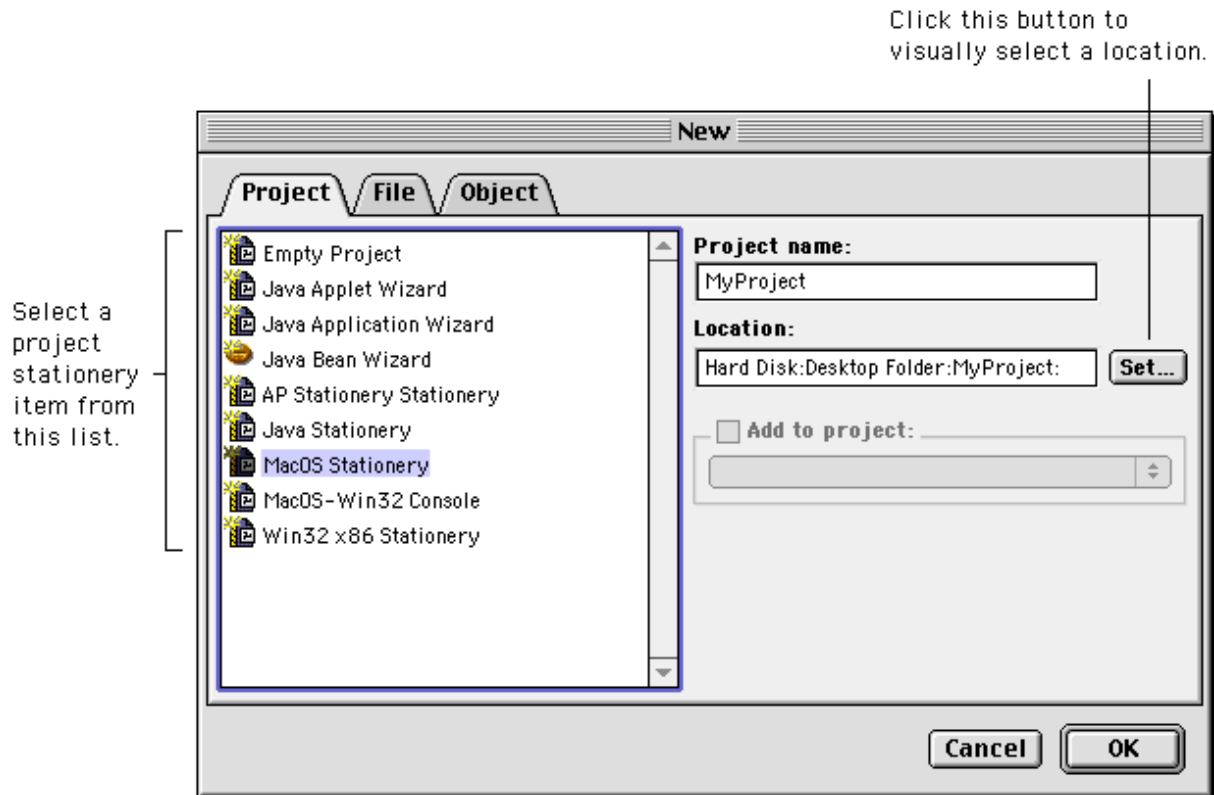
Figure 3.6 New window (Windows)



When you select an item in the Project panel list, various options become available along the right side of the panel.

To create a new project from stationery, select a file with “Stationery” in its name from the list shown in the Project panel. For example, if you want to create a project for the Mac OS platform, select the **Mac OS Stationery** item in the list.

Figure 3.7 New window (Mac OS)



To create an empty project that has no libraries or other support files, select the **Empty Project** item in the Project panel list.

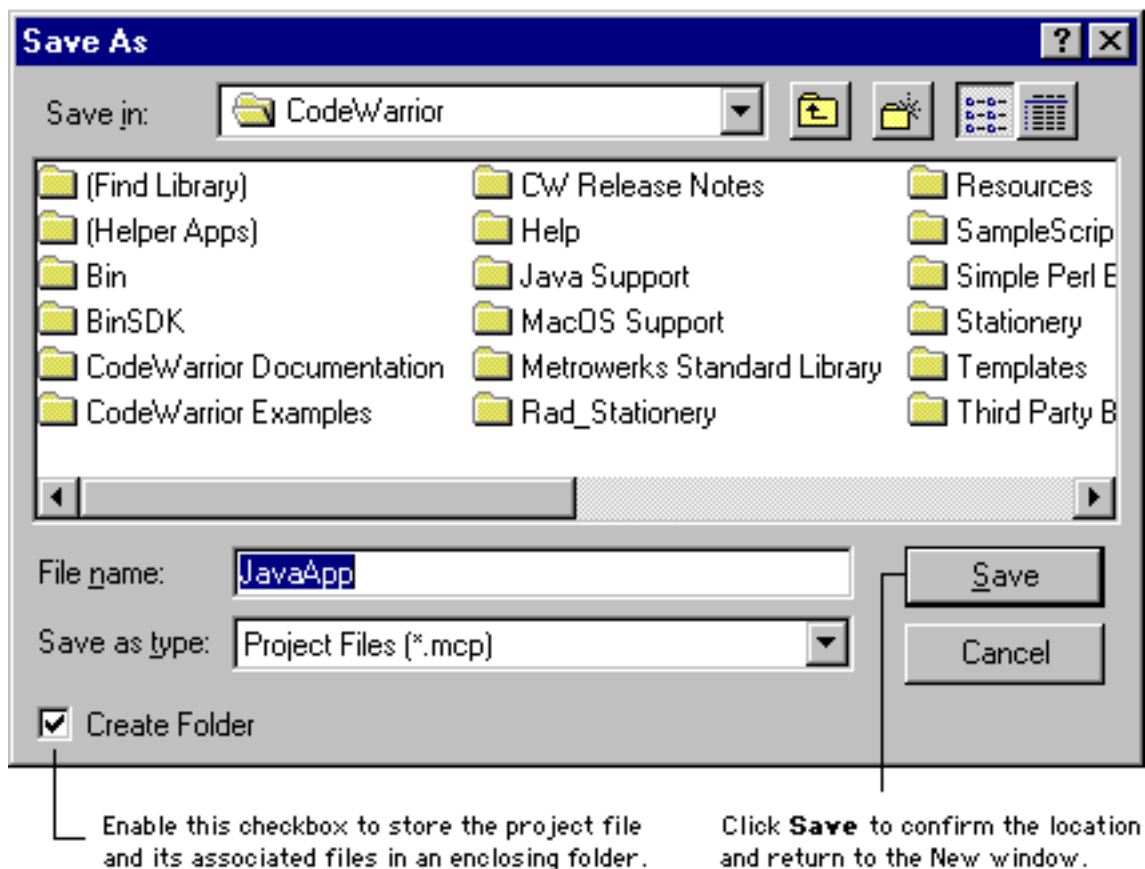
For beginners

Selecting **Empty Project** is not recommended because of the complexities of including the correct libraries and files and choosing the correct target settings. Instead of creating an empty project, use CodeWarrior's project stationery files. These files are pre-configured and convenient to use.

Naming Your New Project

After choosing [New](#) from the [File Menu](#) and selecting an item in the **Project** tab of the New window, type a name for the new project in the **Project Name** field.

Figure 3.8 Save dialog box (Windows)

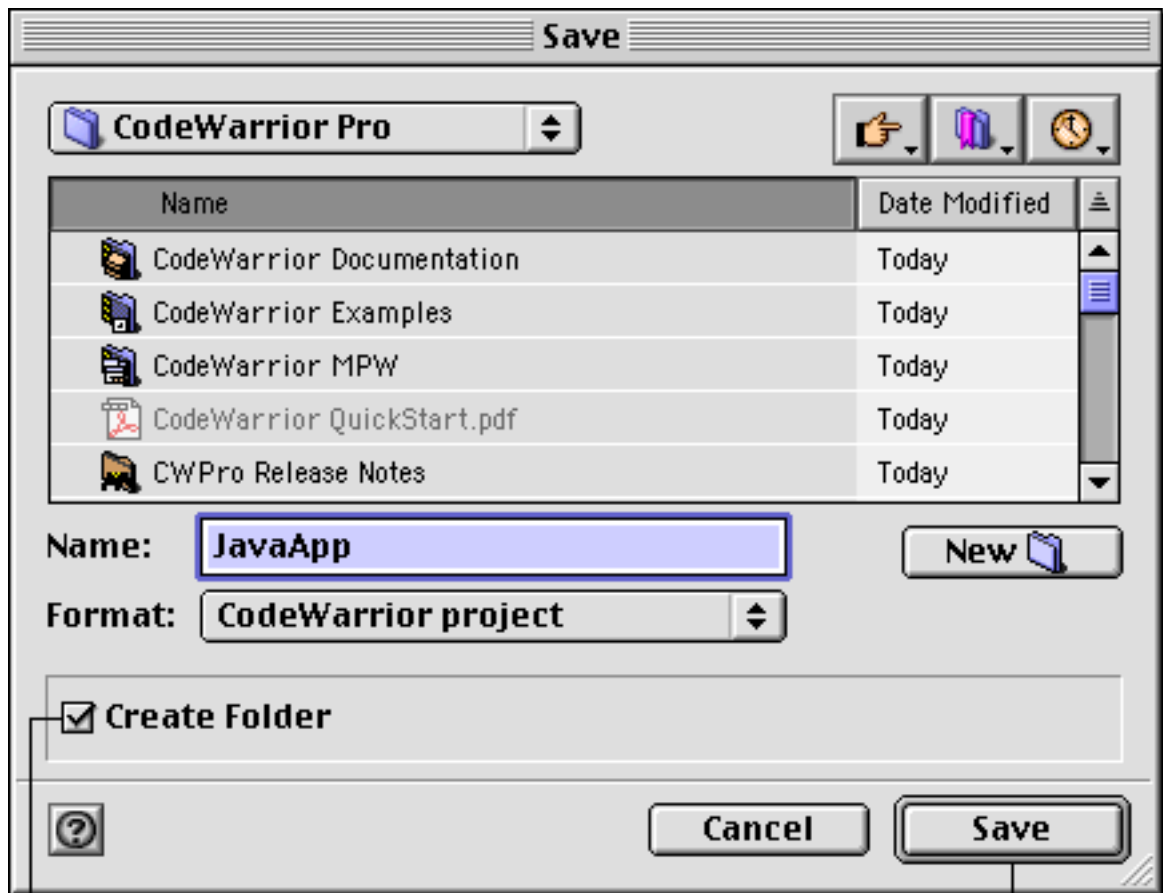


TIP We suggest naming your project file with a `.mcp` file name extension, like this: `MyProject.mcp`. This naming convention helps you quickly identify the project file on your hard disk. In addition, the Windows-hosted version of the CodeWarrior IDE uses this extension to identify the project file.

The **Location** field displays the full path to the folder in which the project is saved. To change the current path, type a new path directly into the field. Alternatively, click the **Set** button to display a dialog box, which is shown in [Figure 3.8](#) (Windows) and [Figure 3.9](#) (Mac OS). Use the dialog box controls to navigate to a location on your hard disk where you want to save the project. To create a new folder to contain the project file and all of its associated files, enable

the **Create Folder** checkbox. Then click **Save** to confirm the location and return to the New window.

Figure 3.9 Save dialog box (Mac OS)



Enable this checkbox to store the project file and its associated files in an enclosing folder.

Click **Save** to confirm the location and return to the New window.

After entering a name for the new project and specifying a location in which to save that project, click **OK** in the New window. The IDE displays the New Project dialog box.

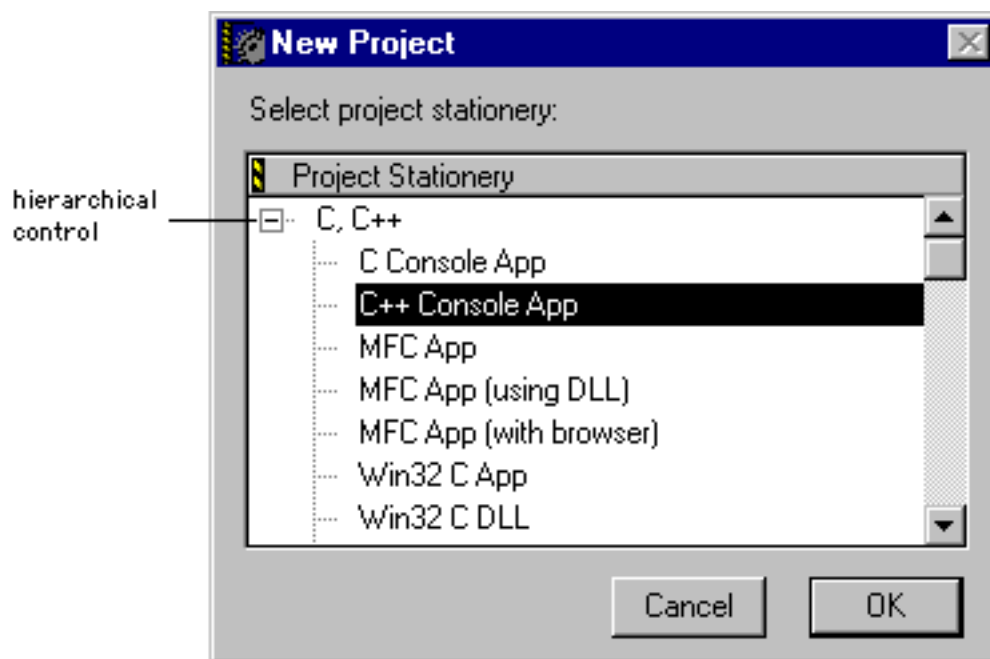
WARNING! If you try to save the new project, and you already have an existing project with the same name in the same location on the hard disk,

the IDE displays an error message. Be sure to use a unique name for your new project.

Using the New Project dialog box

The New Project dialog box, shown in [Figure 3.10](#), presents a hierarchy of project stationery based on your choice from the New window. For example, if you selected the **Win32-x86 Stationery** item in the New window, the New Project dialog box shows a hierarchy of project stationery for the Windows platform.

Figure 3.10 New Project dialog box



The topmost level of the hierarchy shows the languages or programming frameworks for which project stationery is available. Click the hierarchical control next to the particular language or framework you want to use for your project. Then, browse through the expanded list and select the stationery file representing the type of project you want to create.

When you select a stationery file and click **OK**, the CodeWarrior IDE automatically sets up the project, including:

- Creating a project folder with the same name as your project (if you enabled the **Create Folder** checkbox as discussed in [“Naming Your New Project” on page 55](#)), without the file name extension. The new folder contains your new project file, based on the stationery you chose.
- Setting [Preferences](#) and [Target Settings](#) to be the same as the settings stored in the chosen stationery.
- Opening the Project window. The new project contains libraries, source code placeholders, and resource file placeholders. If you chose to create an empty project, no files or libraries are displayed in your new Project window.

For example, suppose you want to create a new project that is geared towards a C++ console application. First, you would click on the hierarchical control next to the **C/C++** option in the New Project dialog box. The list expands and shows the project stationery for the C and C++ programming languages. The **C++ Console App** stationery file provides a simple console-based application structure that is suitable for small non-frameworked programming examples. When you select **C++ Console App** and click **OK**, the CodeWarrior IDE creates a new project with all the libraries and support files you need.

Modifying Your New Project

Most new projects created from stationery contain source files that are basically placeholders. You probably want to delete these files and replace them with source files of your own. See the section [“Managing Files in a Project” on page 75](#) to learn more about manipulating files in a project

You might also want to add additional libraries to your project file. To learn about which libraries to include, refer to the *Targeting* manual of interest to you. See [“Targeting Documentation” on page 27](#) for more information.

Building Your New Project

After you create your project and add your own files to it, you will want to build it to produce your software. To learn how to build a project, refer to [“Compiling and Linking” on page 361](#).

Working with Project Stationery

This section discusses how to create and use project stationery. It includes an explanation of project stationery that you can use to customize the project-creation process described in [“Creating a New Project” on page 52](#).

CodeWarrior projects can be configured to have multiple build targets. In addition, CodeWarrior projects can also contain subprojects. For more information, refer to [“Working with Complex Projects” on page 91](#) after reading this section.

The topics in this section include:

- [About Project Stationery](#)
- [About the Project Stationery Folder](#)
- [Creating Your Own Project Stationery](#)

About Project Stationery

A project stationery file is typically a minimal, pre-configured “starter” project file. Think of it as a template, or blank slate, that you use to quickly create a new project. When you create a new project or open a project stationery file, the CodeWarrior IDE creates a new project and, optionally, a new folder for the project. It then copies all the files related to the stationery project to the new folder.

A stationery project includes:

- All option settings for the project
- All files included in the stationery project (libraries, source code files, and resource files)
- (Mac OS) All segmentation and grouping information, including segment loader settings (68K projects only) and names

After creating your new project from stationery, you can open that project and begin developing code in the CodeWarrior IDE.

About the Project Stationery Folder

CodeWarrior provides project stationery for many different kinds of projects. Project stationery files for common types of projects are located in folders nested within the project stationery folder.

The following files can be included in the project stationery folder and are recognized by CodeWarrior as stationery projects:

- Normally-saved projects
- Project stationery files
- (Mac OS) Aliases to projects

You can find the project stationery folder at the following locations:

Windows The Stationery folder, inside the CodeWarrior folder on your hard disk, stores project stationery.

Mac OS The (Project Stationery) folder, inside the Metrowerks CodeWarrior folder on your hard disk, stores project stationery.

Solaris The (Project Stationery) folder, inside the CodeWarrior folder on your hard disk, stores project stationery.

Creating Your Own Project Stationery

You can create a unique stationery or “template” project file that includes the files and options you want to have for a starter project. This stationery project can be reused whenever you create a new project, so that you always start from your own customized settings.

Any CodeWarrior project can become a stationery project. A project must meet the following conditions to qualify as stationery:

- The project must be located in the project stationery folder, and
- The source files associated with the project must be stored together with the project.

When you choose your project stationery in the New Project dialog box, CodeWarrior duplicates the project and source files associated with the stationery project using the new project name.

Working with Projects

Working with Project Stationery

For beginners Before creating your own project stationery, you should become familiar with the project stationery supplied with CodeWarrior. This familiarity helps you understand the importance of the files included with the project stationery.

To create your own custom stationery, follow these steps:

1. **Create a new project from an existing project stationery file, or just create an empty project.**
2. **Choose Save A Copy As from the File menu.**

The dialog box shown in [Figure 3.11](#) (Windows), [Figure 3.12](#) (Mac OS), or [Figure 3.13](#) (Solaris and Linux) displays.

Figure 3.11 Save A Copy As dialog box (Windows)

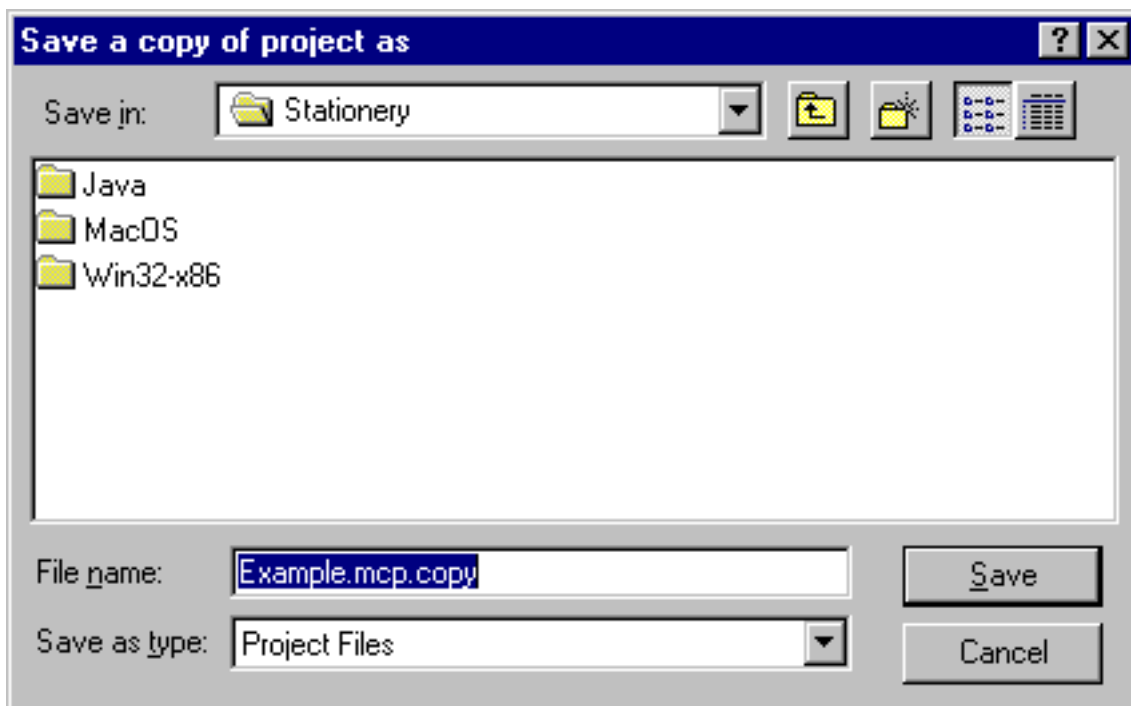


Figure 3.12 Save A Copy As dialog box (Mac OS)

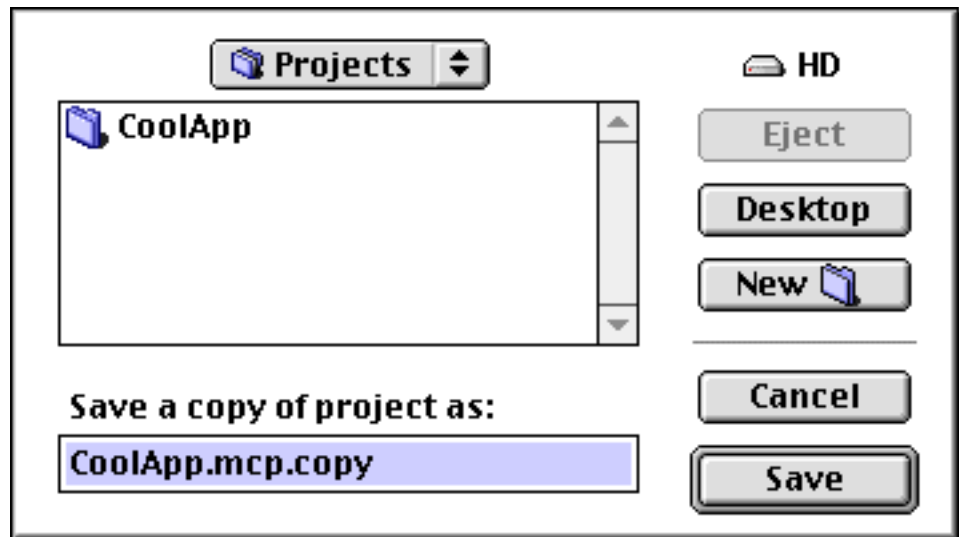
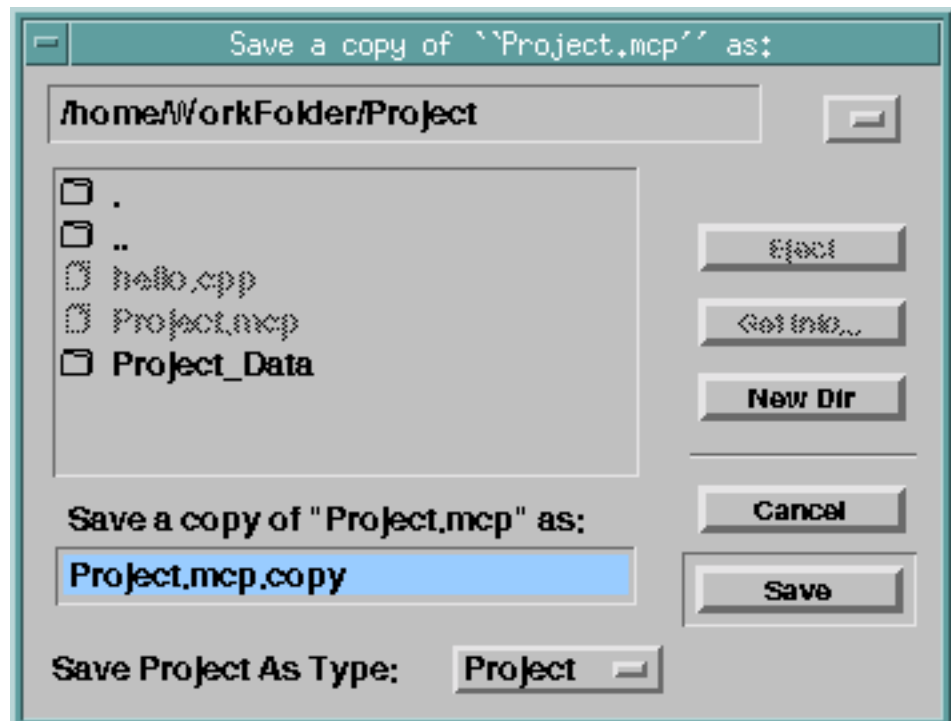


Figure 3.13 Save A Copy As dialog box (Solaris)



3. **Use the dialog box controls to save the new project to the project stationery folder.**

You must create a folder in which to store the new project. The CodeWarrior IDE recognizes a project stationery file by its enclosing folder. If you use the Windows-hosted version of the IDE, make sure that the project file's name includes the .mcp extension.

4. **Modify the project's settings to best suit your requirements.**

Be sure to add and remove files as necessary to create the exact base project you want.

5. **Make sure that copies of all the project's source files are present in that project's folder.**

If the source files are not in the folder, those source files will not be copied to any new projects created with the project stationery.

6. **Save your changes after you finish modifying the project.**

After you save all your changes, the project is ready for use as a new project stationery file.

When you save your stationery file in the project stationery folder, you can use that stationery the next time you choose the [New](#) command from the [File Menu](#).

When you use your custom project stationery file to create a new project, all of the files associated with your project stationery are copied to the new project's location, ready for modification.

To learn how to configure the [Preferences](#) settings in your project stationery, refer to ["Choosing Preferences" on page 256](#). You can customize [Target Settings](#) in a similar way. To learn more about how to do this, refer to ["Choosing Target Settings" on page 324](#).

For information about adding or changing files in the project, see ["Managing Files in a Project" on page 75](#).

See ["Backing up files" on page 120](#) for more information about saving a copy of the project under a different name.

The reason you want to save the stationery project before doing a lot of work with it is because you want to have a "starter" or "template" project file on your hard disk. You can make copies of

the “starter” project to get new projects quickly started, with all your favorite settings.

New projects created with your stationery file include all the settings you initially configured for that stationery file.

If at any time you decide that you want to use different project settings for new projects, you can create a new project stationery file. Just make sure that you have the Project window open, configure your options, and save your new project stationery file in the project stationery folder.

Importing Makefiles into Projects (Windows)

The CodeWarrior IDE can import Visual C nmake or GNU make files into CodeWarrior project files. The IDE uses the Makefile Importer wizard to process the files. This wizard performs the following tasks:

- parses the makefile
- creates a CodeWarrior project
- creates build targets
- adds source files as specified in the makefile
- matches the information specified in the makefile to the output name, output directory, and access paths of the created build targets
- selects a linker to use with the project

This section discusses the following topics:

- [Using the Makefile Importer Wizard](#)
- [Makefile Importer Options](#)

Using the Makefile Importer Wizard

To create a new project from a makefile, choose [New](#) from the Project menu. The IDE displays the New window, shown in [Figure 3.6 on page 54](#). Click the **Project** tab in the New window to display the Project tab, then select **Makefile Importer Wizard** in the list.

Next, follow the conventions for naming the new project and saving it to a particular location. These conventions are described in [“Naming Your New Project” on page 55](#).

When you finish naming the project and choosing a location in which to save it, click **OK** in the New window. The IDE displays the Makefile Importer wizard. See [“Makefile Importer Options” on page 66](#) for more information about configuring the options in the wizard.

Makefile Importer Options

The Makefile Importer wizard, shown in [Figure 3.14](#), presents several options that help you customize the conversion process. These options include:

- [Makefile Location](#)
- [Settings](#)
- [Diagnostic Settings](#)

After configuring these options as desired, click **Finish**. The Makefile Importer wizard presents a summary window. This window shows the current conversion settings. Click **Generate** to import the makefile according to these settings, or click **Cancel** to return to the wizard and make additional changes.

Makefile Location

This field is where you type the location of the makefile you want to import into a CodeWarrior project file. Alternatively, click the **Browse** button to display a standard dialog box. Use the dialog box controls to locate the makefile. After you select the makefile in the dialog box, click **Open**. The path to the makefile is displayed in the **Makefile Location** field.

Figure 3.14 Makefile Importer wizard

The screenshot shows the 'Makefile Importer Wizard' dialog box. It has a title bar with a close button. The dialog is divided into three main sections: 'Makefile Location', 'Settings', and 'Diagnostic Settings'. In the 'Makefile Location' section, there is a text box containing the path 'C:\Desktop\gnu_go_WIN\gnu_go_WIN\Makefile' and a 'Browse' button. The 'Settings' section contains two dropdown menus: 'Tool Set Used In Makefile' set to 'Microsoft Visual C++' and 'Metrowerks Tool Set' set to 'Metrowerks x86'. The 'Diagnostic Settings' section contains three checkboxes: 'Log Targets Bypassed' (checked), 'Log Build Rules Discarded' (checked), and 'Log All Statements Bypassed' (unchecked). At the bottom of the dialog are three buttons: '< Back', 'Finish', and 'Cancel'.

Settings

These options determine the tool sets used for the conversion of the makefile and the selection of a linker for the new project file. The options include:

- **Tool Set Used In Makefile**—Choose from this pop-up menu the tool set on which the makefile's build rules are based.
- **Metrowerks Tool Set**—Choose from this pop-up menu the linker tool set that is used for the CodeWarrior project created by the Makefile Importer

Diagnostic Settings

These checkboxes determine whether message reporting is enabled for various types of information created during the conversion process. The checkboxes are:

- **Log Targets Bypassed**—Enable this checkbox to log information about the build targets parsed in the makefile that were not converted to CodeWarrior build targets
- **Log Build Rules Discarded**—Enable this checkbox to log information about the build rules in the makefile that were discarded in the conversion to a CodeWarrior project.
- **Log All Statements Bypassed**—Enable this checkbox to log the same information as the **Log Targets Bypassed** and **Log Build Rules Discarded** checkboxes, as well as information about other items in the makefile that were not understood during the parsing process.

The messages are displayed in a Project Messages window after the conversion process concludes. This window is similar to the Message window. For more information, see [“Guided Tour of the Message Window” on page 379](#).

Opening an Existing Project

There are several ways to open a project file from within the CodeWarrior IDE. This section tells you how to open your projects so you can work on them.

Note that you can have many different projects open at a time, not just one project. See [“Choosing a Default Project” on page 74](#) for more information on default projects.

You can make an open Project window the active window by using the [Window Menu](#). To switch to one of several opened projects, just choose the project’s name from the Window menu.

The topics in this section are:

- [Using the Open Command](#)
- [Using the Open Recent Command](#)
- [Using the Project Window to Open Subprojects](#)

- [Opening Project Files Created on Other Host Platforms](#)
- [Opening Project Files from Earlier IDE Versions](#)

Using the Open Command

To open a project file, choose the **Open** command from the [File Menu](#). The IDE displays an Open File dialog box, as shown in [Figure 4.1 on page 111](#) (Windows), [Figure 4.2 on page 112](#) (Mac OS), and [Figure 4.3 on page 112](#) (Solaris).

Windows If not already set, use the **Files of Type** pop-up menu to select **Project Files**. The file list changes to show the project files that you can open.

Select the project file you would like to open, then click **Open**. The CodeWarrior IDE opens the project and displays it in a Project window.

If the project was created with an older version of CodeWarrior, you will be prompted to convert the older project to the newest version. If you decide to update, CodeWarrior saves a backup of the project and then converts the project to the newest version. See [“Opening Project Files from Earlier IDE Versions” on page 71](#) for more information.

To learn more about working with CodeWarrior project files, see [“Working with Projects” on page 41](#).

Using the Open Recent Command

The CodeWarrior IDE maintains a list of the projects and files you have opened recently in the [File Menu](#). As a convenience, you may use the **Open Recent** menu command to reopen one of these projects.

To learn about setting the number of files that the IDE remembers in this menu, see [“IDE Extras” on page 259](#).

Using the Project Window to Open Subprojects

If your project contains subprojects, and you want to open one of those subprojects, double-click the subproject file’s icon in the

Project window. The IDE displays the subproject in a new Project window.

To learn more about subprojects, and how to add them to your Project window, refer to [“Working with Complex Projects” on page 91](#).

Opening Project Files Created on Other Host Platforms

Project files are cross-platform compatible. For example, a project created on a Macintosh computer may be opened and used on a Windows computer.

To use a project created on another host platform, copy only its project file, not its associated Data folder, from the other host platform to your computer. After copying the project, open it in the IDE and recompile its files. Although a project’s format is cross-platform compatible, its compiled object code is not.

NOTE Before copying a project, make sure the project has a “.mcp” file name extension (without the quotes). For example, if the project is currently named `MyProject`, rename it to `MyProject.mcp`. The Windows-hosted version of the CodeWarrior IDE uses this file name extension to recognize project files. If the three-letter extension is not present, the Windows IDE cannot identify the project file.

TIP When creating any project, always add the “.mcp” extension to the project file’s name. This makes it easy to move project files to a different platform.

See also [“Options Pop-Up Menu” on page 142](#) for information on editing source code files created on other platforms and [“Host Flags” on page 333](#) for information on setting up access paths for a host platform.

Opening Project Files from Earlier IDE Versions

The CodeWarrior IDE version 3.0 and later cannot use a project file from any version of CodeWarrior earlier than 1.7. You must recreate the project file from scratch using the new IDE, or convert the project file to work with the new IDE. This section discusses how you can convert projects.

Converting a single 1.7 project

The CodeWarrior IDE has the ability to convert project files to the newest format from the format used by 1.7 versions of the IDE.

Windows To convert a single 1.7 project, open it from the latest version of the IDE. The IDE will prompt you to convert the project to the newest format. The name of the converted project file will be based on the name of the original file.

Mac OS The IDE uses a conversion utility called Project Converter. The name of the converted project file will be based on the name of the original file. Project Converter is located in the Other Metrowerks Tools subfolder of the Metrowerks folder on the hard disk where CodeWarrior is installed. You can convert a single project in either of two ways:

- You can simply open the 1.7 project from the latest version of the IDE. The IDE will automatically run Project Converter to convert the project to the new format.
- Or, you can drag and drop the 1.7 project file onto Project Converter itself.

NOTE 1.7.x projects must be converted to the new format if you want to work with them in the latest version of the IDE. The conversion of the project to the newest format is permanent. The latest version of the IDE cannot save a project in 1.7.x format.

Converting multiple 1.7 projects (Mac OS)

Project Converter has the ability to convert multiple 1.7 project files into a series of separate projects under the new format, or into one multi-target project file under the new format.

To use this feature, drag and drop a group of 1.7 project files onto Project Converter. Next, the IDE asks you whether to keep the files as separate projects or merge them into a multi-target project under the new format. Choose the option you wish.

Project Converter will prompt you to select a folder and name for the project or projects.

If you encounter problems, contact support@metrowerks.com.

You should read the Project Converter release notes before converting your projects. The release notes are in the IDE Notes folder, which is inside the CWPro Release Notes folder where CodeWarrior is installed. The release notes give you information on any late-breaking topics regarding the IDE.

Opening project files from versions prior to 1.7 (Mac OS)

Project Converter only converts project files from the 1.7 IDE releases with CodeWarrior 10 and CodeWarrior 11. If you have an earlier project, you must convert it to CW10 or CW11 format before using Project Converter.

To update a project to the 1.7 format, first make certain that the latest version of the IDE is not running. Then launch the 1.7 IDE and open the project.

The 1.7 IDE will update your project. You can then run the updated project through Project Converter to update that project to the newest format.

If you do not have version 1.7 of the IDE, you can find it on the CW Pro Tools CD. Look for the CW11 IDE and Prefs.sit file in the Other Metrowerks Tools folder.

Saving a Project

The CodeWarrior IDE automatically updates and saves your project when you perform certain actions. This section discusses the actions that cause a project file to get saved.

Your settings get saved when you:

- Close the project
- Change [Preferences](#) or [Target Settings](#) for the project
- Add or delete files for the project
- Compile any file in the project
- Edit groups in the project
- Remove object code from the project
- Quit the CodeWarrior IDE

You never have to manually save your project unless you want to create a copy of it, since the project is automatically saved when you perform the common actions listed above.

Items Saved with Your Project

When the CodeWarrior IDE automatically saves your project, it saves the following information:

- The names of the files added to your project and their locations
- All configuration options
- Dependency information (such as the touch state and interface file lists)
- Browser information
- The object code of any compiled source code files

Saving a Copy of Your Project

If you want to save a backup copy of a project file before you make some changes to the original, use the [Save A Copy As](#) command in the [File Menu](#). The CodeWarrior IDE creates a copy of the project file under a new name that you specify, but leaves the original project file unchanged. Furthermore, the IDE does not change the currently open project to use the new file name.

WARNING!

Do not attempt to make a copy of an open project from the desktop. Always close the project before copying the project file to prevent it from being corrupted.

Closing a Project

After you work with your project for a while, you may want to close it to work on another project, or you may want to quit the CodeWarrior IDE application to work on something else.

To close a project, make sure its Project window is frontmost, then choose **Close** from the **File Menu**.

You do not have to close your project before quitting the CodeWarrior IDE application, since your project settings are automatically saved. To learn more details about saved projects, refer to [“Saving a Project” on page 72](#).

The CodeWarrior IDE allows you to have more than one project open at a time, so you do not have to close a project before switching to another project. Just open your new project and begin working with it.

TIP Having multiple projects open at a time consumes more memory on your computer, and also causes project opening times to lengthen slightly. Keep these performance issues in mind as you work with multiple projects in the IDE.

Choosing a Default Project

Since the CodeWarrior IDE permits multiple open projects, it is sometimes ambiguous as to which project is used when you perform a **Make**, **Run**, or other operation. The IDE carries out the operation on the project file whose Project window is currently active.

However, source code files can be in more than one open project. You can specify a default project for builds by using the **Set Default Project** command in the **Project Menu**. In any ambiguous case, such as when a source code file belongs to more than one open project, the CodeWarrior IDE will operate on the default project you choose with the **Set Default Project** menu command.

The first project you open becomes the default project. If you close the default project, the default project becomes the project with the frontmost Project window.

Managing Files in a Project

This section discusses adding, moving, naming, organizing, viewing, marking for compilation, and removing files from your project. The topics in this section are:

- [Expanding and Collapsing Groups](#)
- [Selecting Files and Groups](#)
- [Adding Files](#)
- [Moving Files and Groups](#)
- [Creating Groups](#)
- [Removing Files and Groups](#)
- [Renaming Groups](#)
- [Touching and Untouching Files](#)

Expanding and Collapsing Groups

Groups display files in collapsible lists. To expand a group and view its files, click the hierarchical control next to the desired group's name. To close a group and view only its name, click the hierarchical control again.

Alt/Option click a group's name to expand that group and all its subgroups in the Project window, as shown in [Figure 3.15](#). Note that other groups at the same level as the chosen group are not expanded. Alt/Option click again to collapse the group.

Figure 3.15 Expanding groups and subgroups

Before

File	Code	Data		
Sources	6K	83	•	▼
ANSI Libraries	145K	36K		▼
Mac Libraries	63K	2K		▼
FAT Target Files	0	0		▼

After

File	Code	Data		
Mac Libraries	63K	2K		▼
68k	63K	2K	•	▼
MacOS.lib	31626	0	•	▼
MathLib68K Fa(...	32984	2160	•	▼
PPC	0	0		▼
InterfaceLib	n/a	n/a		▼
MathLib	n/a	n/a		▼
MSL RuntimePP...	n/a	n/a		▼

Ctrl/Command click a group's name to expand all groups at the same level in the Project window, as shown in [Figure 3.16](#). Ctrl/Command click again to collapse the sibling groups.

Figure 3.16 Expanding all sibling groups

Before

File	Code	Data		
Sources	6K	83	•	▼
ANSI Libraries	145K	36K		▼
Mac Libraries	63K	2K		▼
FAT Target Files	0	0		▼

After

File	Code	Data		
ANSI Libraries	145K	36K		▼
68k	145K	36K	•	▼
PPC	0	0		▼
Mac Libraries	63K	2K		▼
68k	63K	2K	•	▼
PPC	0	0		▼
FAT Target Files	0	0		▼

Selecting Files and Groups

From the Project window, you can select one or more files and groups to open, compile, check syntax, remove from the project, or move to a different group.

Selecting a group includes all of the files in that group, regardless of whether or not those files are visually selected in the Project window.

Selection by mouse-clicking

To select a single file or group in the Project window, click its name.

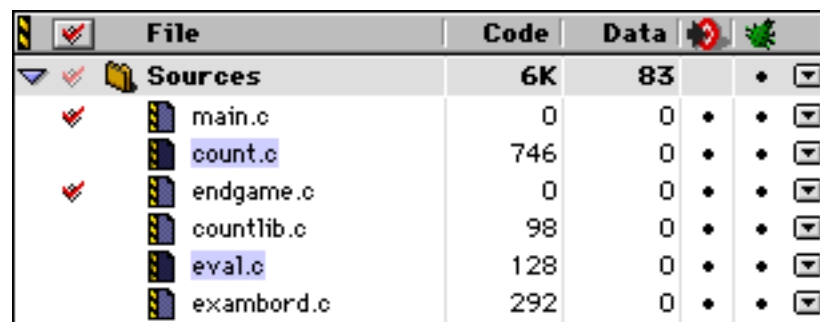
To select a consecutive list of files or groups, select the first file or group in the list by clicking its name, then Shift-click the last file or group.

Everything between and including the first and last file or group you clicked on will be selected, regardless of whether or not the selected items are groups or file names.

Another way to select a consecutive list of items is to drag-select them, like you would select items on the desktop.

Use Ctrl/Command click to select or deselect any non-consecutive file or group in the Project window, as shown in [Figure 3.17](#).

Figure 3.17 Non-consecutive file selection

The screenshot shows the IDE's Project window. At the top, there are tabs for 'File', 'Code', and 'Data'. The 'File' tab is active. Below the tabs, there is a tree view showing a project structure. The 'Sources' folder is expanded, showing a list of files. The files are: 'main.c', 'count.c', 'endgame.c', 'countlib.c', 'eval.c', and 'exambord.c'. The 'count.c' and 'eval.c' files are highlighted with a blue background, indicating they are selected. To the left of the file names, there are small icons: a red checkmark for 'main.c', a yellow folder icon for 'count.c', a red checkmark for 'endgame.c', a yellow folder icon for 'countlib.c', a yellow folder icon for 'eval.c', and a yellow folder icon for 'exambord.c'. To the right of the file names, there are columns for 'Code' and 'Data'. The 'Code' column shows the size of the file in bytes: 0 for 'main.c', 746 for 'count.c', 0 for 'endgame.c', 98 for 'countlib.c', 128 for 'eval.c', and 292 for 'exambord.c'. The 'Data' column shows the size of the file in bytes: 0 for all files. To the right of the 'Data' column, there are two columns of asterisks and a column of small square icons with a downward arrow.

Selection by keyboard

To select an using the keyboard, type the first few characters of the name of the item you want to select. As you type, the CodeWarrior IDE selects a file in the Project window as soon as the characters identify the file closest to your entry.

Use the Backspace/Delete key if you make a typo.

Use the Enter/Return key to open a file.

NOTE Only files in currently expanded groups in the Project window can be selected this way. Files in collapsed groups will not be matched with your keystrokes.

Adding Files

This section tells how to add files to your IDE project.

Here are the topics you will learn about in this section:

- [Where added files are displayed](#)—where files go when they are added to your project
- [Using the Add Files command](#)—one method of adding one or more files
- [Using drag and drop](#)—another method of adding one or more files
- [Using the Add Window command](#)—how to add one file

When adding a file to a project, the IDE adds the path used to locate that file to the project's [Access Paths](#). The Message window informs you whenever a new access path is added.

NOTE If you create a project using the **Empty Project** option, see [“Linker” on page 328](#) and [“File Mappings” on page 341](#). These sections describe how to configure the IDE so that you can add files to your project.

Normally, the IDE forces each file in a project to have a unique name. To remove this restriction, turn on the [Save project entries using relative paths](#) option in the [Target Settings](#) panel.

See [“Access Paths” on page 329](#) for more information about defining additional paths in your project.

For more information about the Message window, see [“Guided Tour of the Message Window” on page 379](#).

Where added files are displayed

Files are always added after the currently selected item in the Project window, or at the bottom of the Project window if there is nothing selected. To place a new file or group in a particular location, always select the file or group above the desired location before performing the [Add Files](#) or [Add Window](#) command.

Regardless of whether or not a selected group is expanded or collapsed, the added files are placed at the end of that group. To learn about how to select a file or group of files, see [“Selecting Files and Groups” on page 77](#).

NOTE If a group or a file within a group is not selected prior to adding files, a new group is created and appended to your project. The added files are placed in this new group.

Of course, you can always move a file or group of files to a new location within the project. To see how to move files and groups around in the Project window, see [“Moving Files and Groups” on page 86](#).

Using the Add Files command

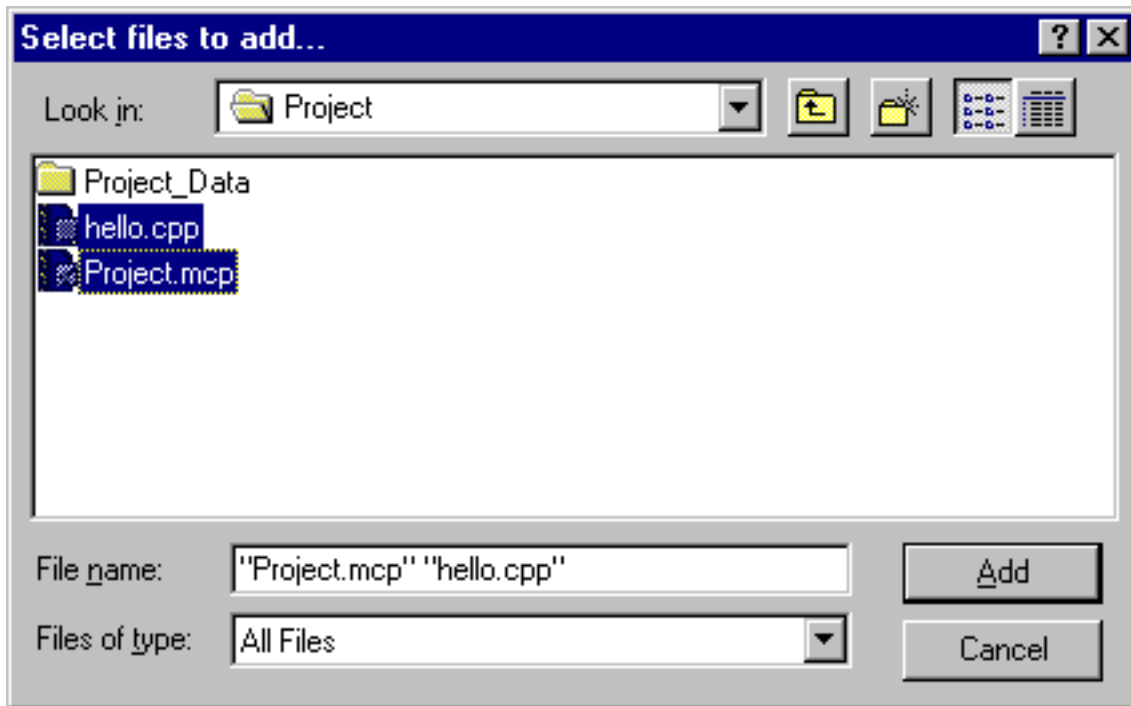
The [Add Files](#) command on the [Project Menu](#) opens a dialog box you can use to add files to your project. Use this command to add source code files, libraries, resource files, and other files.

In order for files to appear in the Add Files dialog box, the files must have a corresponding mapping as defined in the File Mappings preference panel. To examine and configure possible mappings for file names, refer to [“File Mappings” on page 341](#).

Add Files (Windows)

The Add Files dialog box is shown in [Figure 3.18](#). The **Files of Type** pop-up menu is initially set to view all file types.

Figure 3.18 Adding files to a project (Windows)



Use the dialog box controls to navigate through folders and drives to find files to add.

To add a single file to your project, select the file in the file list and click **Add**. Alternatively, you can double-click the file to add it to your project.

To select multiple files, as shown in [Figure 3.18](#), press the Control key while clicking on the file names in the dialog box.

To select a contiguous group of files, click on the first file name in the group, then press the Shift key and click on the last file in the group.

When you select all the files you want to add, click **Add**. Note that if your project contains multiple build targets, you will be prompted ([Figure 3.21 on page 85](#)) to select the build targets to which you want the files added. There may be a delay while the IDE locates the selected files and adds them to your project.

Clicking **Cancel** closes the dialog box without adding any files to the project.

Add Files (Mac OS)

The Add Files dialog box is shown in [Figure 3.19](#).

Use the pop-up menu at the top of the file list or the Shortcuts, Favorites, or Recent buttons to navigate through folders and drives to find files to add.

Figure 3.19 Adding files to a project (Mac OS)



To add a single file to your project, select the file in the file list and click **Add**. Alternatively, you can double-click the file to add it to your project.

To select multiple files, click the disclosure triangles in the list so that files you wish to add are visible. Shift-click each file name that you want to add to your project. If you change your mind about adding a particular file, Shift-click that file's name to deselect it.

When you select all the files you want to add, click **Add**. Note that if your project contains multiple build targets, you will be prompted ([Figure 3.21 on page 85](#)) to select the build targets to which you want the files added. There may be a delay while the IDE locates the selected files and adds them to your project.

Clicking **Cancel** closes the dialog box without adding any files to the project.

Add Files (Solaris)

The Add Files dialog box is shown in [Figure 3.20](#). The file path field at the top of the dialog box displays the path to the current directory listing.

Use the dialog box controls to navigate through folders and drives to find files to add. The Select Files To Add list shows the files you have added to your project

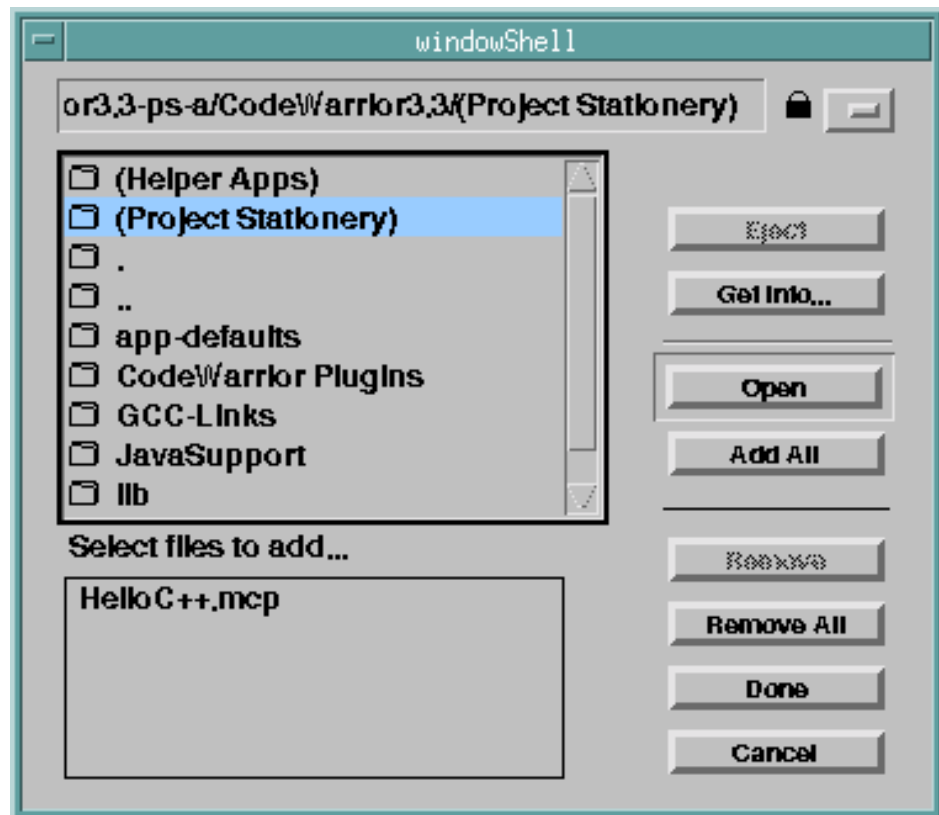
To add a file to your project, select the file in the directory listing and click **Add** to move that file to the Select Files To Add list. Alternatively, you can double-click the file to move it to the list. Repeat this procedure for each additional file that you wish to add to your project.

To add all the files displayed in the file list to the Select Files To Add list (all available files in the directory), click **Add All**.

To remove a file from the Select Files To Add list, select the file to be removed and click **Remove**.

To clear the Select Files To Add list, click **Remove All**.

Figure 3.20 Adding files to a project (Solaris)



When you select all the files you want to add, click **Done**. The files in the Select Files To Add list are then added to your project. Note that if your project contains multiple build targets, you will be prompted ([Figure 3.21 on page 85](#)) to select the build targets to which you want the files added. There may be a delay while the IDE locates the selected files and adds them to your project.

Clicking **Cancel** closes the dialog box without adding any files to the project.

Using drag and drop

When you drag and drop files or folders onto an open Project window, they will be added to the project.

To add files to your project using this method, first select the files or folders you want to add.

You can select files in many places, including the desktop or the multi-file search list in the CodeWarrior IDE's Find dialog box.

To complete the add operation, drag your selection onto the Project window. Note that if your project contains multiple build targets, you will be prompted to select the build targets to which you want the files added.

When dragging the selected files onto the Project window, the CodeWarrior IDE verifies that the files can be added to the project. When dragging a folder, the CodeWarrior IDE checks to make sure that the folder, or one of its subfolders, contains at least one source code file, library, or resource file, and that file is not already in the project.

If the selection does not contain at least one file recognized by the CodeWarrior IDE, the drag is not accepted.

NOTE See [“Linker” on page 328](#) and [“File Mappings” on page 341](#) for more information on configuring the IDE to recognize files.

Use the focus bar (an underline) that appears in the Project window to select the location where the files will be inserted.

Releasing the mouse button (dropping the files) adds the dragged items to the project, inserting them below the position specified by the focus bar. If your project contains multiple build targets, you will be prompted ([Figure 3.21](#)) to select the build targets to which you want the files added.

To create a new group and add files to it, drop the files when the cursor is over the blank space after the last group.

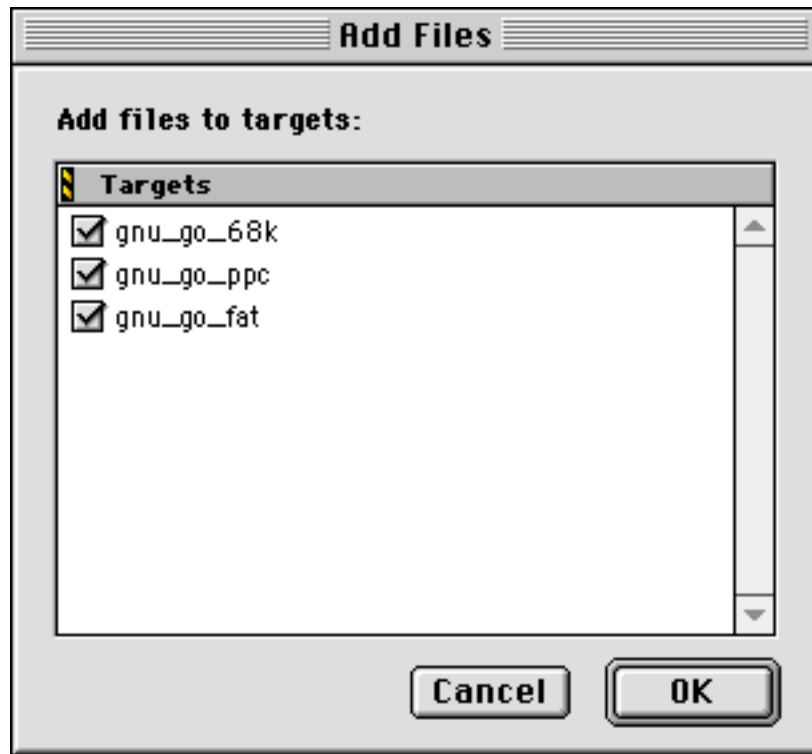
The CodeWarrior IDE does not allow the dragging of entire volumes (such as your hard disk) onto the Project window.

The IDE allows dragging and dropping outside of the Project window. You can also drag files to another application to open them in that application.

Although the CodeWarrior IDE supports dragging and dropping files into the Project window on all platforms, some platforms do

not allow you to remove files by dragging them out of the Project window. To learn how to remove files from the Project window, refer to [“Removing Files and Groups” on page 87](#).

Figure 3.21 Add Files dialog box



Using the Add Window command

The [Add Window](#) command adds the file associated with the active editor window to the project. You typically use this command when you create a new file and decide that you would like to add it to the active project.

To use the [Add Window](#) command, select a location in the Project window. Then, open a source code or text file, make sure its editor window is active, and select the [Add Window](#) command from the [Project Menu](#). If the window is untitled, the [Save As](#) dialog box displays, prompting you to select a location and name for the file. After you save the file, the IDE adds the file to the open project.

Note that if your project contains multiple build targets, you will be prompted ([Figure 3.21](#)) to select the build targets to which you want the files added.

NOTE The [Add Window](#) command is enabled when the active window is a text file, the file is not yet in the project, and the file has a recognized file name extension (to learn about configuring permissible file name extensions, refer to [“File Mappings” on page 341](#)). The [Add Window](#) command is dimmed otherwise.

Moving Files and Groups

To move one or more files or groups within a project’s Files view, or to arrange build targets in a project’s Targets view, select the files or groups to be moved. Selecting a group includes all of the files in that group, regardless of whether or not those files are visually selected in the Project window. If you need help selecting files and groups, see [“Selecting Files and Groups” on page 77](#).

Next, drag the selected files or groups to their new location in the Project window.

A focus bar (an underline) indicates where the selected files will be moved when the mouse button is released.

Whether you are moving files or groups depends on your selection. For example, if your selection consists of files, then the focus bar is shown on each line, under both groups and files.

If your selection includes at least one group, then the underline is shown only under other groups as you move the mouse in the Project window, allowing you to rearrange groups.

When the focus bar is displayed at the desired file or group position, release the mouse button. The selected files or groups are moved to the new position.

TIP The focus bar has a small arrow at the left end that indicates the level of insertion into the existing hierarchy. If the arrow is to the left of a group icon, the insertion will be at the same level as the target

group. If the arrow appears to the right of the icon, the files are inserted into the target group.

Creating Groups

With the Project window frontmost, and the Files view visible, choose the [Create New Group](#) command from the Project menu. Enter a name for the new group in the Create Group dialog box, then click **OK**. For information on how to change this name, see [“Renaming Groups” on page 88](#).

Mac OS The new group may use the word “Segment” in place of the word “Group” if the current build target is the 68K microprocessor.

Embedded The new group may use the word “Overlay” in place of the word “Group.”

Removing Files and Groups

You can use menu commands or drag and drop on supported platforms to remove files from a Project window.

Using menu commands

You can remove files from either the Files view or the Link Order view of the Project window. When removing files from the Project window, you need to be aware of a subtle issue. If you remove files from the Files view, *they are removed from the entire project, including all build targets*. When removing files from the Link Order view (also called the Segments view or Overlays view for some build targets), the files are removed from the current build target.

To learn more about build targets, refer to [“What is a Build Target?” on page 91](#).

To remove one or more files or groups in the Files view, first select the files or groups to be removed. Note that selecting a group includes all of the files in that group, regardless of whether or not those files are visually selected in the Project window.

To learn how to select files, refer to [“Selecting Files and Groups” on page 77](#).

After selecting the files or groups in the Files view, choose the **Remove Selected Items** command from the [Project Menu](#) or press Ctrl/Command-Delete. All the selected files and groups are removed from the project.

WARNING!

The Remove Selected Items command cannot be undone. If you mistakenly remove a group, you must re-add its files using either the [Add Window](#) or [Add Files](#) commands from the [Project Menu](#).

Using drag and drop

The CodeWarrior IDE supports dragging and dropping files into the Project window, although you cannot drag files out of the Project window on some platforms.

Renaming Groups

To rename a group, select the group by clicking on it, then press the Enter/Return key. You can also use the arrow keys to navigate to the group, and then press the Enter/Return key. The Rename Group dialog box displays ([Figure 3.22](#)). The **Enter Group Name** field shows the current name.

Figure 3.22 Changing a group name

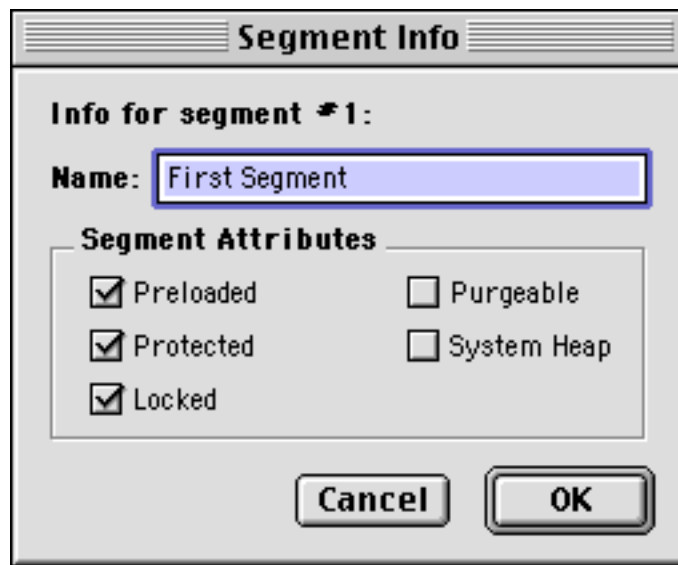


Type the new name in the **Enter Group Name** field and click **OK**. The name of the group is changed in the Project window.

If you selected more than one group, the Rename Group dialog box displays multiple times, once for each group. Each time, the **Enter Group Name** field displays the current name of the group.

Mac OS On Mac OS 68K build targets, the Segment Info dialog box shown in [Figure 3.23](#) displays. This dialog box contains a series of checkboxes that affect segment loader information. The Mac OS 68K linker gives the name displayed in the **Name** field to the code segment when building the project. See the *Targeting Mac OS* manual for more information on using segments.

Figure 3.23 Changing a segment name



Touching and Untouching Files

Use the [Touch column](#) shown in [Figure 3.2 on page 45](#) to mark files that need compilation. The CodeWarrior IDE does not always recognize file changes and may not automatically recompile all files in certain cases, which is why the Touch column features are useful.

There are three possible ways to make sure files get compiled:

- Click in the [Touch column](#) beside the file name in the Project window's Files view. A check displays in the Touch column next to the file name. This check indicates that the file will be recompiled the next time you build your project.
- Select the **Touch** command from the [Interface pop-up menu](#).
- Click on the check icon at the top of the Touch column. The IDE resynchronizes the state of the files in the project depending on the dates they were last modified. This is useful if the files have

been modified outside of the CodeWarrior IDE, perhaps by a third-party editor.

TIP If the file has not changed since it was last compiled, the first command in the Interface pop-up menu is **Touch**. When you choose the Touch command, the CodeWarrior IDE compiles the file the next time it makes your project. If the file has been changed since it was last compiled, the **Untouch** command is shown.

To unmark files so that they are not compiled, click again in the [Touch column](#) left of the file name, or choose **Untouch** from the [Interface pop-up menu](#).

Note that clicking the check icon at the top of the Touch column touches all files in the entire project.

Synchronizing modification dates

To update the modification dates stored in your project file, click the check icon at the top of the [Touch column](#). Alternatively, choose the [Synchronize Modification Dates](#) command in the [Project Menu](#).

After you choose the Synchronize Modification Dates command, the CodeWarrior IDE checks the modification date for each file in the project. If the file has been modified since it was last compiled, the IDE marks it for recompilation. This will resynchronize the state of the files in the project depending on the dates they were last modified. This is useful if the files have been modified outside of the CodeWarrior IDE, perhaps by a third-party editor that does not notify the CodeWarrior IDE when it modifies a file.

NOTE (Mac OS) Some third-party editors use AppleEvents to let the CodeWarrior IDE know when you modify a file. These editors include BBEdit™ from BareBones Software, Object Master™, and Alpha™. You do not need to use the Synchronize Modification Dates command if you use one of these external editors.

Working with Complex Projects

The CodeWarrior IDE provides flexible facilities for creating project files that use sophisticated build rules. This section discusses how to construct complex project files that may contain other projects or different kinds of build target code. This facility allows you to create powerful build hierarchies for your entire software project.

For example, you can set up a complex project that contains build targets for both shipping and debugging versions of your software. By switching between these build targets, the IDE generates different versions of the software during the development process. Each of these build targets can have its own settings. For example, the debugging build target could have optimizations disabled and debugging information enabled, while the shipping build target could have code optimizations enabled.

The topics in this section are:

- [What is a Build Target?](#)
- [What is a Subproject?](#)
- [What is a Design?](#)
- [Strategy for Creating Complex Projects](#)
- [Creating a New Build Target](#)
- [Changing a Build Target Name](#)
- [Changing the Build Target Settings](#)
- [Setting the Current Build Target](#)
- [Creating Build Target Dependencies](#)
- [Assigning Files to Build Targets](#)
- [Creating Subprojects Within Projects](#)

What is a Build Target?

A build target is a set of rules and settings that you configure to produce a an output file, such as an application or library.

The CodeWarrior IDE has the capability to build many different kinds of output files, or build targets, from one project file, as shown in [Figure 3.5 on page 52](#). For example, you can manage two separate

build targets in a single project, one for debugging purposes, and a separate one for your shipping code.

You can also define a specific build order for the various build targets in a project, so that the IDE builds a particular build target before trying to build another. This is useful for sharing resource files between other build targets. You can create a build target that depends on some other build target, forcing the IDE to first build the latter build target.

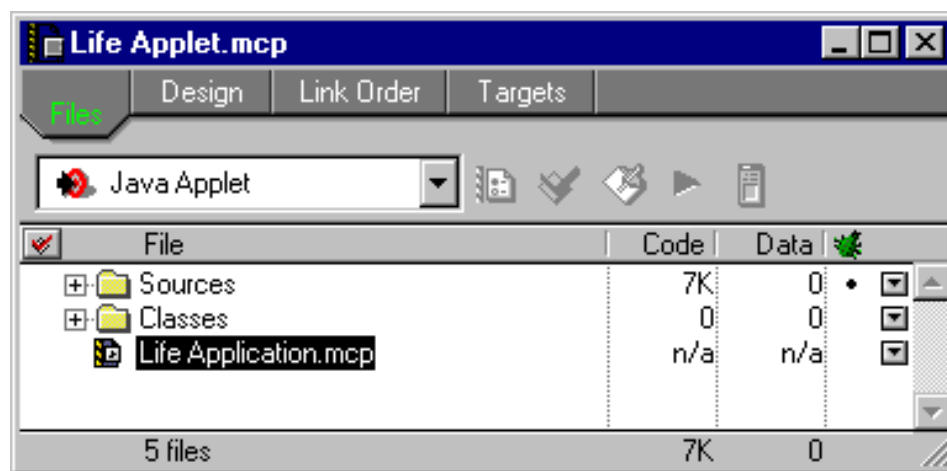
Each build target in the project can have its own distinct set of build settings.

To learn more about considerations for using build targets, refer to [“Targeting Documentation” on page 27](#) and [“Strategy for Creating Complex Projects” on page 94](#).

What is a Subproject?

A subproject is a normal, stand-alone project file that can be nested within another project file, like the project named `Life Application.mcp` shown in [Figure 3.24](#). Subprojects are useful if you have a project file that you want to keep separate from the main project file. This allows you to organize the build process into separate project files.

Figure 3.24 Subproject within a project



One case where this organization might be useful is for developing applications that use a plug-in architecture. Suppose your program uses many different plug-in modules, each sharing some common source code with other plug-ins. You might create one project file to build all the plug-ins by creating a separate build target for each plug-in. In this scenario, the project file for the plug-ins is the subproject. Including the subproject in the project file of the main application causes the IDE to build the subproject before building the main project.

A project file may be assigned to any build target in a project. To learn how to do this, refer to [“Assigning Files to Build Targets” on page 98.](#)

To learn how to add files to a project, refer to [“Adding Files” on page 78.](#)

You can instruct the CodeWarrior IDE to build one or more build targets in a subproject when it builds the containing build target in the main project. When the IDE builds a build target in the main project, it first builds any selected build targets in any of the subprojects.

You can optionally link the output of the main project with the output of the subproject’s build target. To create this link, select the marker for the build target in the Link column of the main project’s Targets view. For information on the Link column, see [“Targets View” on page 51.](#)

A subproject’s build targets are not built automatically when a subproject is added to a parent project. Only the chosen build targets within the subproject will be built.

Subprojects can be made specific to a build target. That is, if you add a subproject, you can choose which build targets it belongs to. Other build targets in the main project will not build the subproject unless the subproject file is added to the build target you choose.

What is a Design?

A design is a collection of components, layouts, and objects for rapid application development (RAD).

Each design in your project can have distinct sets of build targets. As you add source files to a design, those files are automatically added to the build targets in the design.

To learn more about CodeWarrior's RAD features, refer to [“RAD Designs and Layouts” on page 509](#).

Strategy for Creating Complex Projects

The choice of whether to use multiple build targets or subprojects within a project file depends on what works best for you. If you want access to all the source code in one project, then using multiple build targets is a good choice. Subprojects are better when you prefer to keep separate stand-alone project files.

For example, if you need to build a number of plug-in libraries that accompany your application, create a project that builds the subprojects with a single [Make](#) command. Then, include this project file as a subproject in your main application project file. When the IDE builds your main application, the subproject's plug-ins will be built first.

There is a limit of 255 build targets per project. Before you hit that limit, you should consider available memory and project load times. Projects with several build targets take up more disk space, take longer to load, and use more memory.

Once you get past ten or twenty build targets, you can improve performance by moving some of them off to subprojects. Anything that is not built often and uses a distinct set of source files is a good candidate for moving to a subproject.

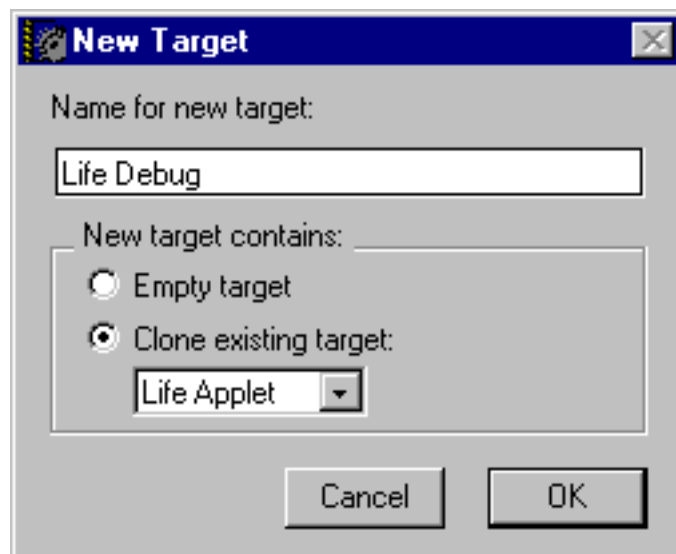
Creating a New Build Target

To create a new build target in your project, use the [Create New Target](#) command in the [Project Menu](#). This command displays if you have the Targets view selected in your Project window, as shown in [Figure 3.5 on page 52](#).

After you choose the Create New Target command, you see the dialog box as shown in [Figure 3.25](#). In this dialog box, you can enter the name of the new build target in the **Name For New Target** field.

Next, choose whether you want your new build target to be an empty build target or a clone of a previous build target. If you choose **Empty Target**, you need to configure all the settings of the build target as if a new Project window had just opened. If you choose **Clone Existing Target**, the settings for the new build target are copied from the build target displayed in the pop-up menu. You also get a copy of all the files that the original build target contains.

Figure 3.25 New Target dialog box



After creating a build target, you may want to associate that build target with other build targets. This association creates dependent build relationships. To learn how to do this, refer to [“Creating Build Target Dependencies” on page 97](#).

To learn how to configure settings for a build target, refer to [“Choosing Target Settings” on page 324](#).

When creating a new build target that depends an output file from another build target, you will need to click in the Link column next to the build target that creates the output file.

Changing a Build Target Name

To change the name of a build target in the Targets view of the Project window, double-click the name of the build target. The IDE

opens the Target Settings window for that build target, as shown in [Figure 9.4 on page 327](#). Select the [Target Settings](#) panel from the list of available panels. Change the name of the build target using the **Target Name** field.

You can also open the Target Settings window by choosing the [Target Settings](#) command from the Edit menu. The actual name of the Target Settings command will include the name of the current build target.

To learn more about the [Target Settings](#) panel, refer to [“Target Settings” on page 327](#).

Changing the Build Target Settings

Each build target in a project has its own settings. You modify these settings through the Target Settings window.

To open the Target Settings window for a particular build target in your project, double-click the name of the build target in the Targets view of the Project window or choose the [Target Settings](#) command from the Edit menu. The actual name of the Target Settings command will include the name of the current build target.

The left side of the Target Settings window contains a list of all the appropriate settings panels for the current build target. Select a panel to see the options you can set.

To learn how to change generic settings, refer to [“Configuring Target Options” on page 321](#). Settings panels that are specific to a particular platform target are described in a corresponding *Targeting* manual, such as *Targeting Windows* or *Targeting Mac OS*.

Setting the Current Build Target

You can change the currently active build target in a project by choosing the [Set Default Target](#) command from the [Project Menu](#). This command is useful if you want to switch between multiple build targets in a project and do a build for each of those build targets.

You can also change the current build target by using the Targets view of the Project window, as shown in [Figure 3.5 on page 52](#). The current build targets that the IDE will build are denoted by a circular icon (an archery “target”) with an arrow going into it. Click once on the name of a build target to choose it as the current build target.

Creating Build Target Dependencies

You can configure a build target to depend on other build targets. These dependencies are useful when you want to ensure that the IDE builds certain build targets before others. For example, you could have a build target that depends on information in a second build target. In order to compile correctly, the IDE must build the second build target before the first build target.

To specify that a build target depends on a second build target, follow these steps:

1. **Go to the Targets view of the Project window.**

Click the Targets tab at the top of the Project window to switch to the Targets view.

2. **Drag the second build target below and indented to the right of the first build target.**

The IDE adds an italicized entry for the second build target within the first build-target group. This entry makes the first build target dependent upon the presence of the second build target.

Now, the IDE builds the second build target before attempting to build the first build target.

To specify that the first build target should be linked with object code from the second build target, follow these steps:

1. **Click the Link column of the italicized entry for the second build target.**

The italicized entry for the second build target is located within the first build-target group.

2. **Note that the object code from the second build target appears in the Link Order view for the first build target.**

With the first build target active, click the Link Order tab at the top of the Project window to switch to the Link Order view. Notice that the object code from the second build-target group now appears in the link order for the first build-target group.

NOTE For Windows and Mac OS hosts, the object code from the second build-target group appears at the beginning of the link order. For Solaris and Linux hosts, the object code from the second build-target group appears at the end of the link order.

If you do not know how to create new build targets, refer to [“Creating a New Build Target” on page 94](#).

Refer to [“Setting the Current Build Target” on page 96](#) to learn how to set the current build target.

To learn more strategies for setting up complex projects with build targets and subprojects, refer to [“Strategy for Creating Complex Projects” on page 94](#).

Assigning Files to Build Targets

You can assign files to a project’s various build targets using either the Target column in the Project window’s Files view or the Project Inspector.

Using the Target column

The Target column in the Project window’s Files view indicates whether a file is in the current build target. The IDE displays this column only if the project has more than one build target. If a file is in the project’s current build target, that file has a marker in the Target column.

To assign a file to the active build target, click in the file’s Target column to place a marker. To remove a file from the active build target, click the file’s marker in the Target column.

To assign or remove all of the current build target's files, Alt/Option click in the Target column to display or erase markers.

Using the Project Inspector

You can use the [Project Inspector](#) window to assign a file to various build targets. To use the Project Inspector, you must select a file in the Project window. To learn how to do this, refer to [“Selecting Files and Groups” on page 77](#). After you select a file, choose the **Project Inspector** command from the [Window Menu](#). The IDE displays the Project Inspector window, as shown in [Figure 3.27](#).

Click the Targets tab to switch the view to that shown in [Figure 3.28 on page 102](#). This window shows you the build targets that your selected file belongs to. Check the checkbox next to a build target to include the file in that build target. Uncheck the checkbox to exclude the file from that build target.

If you make changes that you want to undo, click **Revert**. If you want to apply your changes and keep the Project Inspector window open, click **Save**.

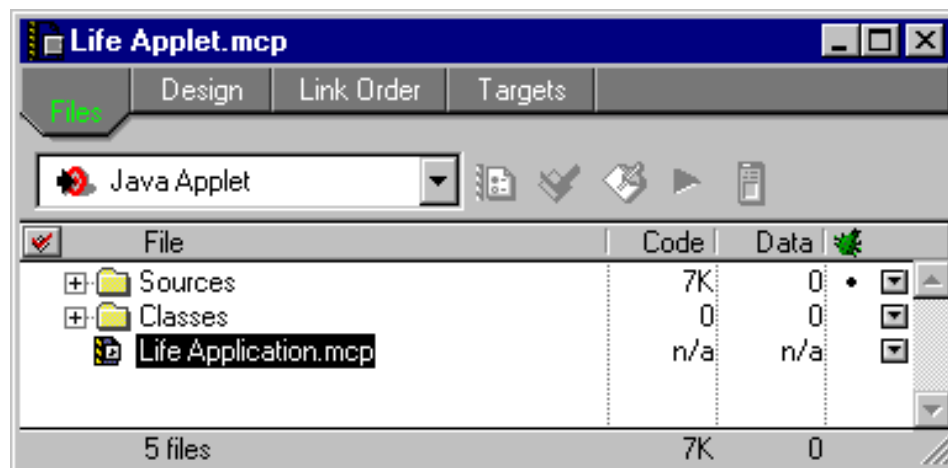
Creating Subprojects Within Projects

To create a subproject, drag and drop a project file into an open Project window. Note that if your project contains multiple build targets, you will be prompted ([Figure 3.21 on page 85](#)) to select the build targets to which you want the files added. After you do this, the Project window will look similar to [Figure 3.26](#), with a project file added to the list of files in your project. You can also add the file using the methods discussed in [“Adding Files” on page 78](#).

The subproject becomes a part of the main project file. When you perform a [Make](#) on the main project, the IDE builds the subproject first.

When you add a subproject to a project file, you can assign build targets to that subproject. To learn how to do this, refer to [“Assigning Files to Build Targets” on page 98](#).

Figure 3.26 Subproject within a project



Examining Project Information

The CodeWarrior IDE lets you review and configure information about your project and source code files in the [Project Inspector](#) window, shown in [Figure 3.27](#). To display this window, choose the [Project Inspector](#) command from the [Window Menu](#).

The Project Inspector window includes an Attributes view and Targets view.

To learn how to inspect and set files to compile with specific build targets, refer to [“Assigning Files to Build Targets” on page 98](#).

To learn more about a given file in your project, use the Project Inspector’s Attributes view. A file’s attributes include its name, its path to a location on your hard disk, the name of the project it belongs to, its code and data size (if it has been compiled), and whether or not the IDE generates its Debug Info when compiling the project.

Figure 3.27 Project Inspector window for attributes



If you decide to undo the changes you make to a file's attributes, click **Revert**. If you want to apply your changes and keep the Project Inspector window open, click **Save**.

To learn about configuring Debug Info for files, refer to [“Controlling Debugging in a Project” on page 104](#).

Mac OS To learn more about setting Initialize Before, Merge Into Output, and Import Weak, refer to [“Special Library Options \(Mac OS\)” on page 391](#).

Figure 3.28 Project Inspector window for build targets



Moving a Project

The CodeWarrior IDE stores all required information about your project in the project file. There are other files, usually stored in a folder having a name similar to your project file, that contain information about window positions, object code, debug info, browser data, and other settings. The CodeWarrior IDE does not need these additional files to recreate your project.

WARNING! (Windows) Previous versions of the Windows-hosted CodeWarrior IDE used a folder called `Resource.frk` to store state information

for the project file. `Resource.frk` is now obsolete and will no longer be generated or utilized by the CodeWarrior IDE.

To move the project on your hard disk, copy the project file (ending in `.mcp` if it obeys the project file naming convention) to a new location on your disk. If you want all the information in the additional files to travel with the project file, you can also copy the folder containing those files. However, the additional files are not needed. The CodeWarrior IDE can reconstruct a project's state when performing a [Bring Up To Date](#) or [Make](#) operation. In a revision control system, you would only check in the main project file and not the other files.

If your project file references other files via absolute [Access Paths](#), you may need to modify those paths when you move the project. To learn how to do this, see [“Access Paths” on page 329](#).

Refer to [“Opening Project Files Created on Other Host Platforms” on page 70](#), for more information.

Importing and Exporting a Project

CodeWarrior lets you import and export a project file in extensible markup language (XML) format.

To export an open project file, choose **Export Project** from the File menu. A dialog box opens and prompts you to name the exported file.

Mac OS You can choose to export the project file in **Importable XML** format or **Text File List** format. The latter format creates a plain-text listing of the project's text files.

To import an XML-formatted file, choose **Import Project** from the File menu. A dialog box opens for you to select a file to import. The IDE imports the selected file into a CodeWarrior project file, and then the IDE prompts you to save the project file.

TIP You should create a new folder in which to save the project file. This way, the IDE stores all files associated with the project in the same folder as the project file.

Controlling Debugging in a Project

Your program will probably not run correctly the first time you build it. In order to troubleshoot your program with the CodeWarrior debugger, you need to enable debugging information for your project and its files. This section tells you how to activate the debugging information.

The topics in this section are:

- [Activating Debugging for a Project](#)
- [Activating Debugging for a File](#)

Activating Debugging for a Project

To enable debugging for a project, choose **Enable Debugger** from the [Project Menu](#). The IDE configures the project's settings to produce debugging information automatically.

To learn how to manually configure debugging information for a project, refer to ["Choosing Target Settings" on page 324](#).

Activating Debugging for a File

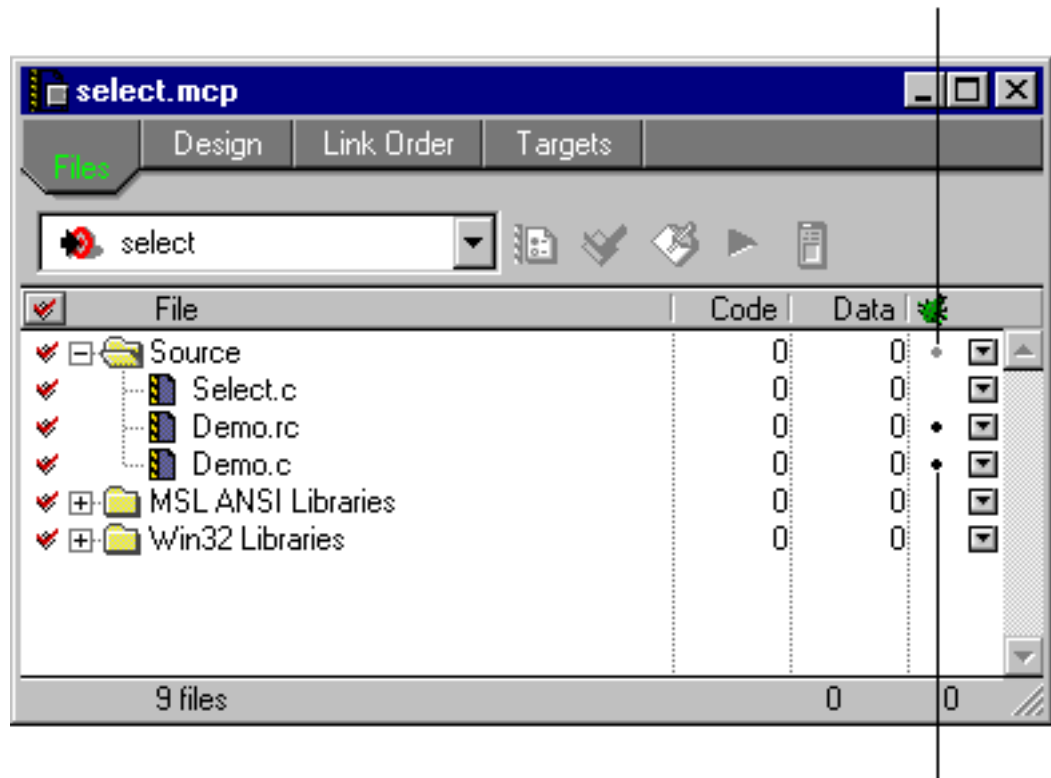
To generate debugging information for a source code file, click in the [Debug column](#). The IDE displays a Debug Info marker in the column, as shown in [Figure 3.29](#). CodeWarrior automatically sets this marker as you add files to your project, unless you deselected the appropriate option in the project's Linker panel.

You can also use the [Project Inspector](#) window to enable or disable debugging information. See ["Guided Tour of the Project Window" on page 42](#) for more information. When you add files to the project, the debug state matches the state of the **Enable Debugger** menu command setting in the [Project Menu](#). When you see the **Enable Debugger** command, the debugger is disabled and debugging

information is therefore disabled. When you see the **Disable Debugger** command, the debugger is enabled and automatically generates debugging information for the files you add to the project.

Figure 3.29 Debug Info markers

A gray dot indicates that only some of the files in the group are selected for debug info generation.



A black dot indicates that debug info will be generated for this file.

The Debug Info marker ([Figure 3.29](#)) indicates whether the IDE will generate debugging information for a file when building the project. Clicking a file's Debug Info marker removes it and disables debugging information for that file.

Whenever you change a Debug Info marker's status, the IDE marks the affected file for recompilation the next time your project is built.

You can generate Debug Info for all the files associated with the current build target. Shift/Option click in the Debug column to enable or disable debugging information for all of the files.

NOTE Marking a source code file for debugging inclusion does not mean that the IDE creates a debugging file during the linking process. The Linker settings panel for the build target contains the options that affect whether CodeWarrior creates a debugging file.

Debug Info marker for groups

Debugging information for groups can be toggled on and off by clicking the Debug Info marker in the Debug column ([Figure 3.29 on page 105](#)). The [Debug column](#) shows one of three markers for a group:

- **Black marker**—all files in the group generate debugging information
- **Gray marker**—only some of the files in the group generate debugging information
- **No marker**—no debugging information is generated

Adding Preprocessor Symbols to a Project

Sometimes you want to add your own symbol definitions to a project so that they are automatically included at the beginning of each source code file when the IDE builds the project.

An example of using your own symbols in the C or C++ language would be:

```
#define GLOBAL_DEBUG
```

Perhaps you want to define the GLOBAL_DEBUG symbol when building development versions of your code, but want to undefine it before shipping your final product.

To do this, you would create a precompiled header and insert the symbol definition into the header. See [“Using Precompiled or Preprocessed Headers” on page 371](#) for more information.

To learn more about this topic, refer to the *Inside CodeWarrior: C/C++ Tools* and *Inside CodeWarrior: Pascal Tools* manuals.

Working with Projects

Adding Preprocessor Symbols to a Project

Working with Files



This chapter introduces the concepts behind working with files in the CodeWarrior IDE.

In this chapter we discuss opening, creating, saving, closing, comparing, and printing files in the CodeWarrior environment.

To learn about editing files, refer to [“Editing Source Code” on page 137](#).

To learn about working with files using revision control systems, refer to [“Using the CodeWarrior IDE with Version Control Systems” on page 601](#).

The sections in this chapter are:

- [Creating a New File](#)
- [Opening an Existing File](#)
- [Saving a File](#)
- [Closing a File](#)
- [Printing a File](#)
- [Reverting to a Previously-Saved File](#)
- [Comparing and Merging Files & Folders](#)

Creating a New File

To create a new untitled window where source code or text may be entered, choose the [New Text File](#) command from the [File Menu](#).

After the new window appears, a text insertion point appears on the first line of the window. The CodeWarrior IDE places text that you type at this insertion point.

To learn more about text editing in the window you have just created, see [“Editing Source Code” on page 137](#).

Opening an Existing File

There are several ways to open a file with the CodeWarrior IDE. The methods discussed here are:

- [Opening Files from the File Menu](#)
- [Opening Files from the Project Window](#)
- [Opening Files from an Editor Window](#)
- [Opening a Related File](#)

NOTE You cannot open libraries with the CodeWarrior editor because of their binary format.

Opening Files from the File Menu

You can open two types of files:

- [Project file](#)—a file containing information on building a CodeWarrior project
- [Text file](#)—a source code, interface, or other text file

Project file

To open a project file, choose the [Open](#) command from the [File Menu](#). For information on opening CodeWarrior project files, see [“Opening an Existing Project” on page 68](#).

Text file

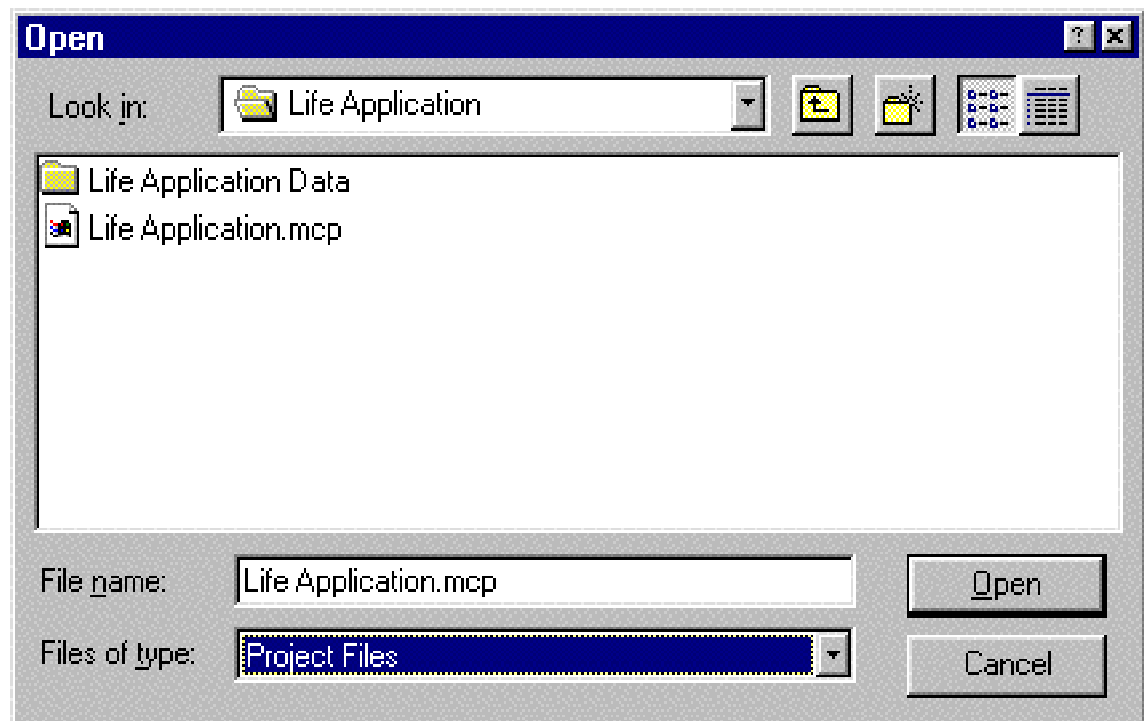
To open a text file or a source code file, choose the [Open](#) command from the [File Menu](#). The IDE displays an Open dialog box, as shown in [Figure 4.1](#) (Windows), [Figure 4.2](#) (Mac OS), and [Figure 4.3](#) (Solaris).

Windows From the **Files of Type** pop-up menu, select **All Files**. The list of files changes to show all the files in the current folder, including text files.

Mac OS The dialog box displays a list of the available project and text files in the current folder. If the file is a stationery text file, the IDE will open a new, untitled editor window and copy the contents of the stationery file into the window.

Solaris The dialog box displays a list of the available project and text files in the current directory.

Figure 4.1 Open dialog box (Windows)



Select the file you would like to open, and click **Open**. The CodeWarrior IDE opens the file in an editor window.

For more information about editing source code, see [“Editing Source Code” on page 137](#).

Figure 4.2 Open dialog box (Mac OS)

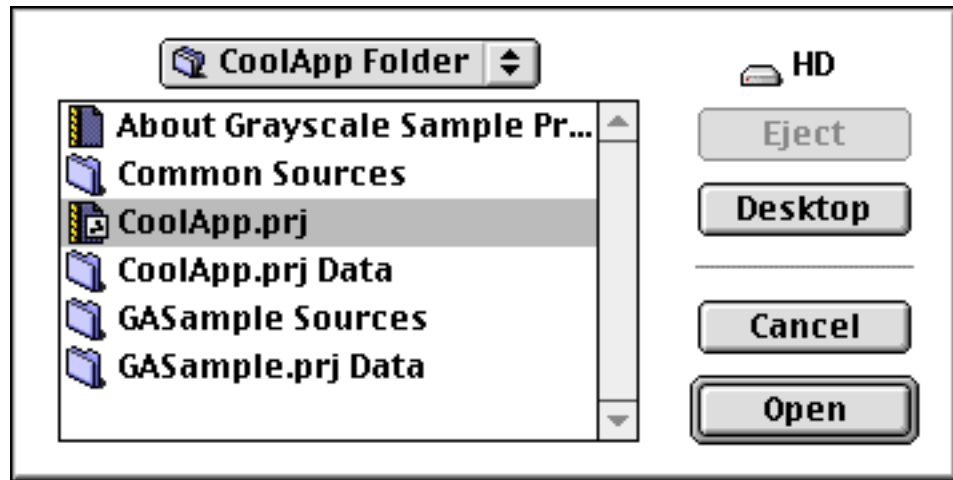
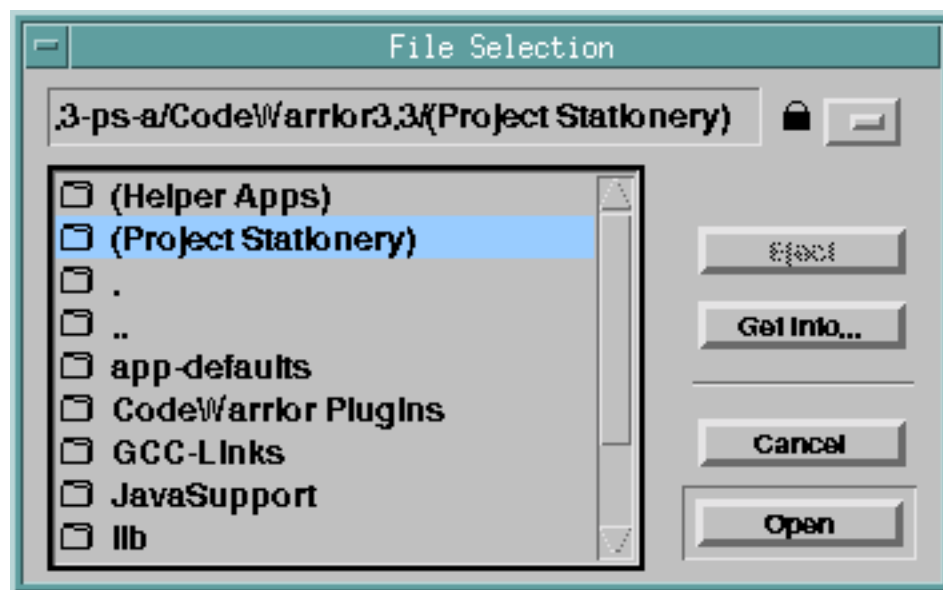


Figure 4.3 Open dialog box (Solaris)



Opening Files from the Project Window

There are different ways to open files from within the project window, depending on the type of file you wish to see. These different ways are:

- [File column](#)—opening a file that is in the project

- [Group pop-up menu](#)—opening a text source file from within a collapsed group
- [Interfaces pop-up menu](#)—opening an interface file included in a project's source file

TIP (Mac OS) Files that contains binary data cannot be opened and displayed in a CodeWarrior editor window. If the file was created using another application, double-clicking the file name in the [File column](#) will open the file in the application that created it. Refer to [“Disassembling Source Code” on page 379](#) to learn how to view the contents of a library file.

File column

If the file you wish to see appears in the [File column](#) of the project window's File or Link Order views, double-click the file name to open it. If the file is a text file, CodeWarrior opens it in an editor window.

Mac OS On the Mac OS, if the file is any other type, the CodeWarrior IDE opens the application that created the file. If the file is a binary file, such as a library file, it will not be opened, because binary files are not viewable using printable characters.

Another way to open a file is to select it, and then press the Enter/Return key. You can select multiple files in the Project window, and open them all by pressing the Enter/Return key. If you do not know how to select multiple files in a project, refer to [“Selecting Files and Groups” on page 77](#).

For more information about the File Column, refer to [“File column” on page 44](#).

TIP To open several files from the File Column at one time, hold down the Ctrl/Command key and click each file that you want to see. Then, double-click on one of the selected files.

Group pop-up menu

Another way to open a source file is to use the [Interface pop-up menu](#) for a group. A similar menu is shown in [Figure 4.4](#). From this

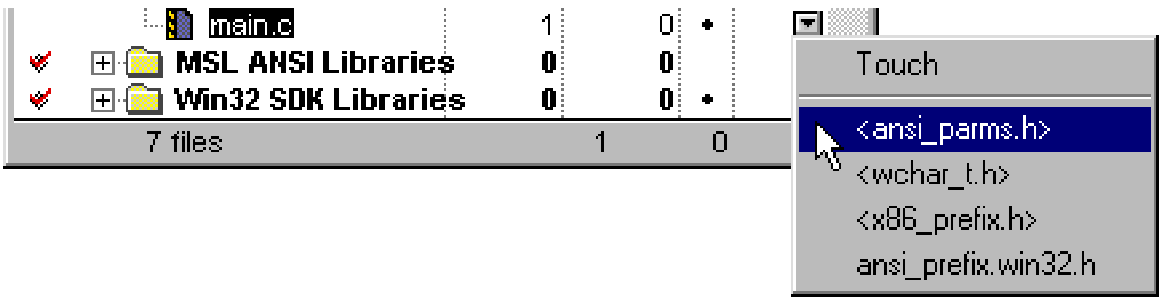
pop-up menu you may select the file in the group that you want to open.

You can open a source file by choosing it from the [Interface pop-up menu](#) for the group that contains the file. This works even if the group is collapsed and the file is not visible in the project window.

Interfaces pop-up menu

To open a header or interface file, click on the [Interface pop-up menu](#) to see a list of files. Select the file you want to open from this list, as shown in [Figure 4.4](#).

Figure 4.4 Interfaces Files pop-up menu in the Project window



Note that header files inside “<...>” are system header files located within the Metrowerks CodeWarrior folder. Files without these symbols are header files that you have created. Your header files are stored in the same folder as your project or other Access Paths you have designated. To learn more about Access Paths and how to configure them, refer to [“Access Paths” on page 329](#).

TIP To switch between a source file and its interface file, use the same name for both files, except for the extension. For example, name your files `foo.cpp` and `foo.h`. Then press `Ctrl/Command-`` to instantly switch between the two files.

When the Interfaces File Pop-up is clicked for a library file that is part of your project, you will only have the option to Touch or Untouch the library file. Since libraries do not contain header or interface files, these files can not be opened from a pop-up corresponding to a library file.

To learn more about touching files, see [“Touching and Untouching Files” on page 89](#).

Opening Files from an Editor Window

To open an interface file from within a source file you are editing, click the [Interface Pop-Up Menu](#) at the top left of the editor window as shown in [Figure 5.2 on page 140](#). This pop-up menu lists all interface or header files used by the source file. Select a file from the Interface Pop-Up Menu to open it in a new editor window.

NOTE If there are no files available in the menu, it means your text file does not contain source code, or that the source file has not yet been compiled.

Here is a different method. If you are editing any source code file, you can open an interface file mentioned anywhere in the text file with the [Find and Open ‘Filename’](#) command.

First, select text in the editor window containing the name of the interface file you would like to open. An example of a file name you might see in a C source code file is `stdio.h`. You could select `stdio.h` by double-clicking on the `stdio` portion of the text. Then, choose the [Find and Open ‘Filename’](#) command from the [File Menu](#).

The CodeWarrior IDE then searches for the file and opens the file in an editor window. If the IDE cannot find the selected file, a system beep sounds.

There is another method for opening a file. The following example demonstrates the usefulness of this method. Suppose you are editing a C++ `.cpp` file. Press Ctrl/Command ` to open a new editor window and display the `.h` file that corresponds to the `.cpp` file. Type the same keyboard shortcut again to display the `.cpp` file.

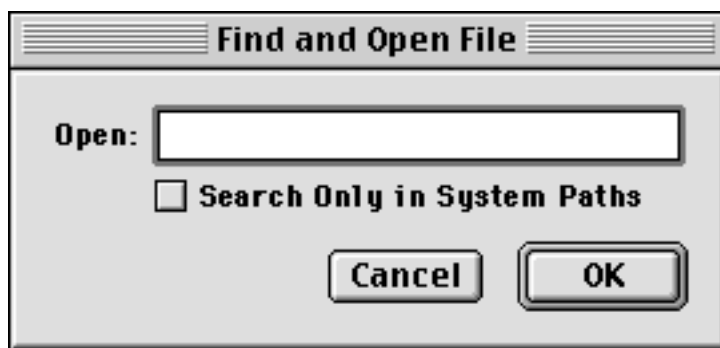
In order for this keyboard shortcut to work, your source and interface files must have the same name, excluding the file name extensions. For example, if you are editing `myFile.cpp` and you want to see the associated interface file, press Ctrl/Command ` to display `myFile.h` in a new editor window. To find these related

files, the IDE searches a project according to the items in the Access Paths settings panel. To learn more, refer to [“Access Paths” on page 329](#).

If you are editing a source code file and want to open a file without selecting any text, choose the [Find and Open File](#) command from the [File Menu](#). This command will use the settings in the [Access Paths](#) for the project to search for the file to open.

After you choose [Find and Open File](#), the CodeWarrior IDE then displays a dialog box, as shown in [Figure 4.5](#). Type the name of the file you wish to search for in the Open editable text field.

Figure 4.5 Find and Open File dialog box



If you want to search both [System Paths pane](#) and [User Paths pane](#) directory paths (all paths specified in the [Access Paths](#)), turn the **Search Only in System Paths** option off.

To search only the CodeWarrior directory structure (the paths specified in the [System Paths pane](#) of the [Access Paths](#)), click on the **Search Only in System Paths** option to enable it.

To learn more about Access Paths and how to configure them, refer to [“Access Paths” on page 329](#) for more information.

Opening a Related File

If you are working in a source code file and wish to open the corresponding interface file, or working with an interface file and wish to open the corresponding source file, use the Ctrl/Command

` keyboard shortcut. You can use this shortcut to easily switch back and forth between the two files.

Saving a File

This section describes the many ways that the CodeWarrior IDE can save files. The topics discussed are:

- [Saving one file](#)
- [Saving files automatically](#)
- [Renaming and saving a file](#)
- [Backing up files](#)
- [Saving as a MS-DOS, Mac OS, or UNIX text file](#)

NOTE (Mac OS and Solaris) When saving a file, you will often have the option of saving it as a stationery file. A stationery file is like a template or “starter” file. For example, creating a stationery file would be useful if you had standard documentation header text that you wanted to appear at the top of every file you create. If you create a stationery file with the header text, you could start every new file by opening this stationery file. See [“Opening Files from the File Menu” on page 110](#) for more information.

Saving one file

To save your changes to the current editor file, choose the [Save](#) command from the [File Menu](#). The CodeWarrior IDE saves your file to your hard disk.

The [Save](#) command is dimmed if the window is new and has no data, if the contents of the active window have already been saved, or when the active window is the project window.

NOTE If the file is new and untitled, the CodeWarrior IDE displays the Save As dialog box, described in [“Renaming and saving a file” on page 118](#). Choose a name and location for your new file with this dialog box.

Projects are saved when they are closed, when you quit or exit the CodeWarrior IDE, or when the [Save A Copy As](#) command is selected. You don't need to explicitly save projects.

Saving all files

To save your changes to all the files currently open, press the keyboard shortcut Shift-Ctrl-S (Windows) or Option-Command-S (Mac OS). The CodeWarrior editor saves all the modified files to your hard disk.

Saving files automatically

The CodeWarrior IDE can automatically save changes to all your modified files whenever you choose the [Preprocess](#), [Precompile](#), [Compile](#), [Disassemble](#), [Bring Up To Date](#), [Make](#), or [Run](#) commands from the [Project Menu](#).

Using the [Save open files before build](#) feature can save your work if your program should crash while running. However, if you're experimenting with a change and don't want to save changes, you may want to turn this option off.

To learn about how to enable or disable this feature, refer to the [Save open files before build](#) option in the section of this manual titled "[Editor Settings](#)" on page 273.

Renaming and saving a file

If you want to save a new untitled file or save a file under a new name, use the [Save As](#) command on the [File Menu](#). If the file is in the current project, the CodeWarrior IDE updates the project to use the new name.

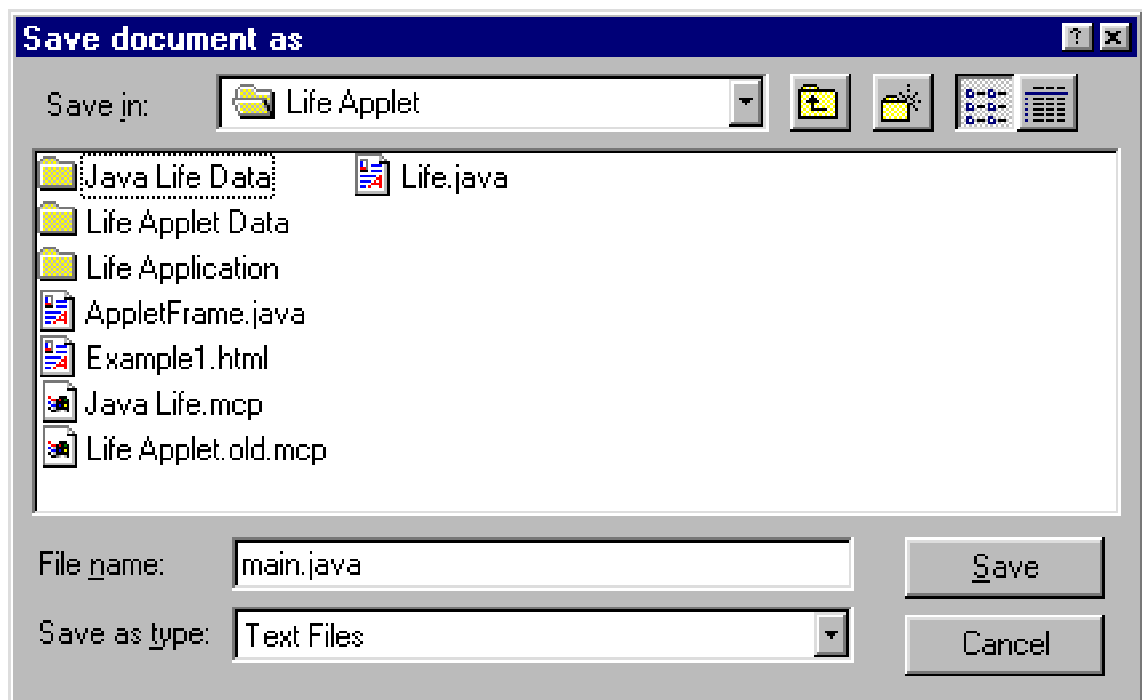
When you choose [Save As](#) from the [File Menu](#), the CodeWarrior IDE displays the dialog box shown in [Figure 4.6](#) (Windows), [Figure 4.7](#) (Mac OS), or [Figure 4.8](#) (Solaris).

Mac OS and Solaris Choose the Text or Stationery button to save the file as either a text file or a stationery file.

NOTE (Mac OS) Beginning with Mac OS 8.5, the dialog box that you see when you select the Save As command will no longer appear as

shown in [Figure 4.7](#). Instead of clicking the Stationery radio button directly, you must first tell the IDE that you want to use the stationery option. To do this, select **Stationery Option** from the Format pop-up list. When the Stationery Option dialog box appears, click the Stationery radio button and click **OK**.

Figure 4.6 Save As dialog box (Windows)



Choose the file location and name the file, then click the **Save** button.

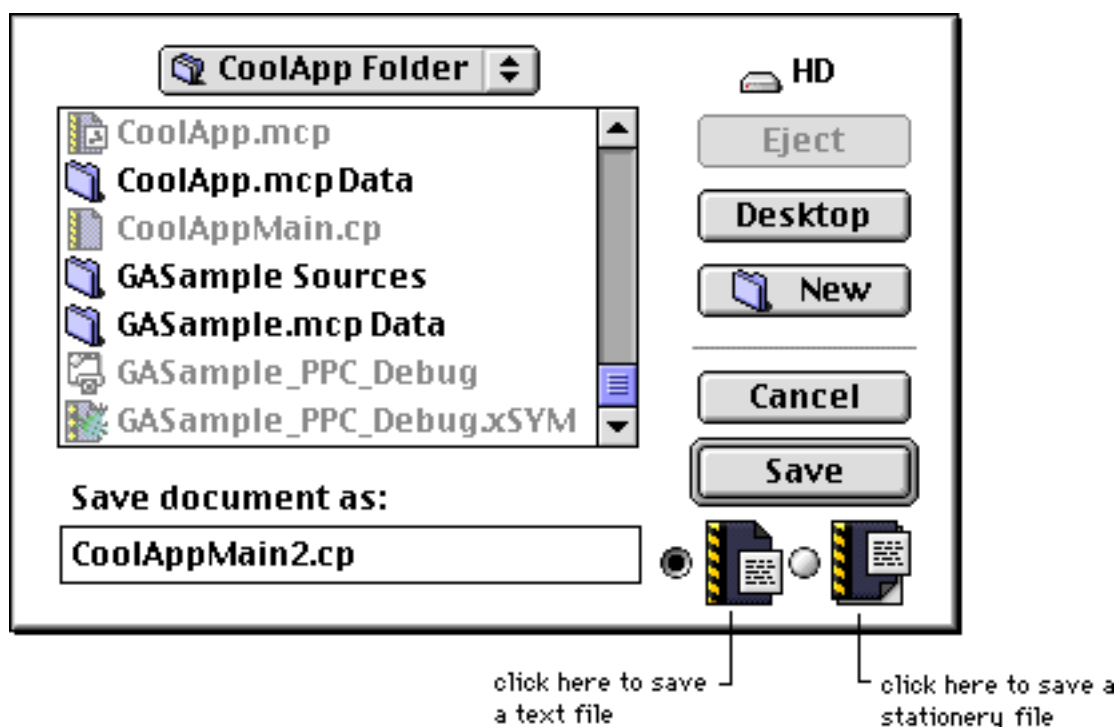
The CodeWarrior IDE saves the file and changes the name of the editor window to the name you entered.

If the file is in the current project, the CodeWarrior IDE changes the file's entry in the project to match the saved name. If you don't want to change the project, but still want to save the file, you can refer to the following section, "[Backing up files.](#)"

Backing up files

If you want to save a backup copy of a text file before you make some changes to the original, use the [Save A Copy As](#) command in the [File Menu](#). The CodeWarrior IDE creates a copy of the file under a new name that you specify, but leaves the original file unchanged and does not change the currently-open project to use the new file name.

Figure 4.7 Save As dialog box (Mac OS)



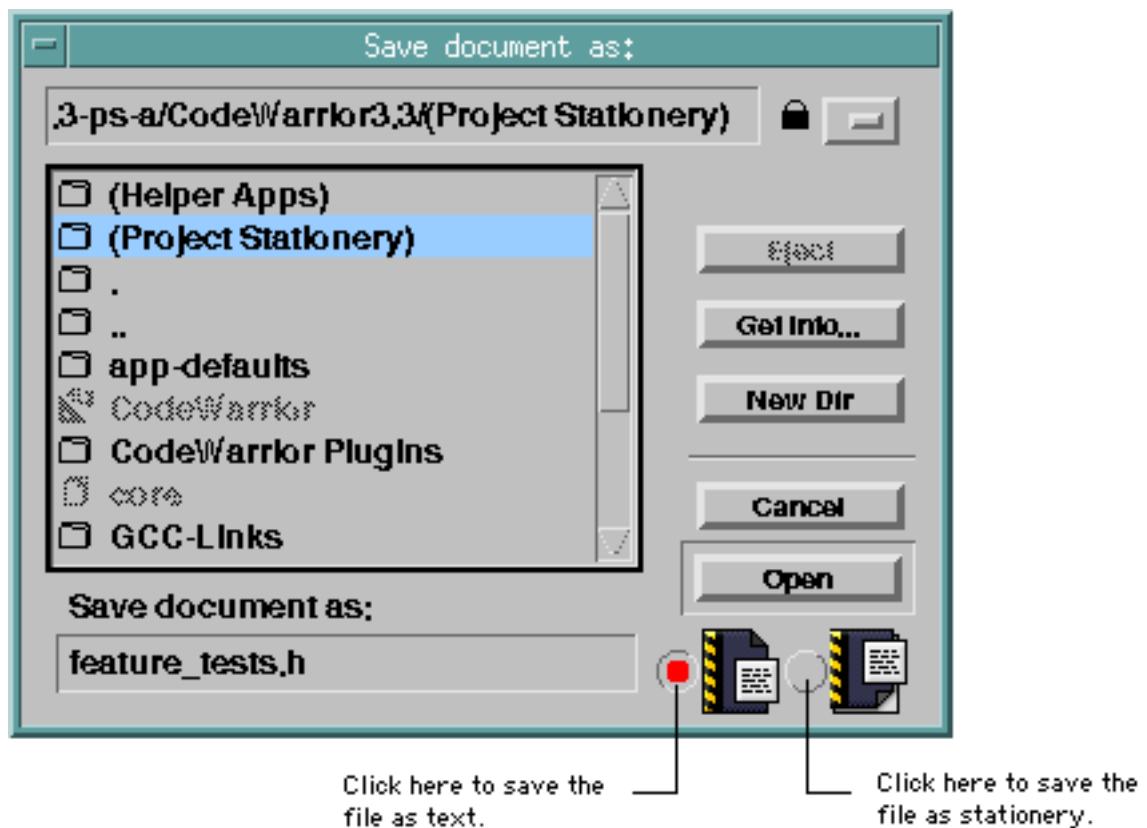
After choosing [Save A Copy As](#) from the [File Menu](#), the CodeWarrior editor displays the dialog shown in [Figure 4.6](#) (Windows), [Figure 4.7](#) (Mac OS), or [Figure 4.8](#) (Solaris). Specify the file's new location and choose a unique name for the file.

Mac OS and Solaris You can also choose whether to save the file as text stationery or a text file.

Now click **Save** and CodeWarrior saves a version of the file with your new name. It does not change the file in the editor window or in the current project.

If the project window is the active window, [Save A Copy As](#) allows you to save the project using a new name, or as a text file. You decide which type of project to create using the **Save Project As Type** pop-up menu shown in [Figure 4.9](#). If you save the project as a text file, that text file will contain the names of all the files in the project.

Figure 4.8 Save As dialog box (Solaris)



Saving as a MS-DOS, Mac OS, or UNIX text file

When you open a text file originally created in a Windows, Mac OS, or UNIX text editor, CodeWarrior internally converts the text file to be compatible with the host platform and corrects inconsistent line endings. When you finish editing the file, CodeWarrior saves the file in its original format.

To learn about saving a text file under a different text format, see [“Options Pop-Up Menu” on page 142.](#)

Figure 4.9 Saving a copy of a project (Windows)

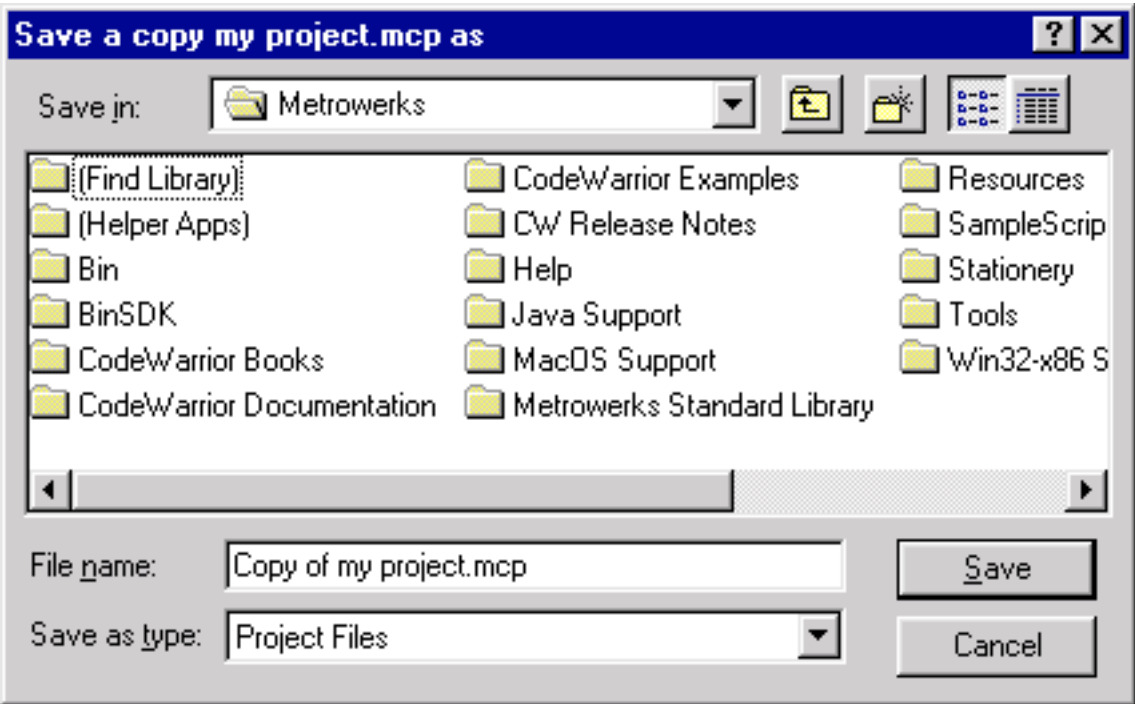


Figure 4.10 Saving a copy of a project (Mac OS)

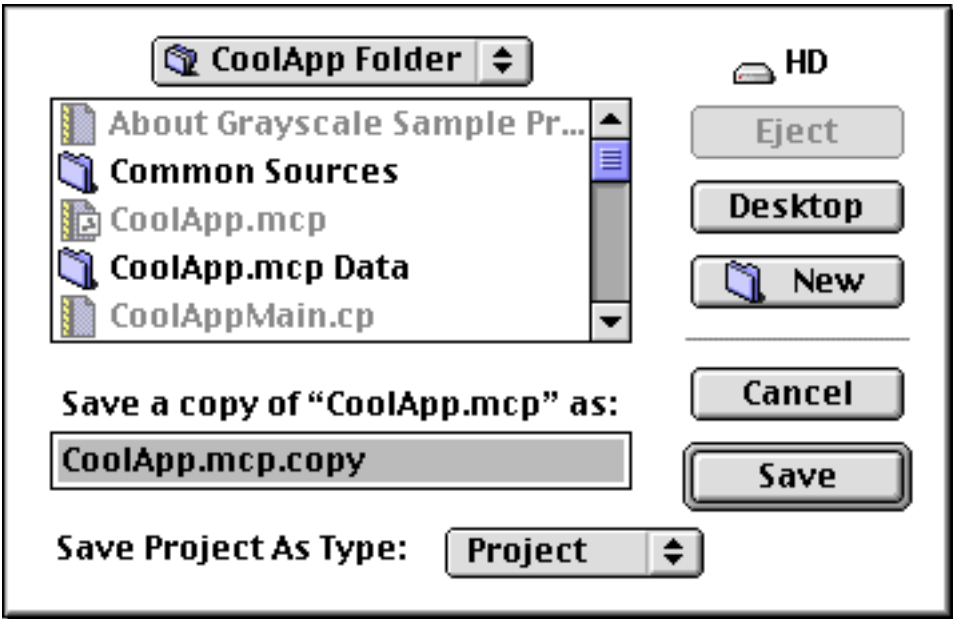
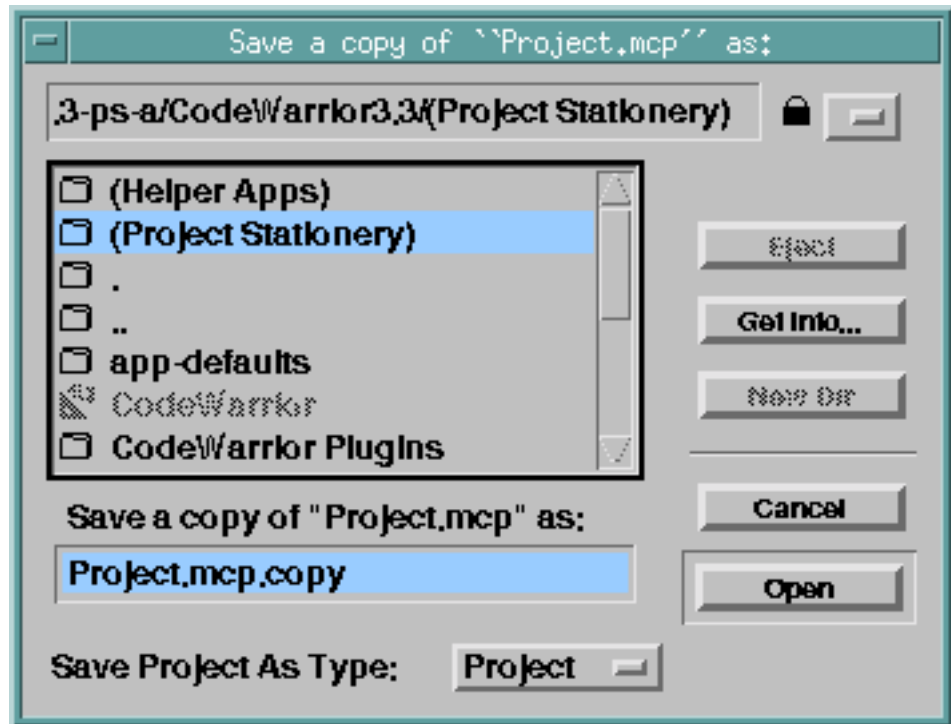


Figure 4.11 Saving a copy of a project (Solaris)



Closing a File

Every editor or project window in the CodeWarrior IDE that you have opened is associated with a file on the hard disk. When you close the window, you close the file. You can close all windows or just a single CodeWarrior IDE window.

The topics in this section are:

- [Closing One File](#)
- [Closing All Files](#)

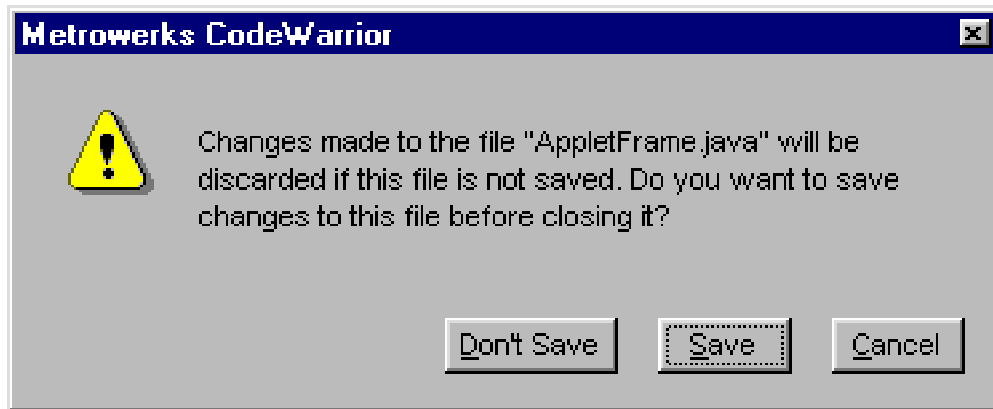
Closing One File

To close a window, choose [Close](#) from the [File Menu](#).

If you close a text file using the [File Menu](#) and have not yet saved your changes, the CodeWarrior IDE asks if you want to save the changes before closing the window, as shown in [Figure 4.12](#). If you

choose to close the file without saving your changes, all changes are lost.

Figure 4.12 The dialog box for unsaved changes



Another way to close a window is by clicking the close box of the active window. This is exactly the same as choosing the [Close](#) command in the [File Menu](#).

Closing an active project window automatically saves the project, and you will not see the dialog shown in [Figure 4.12](#). For more on saving project files, consult ["Saving a Project" on page 72](#).

The [Close](#) command also saves other properties of the window, such as the size, location, and the selected text in the active window. Refer to ["Editor Settings" on page 273](#) for information on how to configure these options. If the appropriate options are enabled, the next time the source code file is opened, it will occupy the same position on your screen and the same text will be selected.

Mac OS The [Close](#) command saves the window position information in a format that is compatible with the Macintosh Programmer's Workshop (MPW). If you open a saved CodeWarrior editor file with MPW, the same window parameters are respected. For more information about MPW, refer to the documentation on the CodeWarrior Reference CD.

Closing All Files

To close all open windows, use the **Close All** command. If you modified any file during the edit session, the editor prompts you to save information before closing each window that contains changes.

[Close All](#) only closes editor and debugger windows. The Find dialog box, as well as project windows, remain open.

Windows This command is available by pressing Ctrl-Shift-W.

Mac OS This command is available when you press the Option key before clicking on the **File** menu.

TIP To close all editor windows at once, press Alt/Option and click on the close box of an editor window.

Printing a File

Use the print options in the CodeWarrior IDE to print open files, a project file, or the contents of a window.

The topics in this section are:

- [Setting Print Options](#)
- [Printing a Window](#)

Setting Print Options

To configure printing options, choose the printer setup appropriate for your platform. CodeWarrior displays the appropriate printer dialog box. Choose [Print Setup \(Windows\)](#) or [Page Setup \(Mac OS\)](#) from the [File Menu](#) to display the printer dialog box.

Use this dialog box to select the paper size, orientation, and other settings. The specific settings and options depend on the printer you have connected to your computer. For more information on using your printer, consult the documentation packaged with your computer and printer.

If you Click **OK**, CodeWarrior saves the options for the next time you print any files.

Printing a Window

To print a window, make the window active and choose the [Print](#) command from the [File Menu](#). This menu command allows you to print some or all of the active window.

When you choose this command, the CodeWarrior IDE displays the print dialog box for your printer. There are two additional CodeWarrior-specific options available for configuration in this dialog box. Depending on your printer and printer software, these options may be displayed in various places in different print dialog boxes.

The CodeWarrior-specific options are:

- [Print Selection Only](#)
- [Print using Syntax Highlighting](#)

Print Selection Only

If there is selected text in the editor window you are printing, the **Print Selection Only** option appears. When this option is on, the CodeWarrior IDE prints only the selected text in the window, not the entire file. When this option is off, the CodeWarrior IDE prints the entire file.

Print using Syntax Highlighting

When the **Print using Syntax Highlighting** option is on, the CodeWarrior IDE prints the file with syntax coloring. On a black and white printer, the colors come out as shades of gray. When the Print using Syntax Highlighting option is off, the CodeWarrior IDE prints the file in black and white without syntax coloring.

TIP (Mac OS) To print color syntax-highlighted text in **bold** and comments in *italics*, choose Black & White printing from the Print dialog box and turn on Print using Syntax Highlighting.

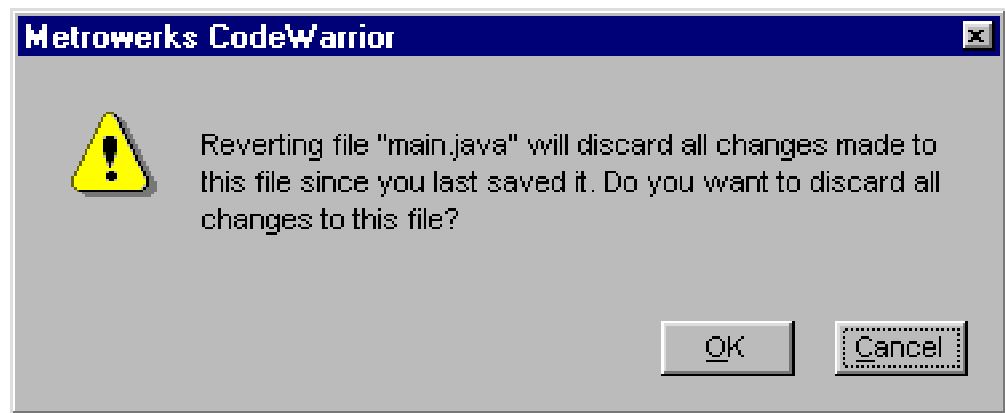
When you are finished configuring printing parameters, click the **Print** button in the printer dialog box. The CodeWarrior IDE then spools the file to your printing software for printing.

For information on other options in the print dialog box, see the documentation that came with your printer or printer software.

Reverting to a Previously-Saved File

If you've opened a text file and started editing it, then realize that you don't want to use the changes you've made, use the **Revert** command on the [File Menu](#). When you select this command the dialog box shown in [Figure 4.13](#) appears.

Figure 4.13 **Revert to a Previous File**



If you click the **OK** button, the last copy of the file you're working with will be opened, and all changes you have made since the last time you saved the file are lost. If you click **Cancel**, the file you're working with is not changed or saved to disk, and you can continue editing it.

Comparing and Merging Files & Folders

The CodeWarrior IDE has a facility to compare two text files, mark the differences between the files, and apply changes between the files. In addition, you can compare the contents of two folders.

The topics in this section show you how to use the IDE's file comparison features:

- [File Comparison and Merge Overview](#)

- [Choosing Files To Compare](#)
- [Examining and Applying Differences](#)
- [Choosing Folders To Compare](#)

File Comparison and Merge Overview

The IDE's file comparison window displays two text files and the differences—insertions and deletions— between them. [Figure 4.14](#) shows an example file comparison window. The window has controls to examine, add, and remove the differences between the files. The currently selected difference is shown with a darker color and outlined in black to contrast it from the other differences visible in the window.

The file comparison window has these parts:

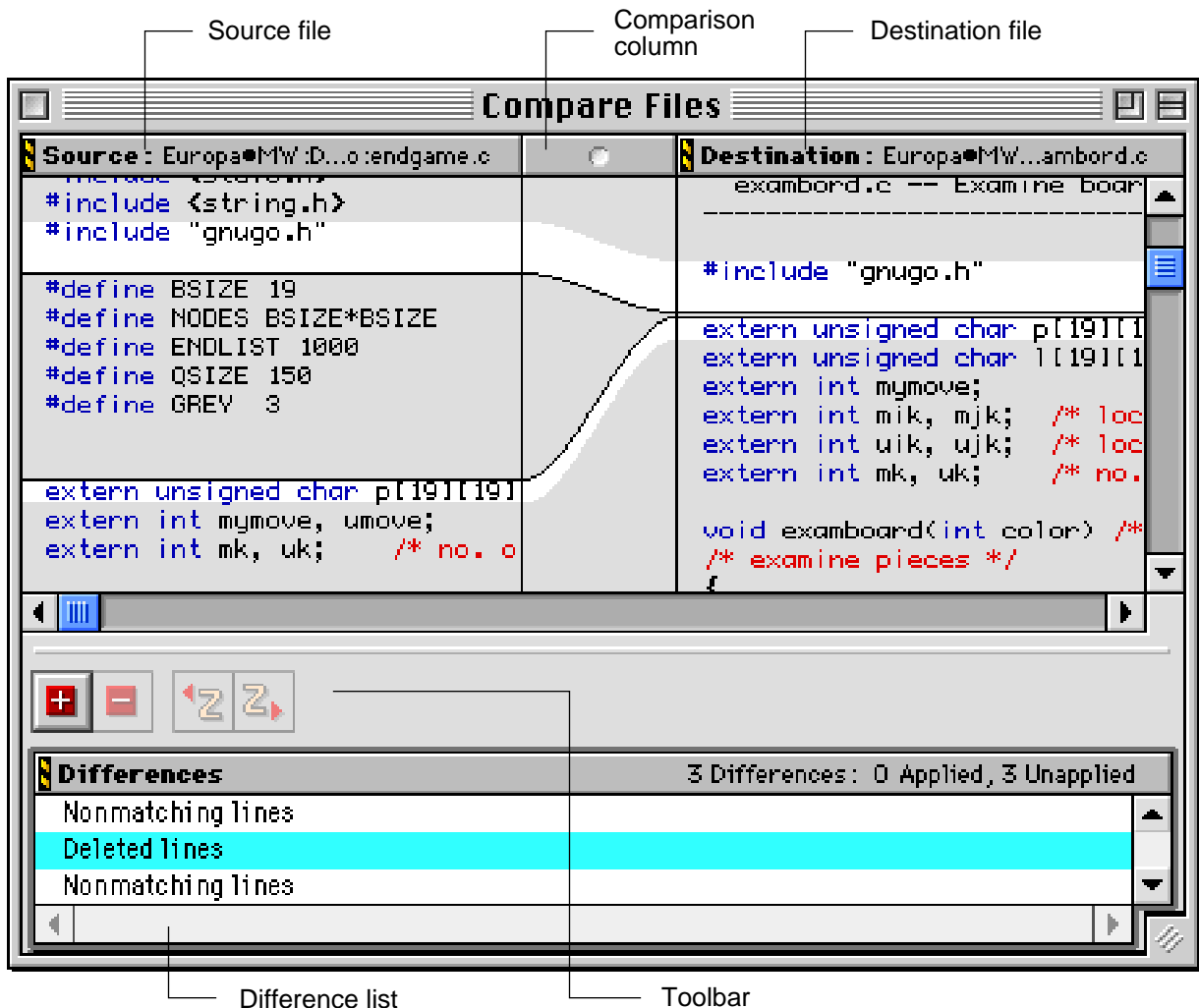
Source file

Displays the source text file that IDE uses as a basis for its comparison with the destination file. This pane appears on the left side of the file comparison window.

Destination file

Displays the source destination file that's compared with the source file. This pane appears on the right side of the file comparison window. Differences between the source file and the destination may be added to or removed from the destination file.

Figure 4.14 The file comparison window



Comparison column

Shows a graphical representation of where text was added or removed between the source and destination files. This column appears between the source and destination panes in the comparison window.

Difference list





Lists the insertions, deletions, and lines of mismatching text between the two files. Selecting an item in the list displays the

difference in the source and destination panes. The comparison column also shows how and where the difference occurs between the two files. Text in the difference list will appear in *italics* when a difference is applied.

Toolbar

Has buttons to apply or remove changes between the two files to the destination file. [Table 4.1](#) describes the toolbar’s buttons. The toolbar also has buttons to undo and redo changes to the source and destination files. For information on configuring toolbars, see [“Customizing the IDE” on page 296](#).

Table 4.1 Common toolbar buttons

Button	Description
	Apply difference
	Unapply difference
	Undo
	Redo

Choosing Files To Compare

To open a file comparison window, choose [Compare Files](#) from the [Search Menu](#) to show a dialog box that prompts you for two files, source and destination files, to compare. To use a file dialog box to browse for the source and destination files, click their respective **Choose** buttons. You may also drag and drop text files into their respective boxes.

Windows Make sure that the **Compare Files** radio button, shown in [Figure 4.15](#), is enabled. This allows you to select files by using the Choose buttons.

Text Compare Options

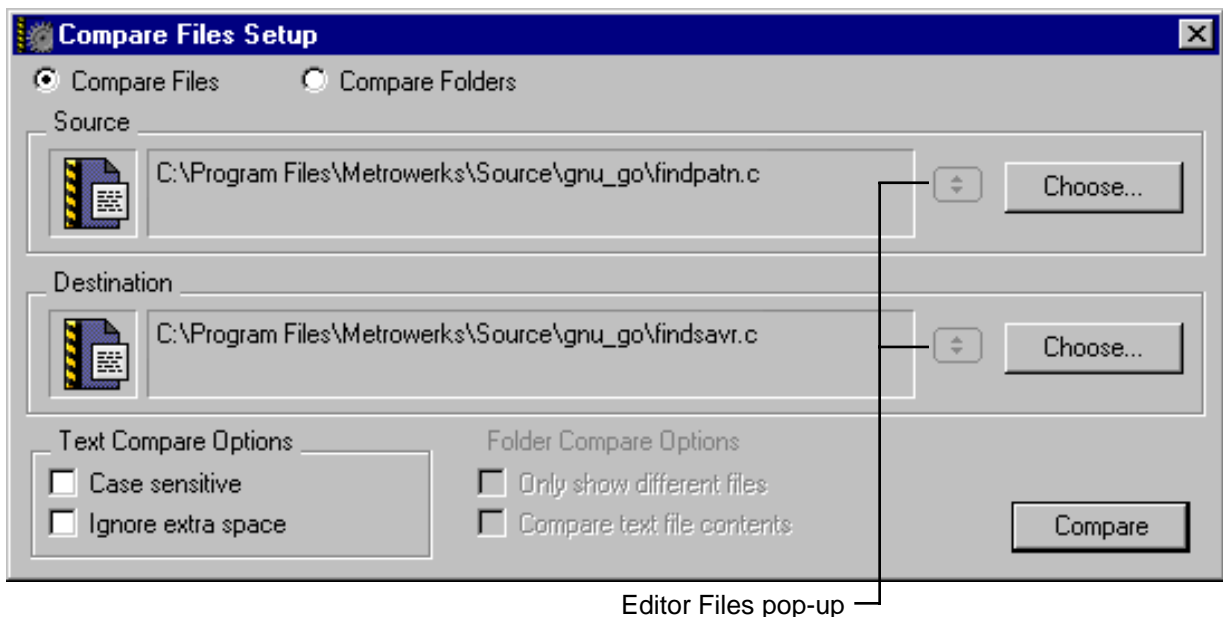
Select the **Case Sensitive** checkbox to consider the case of letters as part of the comparison operation. To ignore the case of letter, deselect this checkbox.

To ignore extra space and tab characters while comparing text, enable the **Ignore Extra Space** checkbox. To take space and tab characters into account, disable the checkbox.

See [“Folder Compare Options” on page 132](#) for information on the folder comparison options.

To show the comparison window after choosing files and setting options, click the **Compare** button.

Figure 4.15 Compare Files Setup dialog box



Examining and Applying Differences

Use the comparison window’s toolbar and difference list to choose among the differences between the source and destination files and apply changes to the destination file.

To view a difference between the two files, click its entry in the difference list. To apply the difference, click the **Apply** button in the toolbar or choose [Apply Difference](#) from the [Search Menu](#). To reverse a difference you've already applied, click the **Unapply** button or choose [Unapply Difference](#) from the [Search Menu](#).

WARNING!

Currently, using the Apply Difference and Unapply Difference buttons erase all actions from the Undo stack. In other words, when you exit the Difference window after apply or unapplying, all undo and redo actions will have been cleared from the Undo stack.

To Compare Editor Files

To compare two files that are already open in editor windows, click on the **Editor Files** pop-up menu next to the source and destination paths, as shown in [Figure 4.15](#). A list of open editor windows appears. Choose a file name from the menu to make it the source or destination file. Click **Compare** once the source and destination files are chosen to display the comparison window.

Choosing Folders To Compare

To open a folder comparison window, choose [Compare Files](#) from the [Search Menu](#) to show a dialog box that prompts you for two folders, the source and destination folders, to compare. To set these folders, drag and drop the folders into their respective boxes ([Figure 4.16](#)).

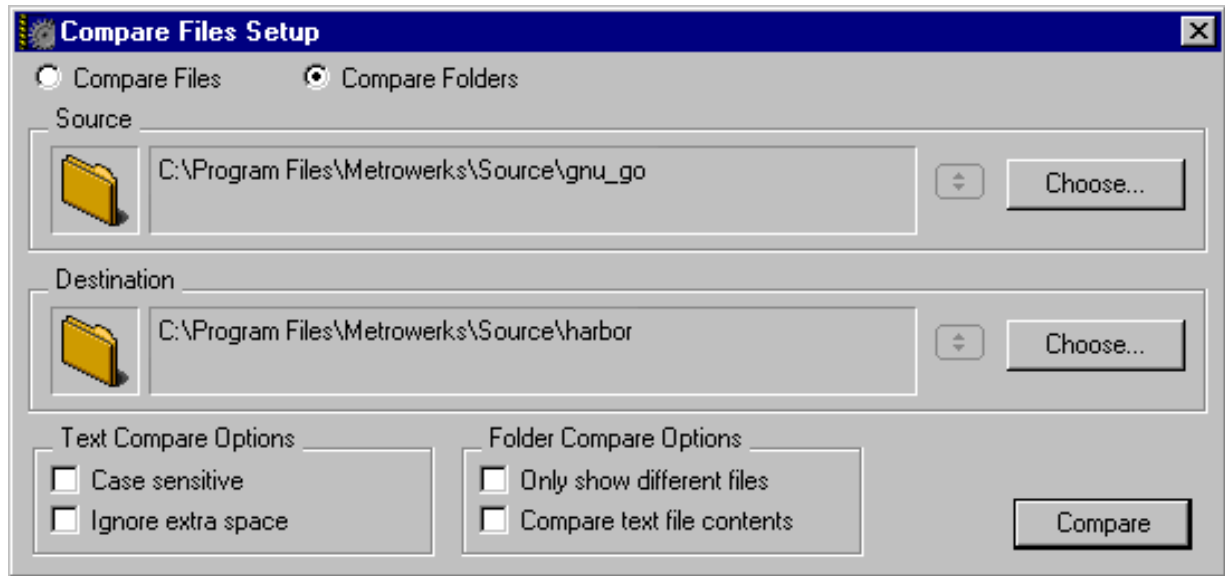
Windows To enable Folder Comparison Options as described in the following section, click the **Compare Folders** radio button. Clicking this radio button also allows you to select folders by using the Choose buttons in the Compare Files Setup dialog box.

Folder Compare Options

Select **Only Show Different Files** to only display files that are different in both folders in the Files In Both Folders list of the Compare Folders window ([Figure 4.17](#)). By default, this option is disabled, so all files in the source and destination folders are displayed.

Comparisons between files in the source and destination folders are normally based upon the file modification dates and file sizes. This is usually good enough to determine if there are differences between the two files. If there are invisible items in the folders, the comparison will skip over those items.

Figure 4.16 Compare Folders Setup dialog box



Select **Compare Text File Contents** to perform a more accurate compare of the files in the two folders. In essence, this performs a **Compare Files** command on every file in the source and destination folders and checks neither the modification dates nor the file sizes. This option is a lot slower since every file has to be opened, but the comparison information is more accurate.

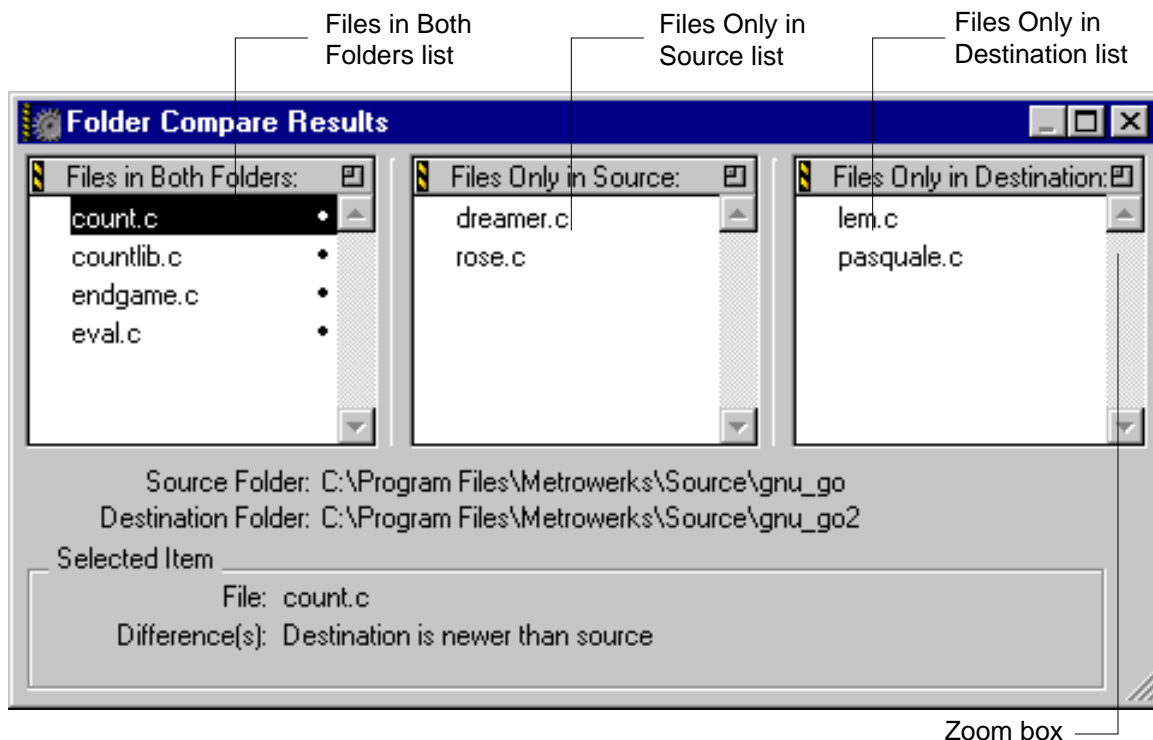
See [“Text Compare Options” on page 131](#) for information on the file comparison options.

When you click the **Compare** button, the IDE displays the Folder Compare Results Window, as shown in [Figure 4.17](#). Source code files, header files, text files, and folders appear in plain face. All other files appear italicized.

The Files In Both Folders list displays all files in both the source and destination folders unless the **Only Show Different Files** option is enabled. Files that are different in the two folders have a small bullet positioned to the right of their name.

When you click on a file in the Files In Both Folders list, Files Only In Source list, or Files Only In Destination list, specific information about the selected file appears in the Selected Item box at the bottom of the folder comparison window.

Figure 4.17 Folder Compare Results window



Double-click on a file in the Files In Both Folders list to open a Compare Files window for resolving the differences between the two differing files.

The Files Only In Source list displays all the files that appear only in the source folder while the Files Only In Destination list displays only files that appear within the destination folder.

You can click on a zoom box for any of the three lists to expand them to fill the window. Click again to collapse back to their original size.

Working with Files

Comparing and Merging Files & Folders

Editing Source Code



This chapter explains how to use the CodeWarrior IDE text editor to edit your source code.

The CodeWarrior editor is a full-featured text editor specially designed for programmers, with the following features:

- Pop-up menus on every editor window, useful for opening interface files and quickly navigating through routines
- Syntax coloring that formats source code for easy identification of comments and keywords
- Point-and-click access to on-line reference material for routines

This chapter covers the following topics:

- [Guided Tour of the Editor Window](#)
- [Editor Window Configuration](#)
- [Basic Text Editing](#)
- [Navigating the Text](#)
- [Online References](#)

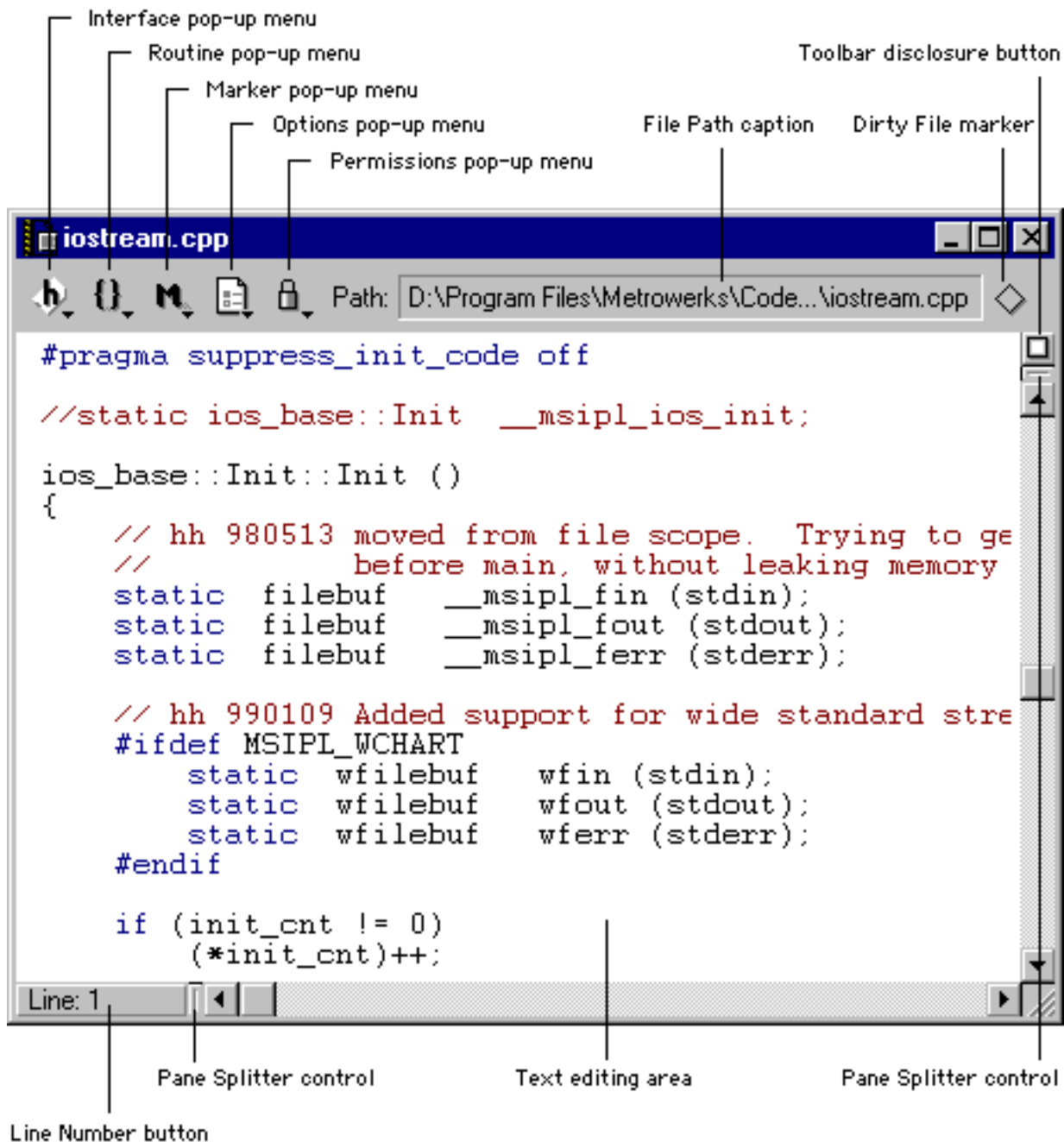
You can also customize options that affect the way the CodeWarrior editor works. To learn more about this topic, refer to [“Editor Settings” on page 273](#).

Guided Tour of the Editor Window

The CodeWarrior editor window, shown in [Figure 5.1](#), contains useful elements for viewing and editing text files.

To open a new editor window, create a new text file by choosing the [New Text File](#) command from the [File Menu](#).

Figure 5.1 The editor window



The sections that follow describe the elements of the editor window shown in [Figure 5.1](#).

- [Text Editing Area](#)

- [Interface Pop-Up Menu](#)
- [Routine Pop-Up Menu](#)
- [Marker Pop-Up Menu](#)
- [Options Pop-Up Menu](#)
- [VCS Pop-Up Menu](#)
- [File Path Caption](#)
- [Dirty File Marker](#)
- [Pane Splitter Controls](#)
- [Line Number Button](#)
- [Toolbar Disclosure Button](#)
- [Path Pop-Up Menu \(Mac OS\)](#)

Text Editing Area

The text editing area of the editor window is where you enter and edit text.

You can select and drag text out of an editor window to any destination that can accept a drop, such as another open editor window. You can also drag selected text into an editor window from other applications that support drag-and-drop.

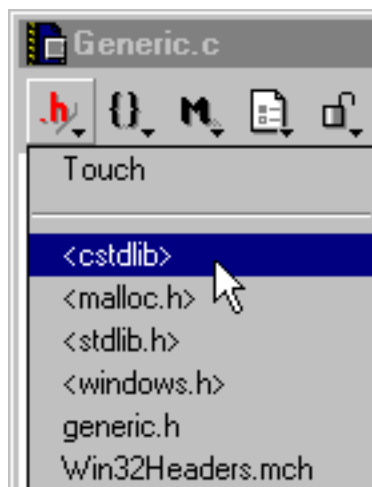
For more information about drag-and-drop operations with text, see [“Moving Text \(Drag and Drop\)” on page 154](#).

Interface Pop-Up Menu



Use the Interface pop-up menu shown in [Figure 5.2](#) to open interface or header files referenced by the current file. You can also choose the **Touch** and **Untouch** commands from this pop-up menu.

Figure 5.2 The Interface pop-up menu



To open a file in the list, choose the file name. Note that in order to see a list of files in the menu, the project file must be open. Note also that some files cannot be opened, such as precompiled header files and libraries.

For more information on opening files, see [“Opening an Existing File” on page 110](#).

To recompile your file the next time the IDE builds the project, choose the **Touch** command. If you click the Interface pop-up menu again, you can deselect the file for compilation with the **Untouch** command in the menu.

To learn more about touching files, see [“Touching and Untouching Files” on page 89](#).

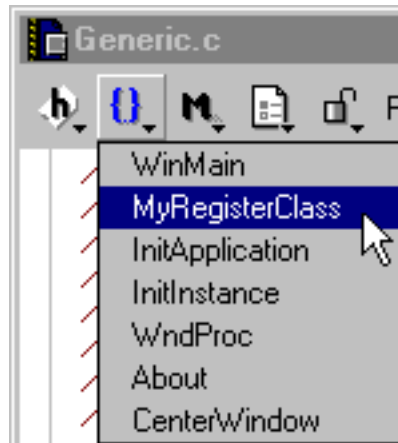
Routine Pop-Up Menu



Use the Routine pop-up menu shown in [Figure 5.3](#) to quickly navigate to specific routines in the current source-code file.

If the pop-up menu is empty, the file shown in the text editing area is not a source-code file. A bulleted routine in the pop-up menu indicates the current location of the text-insertion point.

Figure 5.3 The Routine pop-up menu



NOTE By default, the pop-up menu lists routines in order of appearance in the source-code file. To list routines alphabetically, hold down Ctrl/Option and click the Routine pop-up menu button.

To display routines in alphabetical order by default, enable the [Sort function popup](#) option. See [“Editor Settings” on page 273](#) for more information about editor options.

TIP If you are editing a Pascal file, the Routine pop-up menu shows function names in *italics*, procedure names in plain face, and the main program in **bold**.

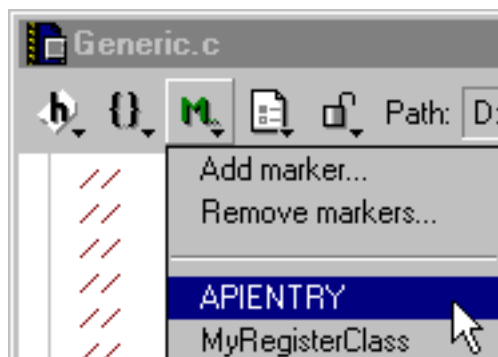
Marker Pop-Up Menu



Use the Marker pop-up menu shown in [Figure 5.4](#) to add and remove markers in your text files. These markers help you access a particular line of text, keep track of where you left off, or note important parts of your text.

For more information on using markers, see [“Adding, Removing, and Selecting a Marker” on page 158](#).

Figure 5.4 The Marker pop-up menu



Options Pop-Up Menu



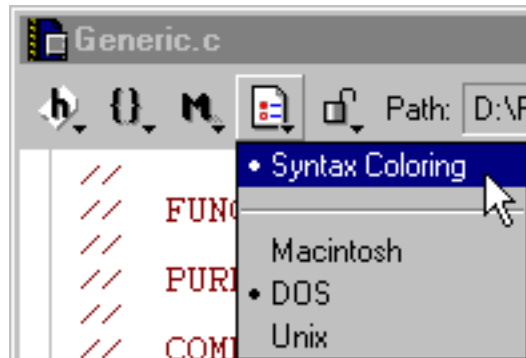
Use the Options pop-up menu, shown in [Figure 5.5](#), to toggle syntax coloring on or off for the current file, and also to set the format for saving a text file.

Choose **Syntax Coloring** from the Options pop-up menu to enable or disable the display of syntax elements in various colors onscreen.

The pop-up menu also lets you choose file formats for saving text files: **Macintosh**, **DOS**, and **UNIX**. A bullet appears next to the current save format for the file shown in the editor window. To save the file in a different format, choose the desired format from the pop-up menu. The CodeWarrior IDE uses the selected format each time you save the text file.

For more information on the Syntax Coloring option shown in this menu, see [“Syntax Coloring” on page 279](#).

Figure 5.5 The Options pop-up menu



VCS Pop-Up Menu

When the current file is part of a revision-control system, the VCS pop-up menu indicates the read/write database status of the file. The VCS pop-up menu icon indicates whether you can modify the file. See [“Common VCS Operations” on page 613](#) for additional details about the icons and their meanings.

Using this pop-up menu, you can get a new copy of your file, checkout the file for modification, make the file writable so you can make changes without doing a checkout, and perform other operations.

For more information about revision-control software, refer to [“Using Version Control Systems” on page 601](#).

File Path Caption



The CodeWarrior IDE automatically displays the directory path of the current file in the File Path caption, which is at the top right of the window shown in [Figure 5.1 on page 138](#).

Mac OS To learn about another way to determine file paths, refer to [“Path Pop-Up Menu \(Mac OS\)” on page 145](#).

Dirty File Marker

The Dirty File marker tells you if the file displayed in a window has been modified since it was last saved or opened. [Table 5.1](#) details the states of the Dirty File marker.

Table 5.1 **Dirty File marker states**

State	Description
	Unchanged file
	Modified and unsaved file ("dirty")

Pane Splitter Controls

Pane Splitter controls divide an editor window into panes so you can view different portions of a file in the same window.

You use these controls to adjust the sizes of the panes after creating them. [Figure 5.8 on page 147](#) shows an editor window with multiple panes.

For more information on this topic, see [“Splitting the Window into Panes” on page 147](#).

Line Number Button

The Line Number button ([Figure 5.1](#)) shows the number of the line that contains the text-insertion point. You also can use this button to go to another line in the file.

For information about moving the text-insertion point to another line, see [“Going to a Particular Line” on page 161](#).

Toolbar Disclosure Button

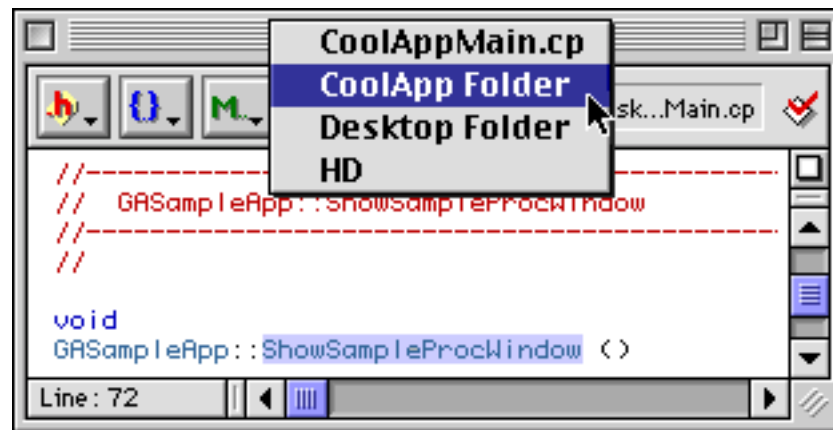
The Toolbar Disclosure button toggles the display of the editor-window toolbar. If the toolbar is hidden, a row of smaller controls appears at the bottom of the editor window (see [Figure 5.7 on page 146](#)).

For more information on using the Toolbar Disclosure button, refer to [“Seeing Window Controls” on page 146](#).

Path Pop-Up Menu (Mac OS)

To see the directory path of the file in the active editor window, press the Command key and click on the name of the file in the title bar of the window, as shown in [Figure 5.6](#). You can also directly open a folder by choosing its name from the pop-up menu.

Figure 5.6 File Path pop-up in the window title bar



Editor Window Configuration

The IDE allows you to customize the way the editor displays your source code. This section covers the following topics:

- [Setting Text Size and Font](#)
- [Seeing Window Controls](#)
- [Splitting the Window into Panes](#)
- [Saving Editor Window Settings](#)

To learn about configuring the editor window toolbar, see [“Customizing the IDE” on page 296](#).

Setting Text Size and Font

The Font & Tabs preference panel controls the size and font used to display text in an editor window. For more information on this topic, refer to [“Font & Tabs” on page 277](#).

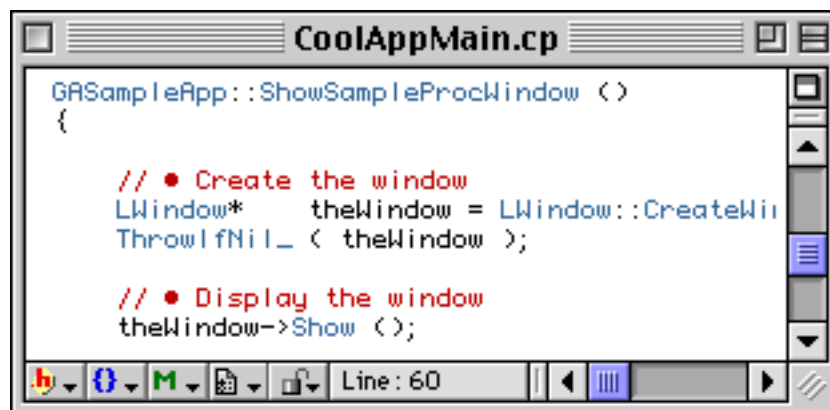
Seeing Window Controls

The row of pop-up menus and controls that appears along the top of the editor window is called the toolbar. Click the [Toolbar Disclosure Button](#), shown in [Figure 5.1 on page 138](#), to show or hide the toolbar.

If you hide the toolbar, the default pop-up menu controls appear along the bottom of the editor window, as shown in [Figure 5.7](#). Note that the [File Path Caption](#) is not visible in this arrangement.

NOTE If you hide a customized editor window toolbar, custom items do not appear at the bottom of the window. The default toolbar items always appear when the toolbar is hidden. When you show the toolbar again, the custom configuration reappears. For information on toolbars in general, and customizing them, see [“Customizing the IDE” on page 296](#).

Figure 5.7 Pop-ups along the editor window bottom



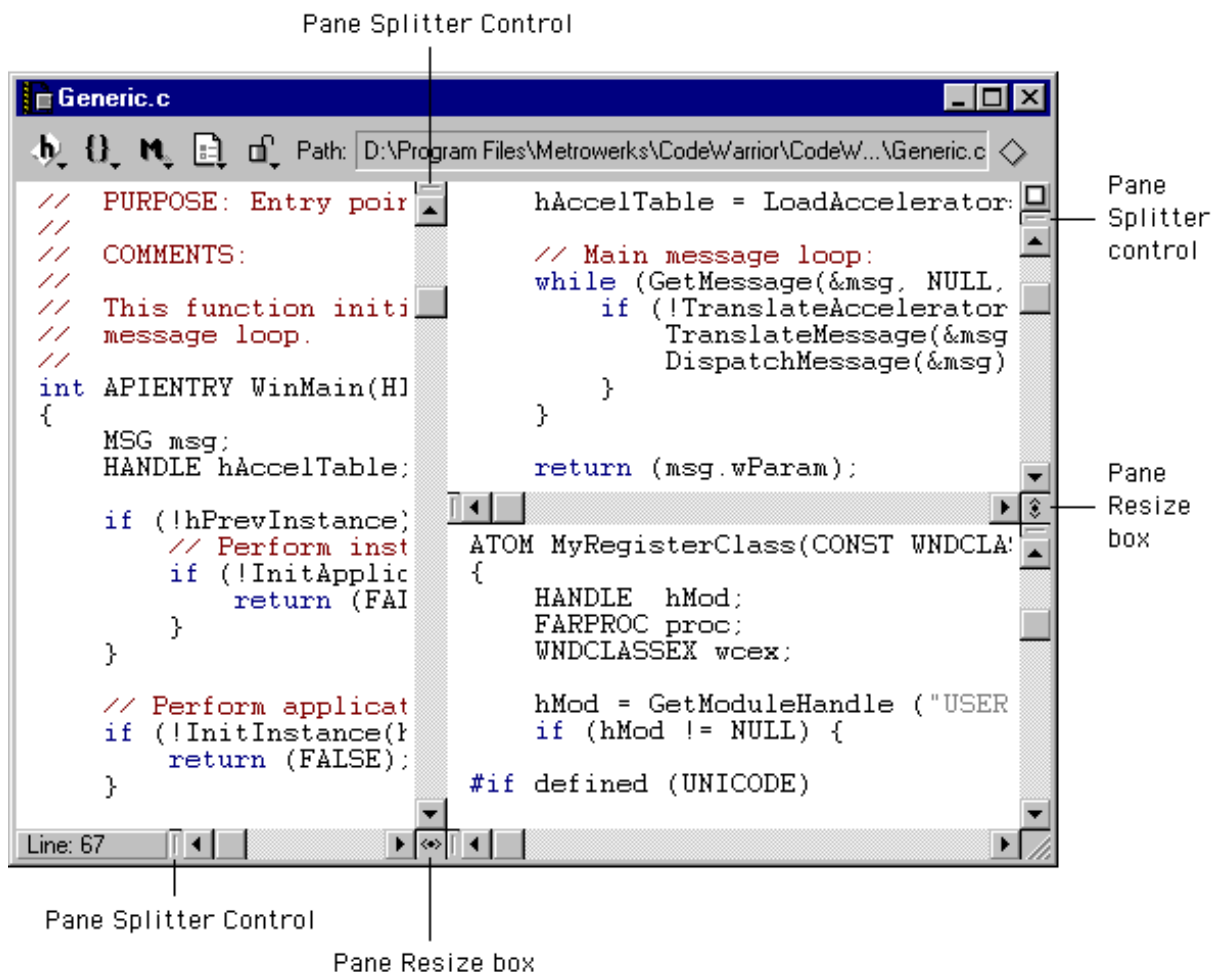
To show the toolbar along the top of the editor window again, click the Toolbar Disclosure Button once more.

You can choose a default setting to display or hide the toolbar in editor windows. To do so, see [“Toolbar Disclosure Button” on page 144](#) for more information. See also [“Showing and hiding a toolbar” on page 315](#).

Splitting the Window into Panes

You can split the editor window into panes to view different parts of a file in the same window, as shown in [Figure 5.8](#). This section describes creating, adjusting, and removing multiple panes.

Figure 5.8 Multiple panes in a window





Creating a new pane

- To create a new pane in an editor window, click and drag a Pane Splitter control. Pane Splitter controls are located on each scroll bar of a pane in the editor window, on the top right and bottom left sides.

As you drag a Pane Splitter control, a gray focus line tracks your progress and indicates where the new pane appears. When you release the mouse button, the editor creates a new pane.



You can also double-click the Pane Splitter control to split a pane into two equal parts.

Resizing a pane

  To change the sizes of the panes in an editor window, click and drag the Pane Resize boxes.

As you drag a resize box, a gray focus line indicates your progress. When you release the mouse button, the editor redraws the panes in their new positions.

Removing a pane

  To remove a pane from an editor window, click and drag a Pane Resize box all the way to the edge of a window.

As you drag the Pane Resize box, a gray focus line indicates your progress. If you drag the box near the edge of the window, the focus line disappears. If you release the mouse button at that time, the editor removes one of the panes from the window.

You can also double-click a Pane Resize box to remove a pane.

Saving Editor Window Settings

The IDE saves the current settings of an editor window whenever you close a window or toggle the display of the toolbar.

The saved settings include the size and location of the window, and the current toolbar display setting. When you reopen the same editor window at a later time, the IDE uses these saved settings.

WARNING!

(Mac OS) If you are using a revision-control system for your source code files that uses native file locking, you must have your file checked out in order to save settings. If you are using `ckid` resources to lock the files, the IDE saves the information without checking out the file.

To learn more about configuring editor window settings, refer to [“Font & Tabs” on page 277](#).

To learn about saving default settings for browser windows, see [“Saving a Default Browser” on page 251](#).

Basic Text Editing

The CodeWarrior IDE provides several aids for editing source-code files, all of which are described in the topics that follow.

This section discusses the following topics:

- [Basic Editor Window Navigation](#)
- [Adding Text](#)
- [Deleting Text](#)
- [Selecting Text](#)
- [Moving Text \(Drag and Drop\)](#)
- [Using Cut, Copy, Paste, and Clear](#)
- [Balancing Punctuation](#)
- [Shifting Text Left and Right](#)
- [Undoing Changes](#)
- [Controlling Color](#)

Basic Editor Window Navigation

The CodeWarrior IDE gives you several ways to move the text-insertion point in a file. Scrollbar navigation and keyboard navigation are the most often-used ways to move around files.

Scrollbar navigation

Like any other text editor, the CodeWarrior editor lets you adjust the scroll bars in the window to view different parts of your text.

You can modify the way that scrolling behaves in the editor windows by changing the [Dynamic scrolling](#) option. To learn how to do this, refer to [“Dynamic scrolling” on page 275](#).

Keyboard navigation

[Table 5.2](#) describes how to move the insertion point around in a file by pressing default key combinations on your keyboard. These keyboard combinations are also called “key bindings” in CodeWarrior terminology.

For information on Solaris modifier key settings, refer to the section entitled [“Special note for Solaris users” on page 23](#).

Table 5.2 Text navigation with the keyboard

To move the insertion point to the...	Windows	Mac OS	Solaris
Previous word	Ctrl-Left Arrow	Option-Left Arrow	Alt-Left Arrow
Next word	Ctrl-Right Arrow	Option-Right Arrow	Alt-Right Arrow
Beginning of the line	Home	Command-Left Arrow	Meta-Left Arrow
End of the line	End	Command-Right Arrow	Meta-Right Arrow
Beginning of the file	Ctrl-Home	Command-Up Arrow	Meta-Up Arrow
End of the file	Ctrl-End	Command-Down Arrow	Meta-Down Arrow

[Table 5.3](#) describes how to scroll to different locations in a file, without moving the insertion point.

For information on Solaris modifier key settings, refer to the section entitled [“Special note for Solaris users” on page 23](#).

Table 5.3 Scroll with the keyboard

To move the insertion point to the...	Windows	Mac OS	Solaris
Previous page	Page Up	Page Up	Page Up
Next page	Page Down	Page Down	Page Down
Beginning of the file	Ctrl-Home	Home	Home
End of the file	Ctrl-End	End	End
Previous line	Ctrl-Up Arrow	Control-Up Arrow	Control-Up Arrow
Next line	Ctrl-Down Arrow	Control-Down Arrow	Control-Down Arrow

Adding Text


To add text to an open file, click once in the [Text Editing Area](#) of the window. After an insertion point appears at location you clicked, you can begin typing on the keyboard to enter text.

To read about different ways to move the insertion point in an editor window, see [“Basic Editor Window Navigation” on page 149](#).

Deleting Text

There are several different ways to delete text.

To delete text that you just typed, press the Backspace/Delete key.

To delete text that is in front of the text-insertion point, use the Delete/Del () key.

To delete more than one contiguous character at a time, select the text you want to delete and press the Backspace/Delete key.

To learn about selecting text, read the next section, [“Selecting Text”](#).

Selecting Text

There are several different ways to select text in an editor window.

You can select a word, a line, or a range of text. You can also select text by holding down the Shift key while pressing most of the key bindings listed in [Table 5.2](#).

To select a word:

- double-click on the word

To select a line:

- triple-click anywhere in the line
- move the mouse pointer to the left edge of the editor window so that the mouse pointer points to the right, then click

This selection method is available when the [Left margin click selects line](#) option is enabled in the [Editor Settings](#) preference panel.

To select a range of text:

- hold down the mouse button, drag the mouse pointer over the contiguous range of text that you want to select, and then release the mouse button
- set the text-insertion point to mark the beginning of your selection, then press the Shift key while clicking the place in your text where you want the selection to end
- move the mouse pointer to the left edge of the editor window so that the mouse pointer points to the right, then click and drag the mouse pointer up or down to select lines of text

This selection method is available when the [Left margin click selects line](#) option is enabled in the [Editor Settings](#) preference panel.

Mac OS and Solaris The editor can select parts of text identifiers by holding down the Control key while using the left or right arrow keys, or when double-clicking. For example, double-clicking between the two “m” characters in `FindCommandStatus()` would result in the word `Command` being selected.

To select an entire routine in the editor window, press the Shift key while selecting the name of the routine from the [Routine Pop-Up Menu](#). This technique is particularly useful for copy and paste operations and for using drag and drop to move code around in your file.

[Table 5.4](#) describes how to select text by using the keyboard, starting at the current insertion point.

For information on Solaris modifier key settings, see [“Special note for Solaris users” on page 23](#).

Table 5.4 Text selection with the keyboard

To select text to the...	Windows	Mac OS	Solaris
Previous word	Ctrl-Shift-Left Arrow	Option-Shift-Left Arrow	Alt-Shift-Left Arrow
Next word	Ctrl-Shift-Right Arrow	Option-Shift-Right Arrow	Alt-Shift-Right Arrow
Beginning of the line	Shift-Home	Shift-Command-Left Arrow	Shift-Meta-Left Arrow
End of the line	Shift-End	Shift-Command-Right Arrow	Shift-Meta-Right Arrow
Beginning of the page	Shift-Page Up	Option-Shift-Up Arrow	Alt-Shift-Up Arrow
End of the page	Shift-Page Down	Option-Shift-Down Arrow	Alt-Shift-Down Arrow
Beginning of the file	Ctrl-Shift-Home	Shift-Command-Up Arrow	Shift-Meta-Up Arrow
End of the file	Ctrl-Shift-End	Shift-Command-Down Arrow	Shift-Meta-Down Arrow

For more information about using drag-and-drop with text, see [“Moving Text \(Drag and Drop\)”](#) in the next section.

You can also quickly select blocks of code by using the Balance command. To learn how to do this, refer to [“Balancing Punctuation” on page 154](#).

Moving Text (Drag and Drop)

To move selected text to a new location, you can use the drag-and-drop features of the editor. To learn more about enabling and disabling this feature, refer to [“Editor Settings” on page 273](#).

The CodeWarrior editor can also accept text items from other applications that support drag and drop. Refer to the documentation that came with the other applications for information about drag-and-drop support.

TIP (Mac OS) You can also copy selected text to a new location. To perform this operation, hold down the Option key while dragging the text. After you release the mouse button, the editor places a copy of the selected text at the new location.

Using Cut, Copy, Paste, and Clear

Most computer applications support a standard set of menu commands, such as **Cut**, **Copy**, and **Paste**. In the CodeWarrior IDE, these commands appear in the Edit menu.

Use these commands to manipulate text within a window, between windows, or between applications.

For more information about these commands, refer to [“Edit Menu” on page 623](#).

Balancing Punctuation

When you edit source code, you often have to make sure that every parenthesis (()), bracket ([]), and brace ({ }) has a mate, or the compiler could misinterpret your code or give you an error.

The CodeWarrior IDE provides several checks that help you balance these elements correctly.

To check for balanced parentheses, brackets, or braces, place the insertion point in the text you want to check. Then, choose **Balance** from the [Edit Menu](#). Alternatively, double-click on the parenthesis, bracket, or brace character that you want to check for a matching character.

The CodeWarrior editor searches through the text file, starting from the text-insertion point, for a parenthesis, bracket, or brace. After the editor finds one of these characters, it searches through the text file in the opposite direction for a matching character. When the editor finds a match, it selects the text between the two characters. The computer beeps when the punctuation is not balanced. For example, if the insertion point is not enclosed by matching parentheses, brackets, or braces, the computer beeps.

TIP Use the [Balance](#) command to quickly select blocks of code.

Using automatic balancing

You can have the editor check for balanced punctuation automatically. To learn more about this feature, refer to [“Balance while typing” on page 275](#).

Shifting Text Left and Right

Use the [Shift Left](#) and [Shift Right](#) commands on the [Edit Menu](#) to shift a block of text to the left or right.

To shift a selected block of text, choose [Shift Right](#) or [Shift Left](#) from the [Edit Menu](#). If you do not know how to do select text, refer to [“Selecting Text” on page 152](#).

The CodeWarrior editor shifts the selected text one tab stop to the right or left by inserting or deleting a tab at the beginning of every line in the selection.

To learn more about controlling the number of spaces used for indenting text, refer to [“Font & Tabs” on page 277](#).

Undoing Changes

The CodeWarrior editor offers several ways to undo mistakes as you edit a file.

Undoing the last edit

The [Undo](#) command reverses the effect of your last action. The name of the Undo command on the [Edit Menu](#) varies, depending on your last action. For example, if you just typed some text, the name of the command changes to **Undo Typing**. Choose this command to remove the text you just typed.

Undoing and redoing multiple edits

When you enable the [Use multiple undo](#) option, you can undo or redo several previous actions.

For instance, if you [Cut](#) a word, then [Paste](#) it, then type some text, you can backtrack through all of those actions by choosing the [Undo](#) command three times. The first Undo removes the text you typed, the second Undo unpastes the text you pasted, and the third Undo uncuts the text you cut, therefore returning the text to its original condition. You can perform the three original activities again in the same order by choosing the **Redo** command three times.

To enable the [Use multiple undo](#) option, refer to [“Use multiple undo” on page 276](#).

Note that the keyboard shortcut for the Redo command changes when the [Use multiple undo](#) option is disabled.

WARNING!

The IDE saves undo actions in a stack, which could cause you to lose certain actions when you use multiple undo and redo. Each undo action adds an item to the stack, while each redo action repositions a pointer to the next undo action. For example, if there were five undo actions on the stack (ABCDE), and you redo two of them, the stack appears to the pointer like this: ABC. When you perform a new action (ABCF), the undo events (DE) are no longer available.

Reverting to the last saved version of a file

The **Revert** command in the File menu returns a file to its last saved version. To learn more about reverting to the previous version of a file, see [“Reverting to a Previously-Saved File” on page 127](#).

Controlling Color

You can use color to highlight many elements in your source code, such as comments, keywords, and quoted character strings. Highlighting these elements helps you identify them in the text, so you can check your spelling and syntax as you type by recognizing color patterns. For information on configuring color syntax options, see [“Syntax Coloring” on page 279](#).

You can also highlight custom keywords, which are in a list of words you designate. See [“Syntax Coloring” on page 279](#) for instructions on configuring the editor to do this for you.

Navigating the Text

The CodeWarrior editor offers flexible means for navigating through text in your source-code files.

This section covers the following topics:

- [Finding a Routine](#)
- [Adding, Removing, and Selecting a Marker](#)
- [Opening a Related File](#)
- [Going to a Particular Line](#)
- [Using Go Back and Go Forward](#)
- [Configuring Editor Commands](#)
- [Opening the Directory of a File \(Mac OS\)](#)

In addition, the integrated source-code browser has many powerful techniques for navigating through your code. To learn more about the CodeWarrior browser, refer to [“Browsing Source Code” on page 207](#).

You can change the key bindings that cause the text-insertion point to move around in a file. Refer to [“Editor Commands” on page 671](#) for more information.

Finding a Routine



Click the icon shown at left to display the Routine pop-up menu, discussed in [“Routine Pop-Up Menu” on page 140](#). This pop-up menu lets you access routines in your code.

NOTE If the Routine pop-up menu is empty, the current file is not a source-code file.

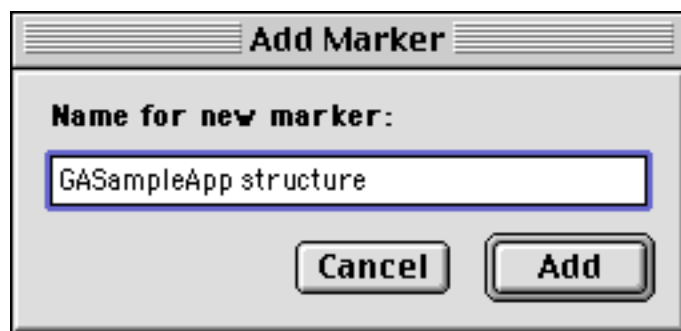
Adding, Removing, and Selecting a Marker

You can add or remove a marker in any of your text files using the facilities built into the CodeWarrior editor. Markers are like bookmarks. They are useful for quickly jumping to specific points in a file, or for leaving notes to yourself about work in progress on your code.

Adding a marker

To add a marker, move the text-insertion point to the location in the text you wish to mark, then choose **Add marker** from the [Marker Pop-Up Menu](#). A dialog box named Add Marker appears, shown in [Figure 5.9](#).

Figure 5.9 Add Marker dialog box

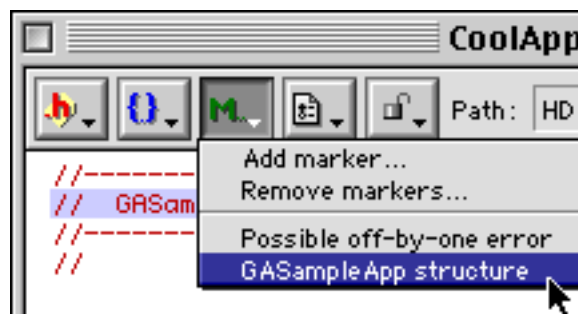


Enter text in the Add Marker dialog box to mark the insertion-point location in the file with a note, comment, routine name, or other information that would be helpful to you.

When you finish entering your text note, click **Add**. The marker text now appears in the [Marker Pop-Up Menu](#), as shown in [Figure 5.10](#).

TIP If you select some text in a source file, then choose **Add Marker**, the selected text appears in the text field of the Add Marker dialog box. This shortcut is handy for quickly adding specific routines or lines as markers.

Figure 5.10 Example text File with a marker added



Adding Markers with #pragma

There is another method for marking files on a more permanent basis. In C/C++ language programs, use

```
#pragma mark myMarker
```

to create a marker. In Pascal, use

```
{ $PRAGMA MAC MARK myMarker }
```

to create a marker.

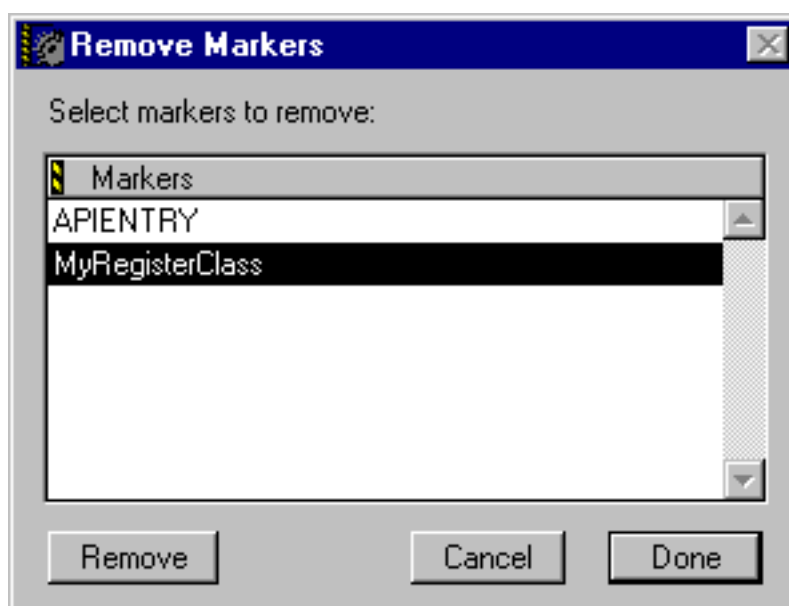
When embedded in your file, this example automatically adds *myMarker* to the [Routine Pop-Up Menu](#) when you open the file in the editor.

Unlike the markers defined using the [Marker Pop-Up Menu](#), markers defined with #pragma statements appear in the Routine pop-up menu.

Removing a marker

To remove a marker, click the [Marker Pop-Up Menu](#) and choose the **Remove markers** command. The dialog box shown in [Figure 5.11](#) appears. After you select the marker you wish to delete, click **Remove** to permanently remove the marker from the list. To close the Remove Markers dialog box, click **Done** to confirm your changes or click **Cancel** to discard your changes.

Figure 5.11 Remove Markers dialog box



Jumping to a marker

To jump to a specific marker in your file, choose the name of the marker from the [Marker Pop-Up Menu](#). The IDE sets the text-insertion point at the location of the marker.

Opening a Related File

The CodeWarrior editor lets you open files that are related to the file in the active editor window. For example, you can view a header file used by a source-code file.

One way to open interface or header files referenced by the current file is to use the [Interface Pop-Up Menu](#), shown in [Figure 5.2 on](#)

[page 140](#). To open a related file, choose its name from the pop-up menu. You can also use the pop-up menu to **Touch** and **Untouch** the current file.

There is another method for opening an interface or header file used by your source code. To open the related file, press Ctrl-D (Windows) or Command-D (Mac OS) after selecting the file name in the active window. To learn more about this technique, refer to [“Opening an Existing File” on page 110](#).

Going to a Particular Line

You can go to a specific line in an editor window if you know its line number. The IDE numbers lines consecutively, with the first line designated as line 1.

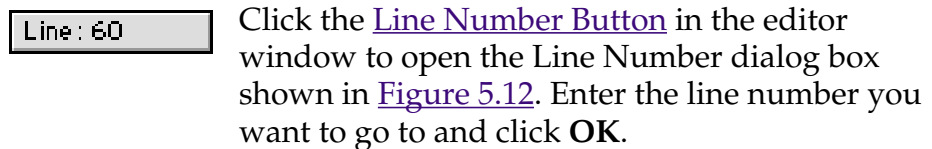
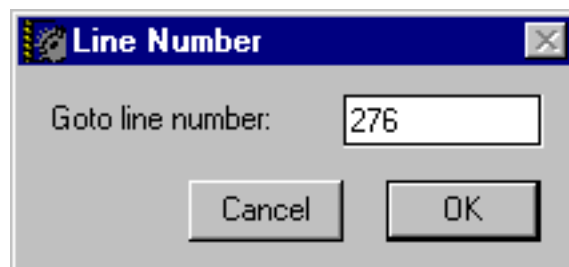


Figure 5.12 Line Number dialog box



Using Go Back and Go Forward

The [Go Back](#) and [Go Forward](#) commands are only available when you use the CodeWarrior browser. If you already enabled the browser, refer to [“Go Back and Go Forward” on page 246](#) for information about using these commands.

For more information about the CodeWarrior browser, see [“Browsing Source Code” on page 207](#).

Configuring Editor Commands

The CodeWarrior IDE lets you customize editor key bindings to suit your working style. Refer to [“Editor Commands” on page 671](#) to learn about the commands you can customize to your liking.

Opening the Directory of a File (Mac OS)

If you want to know the full directory path of the file in the active editor window, use the Path pop-up menu. To display this pop-up menu, press the Command key and click on the file name in the title bar of the editor window.

You can open the directory that contains the file shown in the active editor window by choosing the directory name from the Path pop-up menu, as shown in [Figure 5.6 on page 145](#).

Online References

As you work on a program, you often need to look up the documentation or definition of a particular routine, variable, or type. You might also need to look up documentation about using the IDE. CodeWarrior includes both kinds of documentation.

Before you can use the online help available from the IDE, you need to install the help files and viewer applications from the CodeWarrior Reference CD. Follow the instructions for your particular host:

Windows Insert the CodeWarrior Reference CD into your CD-ROM drive. Wait for a dialog box to appear. Click “Launch CodeWarrior Setup” and follow the onscreen instructions.

Mac OS Insert the CodeWarrior Reference CD into your CD-ROM drive. Double-click the CodeWarrior Reference Installer icon inside the CD. Follow the onscreen instructions to install the help files and viewer applications.

Solaris and Linux Insert the CodeWarrior Tools CD into your CD-ROM drive. Change your current directory to that of the CD-ROM drive. Locate and run the `install.sh` file on the CD, then follow the onscreen instructions.

This section describes how to look up documentation online and how to set up the online reference databases.

The topics in this section are:

- [Finding Symbol Definitions](#)
- [WinHelp \(Windows\)](#)
- [QuickHelp \(Mac OS\)](#)
- [QuickView \(Mac OS\)](#)
- [THINK Reference \(Mac OS\)](#)
- [Inserting Routine \(Reference\) Templates \(Mac OS\)](#)

Finding Symbol Definitions

You can find the definition of a symbol in the source-code files of your project. You can also find symbol definitions by using the online documentation viewer selected in IDE Extras preferences panel. Supported online reference viewers include [WinHelp \(Windows\)](#) and [QuickHelp \(Mac OS\)](#), as well as older online help systems such as [QuickView \(Mac OS\)](#) and [THINK Reference \(Mac OS\)](#).

TIP The CodeWarrior browser also looks up symbol definitions. See [“Using the Browser” on page 244](#) for more information.

To look up the definition of a selected symbol, choose **Find Definition** from the [Search Menu](#). CodeWarrior searches all of the files in your project for the symbol definition.

If CodeWarrior finds a definition, it opens an editor window and highlights the definition for you to examine. If CodeWarrior does not find a definition, a system beep sounds.

TIP To return to your original location after viewing a symbol definition, press Shift-Ctrl/Command-B. This key binding is equivalent to the **Go Back** menu command. For more information, see [“Go Back and Go Forward” on page 246](#).

Mac OS, Solaris, and Linux You can also use the **Find Reference** and **Find Definition & Reference** commands to look up symbol definitions. After you select a symbol and choose the [Find Reference \(Mac OS\)](#) command, CodeWarrior searches the online documentation for the symbol definition (see [“IDE Extras” on page 259](#) for more information). After you select a symbol and choose the [Find Definition & Reference \(Mac OS\)](#) command, CodeWarrior searches both the project files and the online documentation for the symbol definition.

WinHelp (Windows)

The CodeWarrior IDE provides help files for use with the WinHelp viewer application. WinHelp is part of the Windows operating system.

You can use WinHelp to look up documentation on Win32, MFC, the ANSI libraries, and the CodeWarrior IDE. In addition, you can right-click on many components in the IDE to view pop-up help information.

To use WinHelp, choose **CodeWarrior Help** from the Help menu. The help file for the IDE appears.

Click the hypertext links in the WinHelp documentation to learn more about the IDE. To learn how to perform specific tasks, click the **How to** hypertext link in the WinHelp window. Clicking the **Glossary** hypertext link provides descriptions of the terms used in the online documentation.

Refer to [“Finding Symbol Definitions” on page 163](#) to begin looking up documentation online.

QuickHelp (Mac OS)

The CodeWarrior IDE provides help files for use with the QuickHelp viewer application. The CodeWarrior Help folder contains the QuickHelp files.

You can use QuickHelp to look up documentation on PowerPlant, the ANSI libraries, and the CodeWarrior IDE. In addition, you can

use Balloon Help on many components in the IDE to view help information.

To use QuickHelp, choose **CodeWarrior Help** from the Help menu. The help file for the IDE appears.

Click the hypertext links in the WinHelp documentation to learn more about the IDE. To learn how to perform specific tasks, click the **How to** hypertext link in the WinHelp window. Clicking the **Glossary** hypertext link provides descriptions of the terms used in the online documentation.

Refer to [“Finding Symbol Definitions” on page 163](#) to begin looking up documentation online.

QuickView (Mac OS)

QuickView is part of the *Macintosh Programmer's Toolbox Assistant* (MPTA) published by Addison-Wesley for Apple Computer, Inc.

You can use QuickView to look up documentation on PowerPlant, the ANSI libraries, and the CodeWarrior IDE.

If you have the *Macintosh Programmer's Toolbox Assistant*, you can also look up documentation on Mac OS Toolbox routines.

To use QuickView with the CodeWarrior IDE, you must properly configure the IDE to use the QuickView files:

1. **Position your files.**

Place all the QuickView files in the folder that contains the QuickView application. If you already have the Macintosh Programmer's Toolbox Assistant (MPTA), copy all its QuickView files to the same folder.

2. **Remove multiple copies of the QuickView application.**

In order to function properly with CodeWarrior, you cannot have more than one copy of the QuickView application on your computer system. If your computer system includes duplicate copies of QuickView, remove all copies except the one in which you placed the CodeWarrior QuickView files.

3. Select QuickView as the online reference.

Refer to [“IDE Extras” on page 259](#) to learn how to select the QuickView application as the online reference viewer for your project.

Refer to [“Inserting Routine \(Reference\) Templates \(Mac OS\)” on page 166](#) or [“Finding Symbol Definitions” on page 163](#) to begin looking up documentation online.

THINK Reference (Mac OS)

Refer to [“IDE Extras” on page 259](#) to learn how to select the THINK Reference application as the online reference viewer for your project. THINK Reference includes several databases on Mac OS Toolbox routines.

THINK Reference is not part of the CodeWarrior product. This reference viewer is available from third-party sources, such as Developer Depot on the MacTech CD-ROM.

Refer to [“Inserting Routine \(Reference\) Templates \(Mac OS\)” on page 166](#) or [“Finding Symbol Definitions” on page 163](#) to begin looking up documentation online.

Inserting Routine (Reference) Templates (Mac OS)

If you look up a routine (such as an operating system call) in the QuickView or THINK Reference online viewers, you can paste the template for the call into the active editor window at the text-insertion point. This technique is useful when you know the name of the call you want to add to your source code, but you are not familiar with the call parameters.

A routine template looks like this:

```
SetRect(r, left, top, right, bottom);
```

To insert a reference template into your code, type the routine name that you want to insert, then select the name you just typed. Next, choose [Insert Reference Template \(Mac OS\)](#) from the [Edit Menu](#).

The CodeWarrior IDE searches for the routine in either [QuickView \(Mac OS\)](#) or [THINK Reference \(Mac OS\)](#), starting the required application if it is not already running. If CodeWarrior finds the routine, the IDE copies the template to the active editor window and replaces the text you selected with the template. If CodeWarrior does not find the routine template, a system beep sounds.

Searching and Replacing Text



This chapter explains how to use the CodeWarrior IDE facilities to search and replace text in files.

The CodeWarrior IDE provides comprehensive search and replace features with the Find dialog box. You can search for and replace text in a single file, in every file in a project, or in any combination of files. You can also search for regular expressions, like as those used in UNIX's `grep` command.

The topics in this chapter are:

- [Guided Tour of the Find Dialog Box](#)
- [Finding and Replacing Text in a Single File](#)
- [Finding and Replacing Text in Multiple Files](#)
- [Searching for Selected Text](#)
- [Using Regular Expressions \(grep\)](#)

Guided Tour of the Find Dialog Box

The [Find](#) window, shown in [Figure 6.1 on page 171](#) (Windows) and [Figure 6.4 on page 177](#) (Mac OS and Solaris), is a versatile feature of the CodeWarrior IDE. To show the Find dialog box, choose the [Find](#) command in the [Search Menu](#). Use this dialog box to perform find and replace operations for text in a single file or for text in multiple files in your project. You can use text strings, text substrings, and pattern matching to carry out find and replace operations.

There are two different sections in the Find dialog box.

- [Find and Replace Section](#)

- [Multi-File Search Section](#)

Find and Replace Section

This section presents a short tour of the find and replace user interface items in the [Find](#) window shown in [Figure 6.1 on page 171](#). The items in the window are:

- [Find text box](#)
- [Replace text box](#)
- [Recent Strings pop-up menu](#)
- [Find button](#)
- [Replace button](#)
- [Replace & Find button](#)
- [Replace All button](#)
- [Batch checkbox](#)
- [Wrap checkbox](#)
- [Ignore Case checkbox](#)
- [Entire Word checkbox](#)
- [Regexp checkbox](#)
- [Multi-File Search Disclosure triangle](#)
- [Multi-File Search button](#)

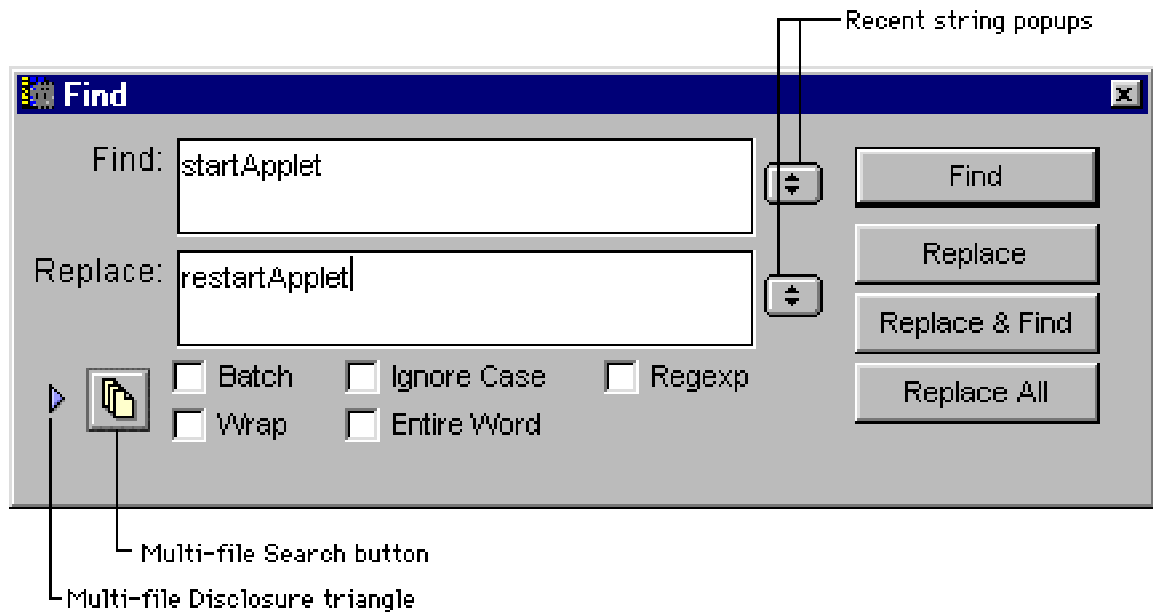
Find text box

The Find text box is one of the editable text fields in the Find dialog box, shown in [Figure 6.1](#). You enter text in this field that you want to search for.

You can use the [Cut](#), [Copy](#), [Paste](#), and [Clear](#) commands with the Find text box. These commands are documented in the section called [“Edit Menu” on page 623](#).

Also, the discussion [“Enter ‘Find’ String” on page 628](#) tells how to enter text into the Find Text Box without using the Find dialog box.

Figure 6.1 The Find Dialog search and replace section



Replace text box

The Replace text box is one of the editable text fields in the Find dialog box, shown in [Figure 6.1](#). The text you enter in this field will be used to replace the text you’re searching for.

You can use the [Cut](#), [Copy](#), [Paste](#), and [Clear](#) commands with the Find Text Box. These commands are documented in the section called [“Edit Menu” on page 623](#).

Also, the discussion [“Enter ‘Replace’ String” on page 628](#) tells how to enter text into the Replace Text Box without using the Find dialog box.

Recent Strings pop-up menu

The Recent Strings pop-up menu is shown in [Figure 6.2](#). It contains strings that were recently used for searches.

There are actually two of these pop-ups. Each pop-up is to the right of both the [Find text box](#) and the [Replace text box](#). Selecting an item in one of these pop-ups enters it in the corresponding text box.

Figure 6.2 Recent Strings pop-up menu



Find button

The **Find** button, shown in [Figure 6.1 on page 171](#), confirms the contents of the Find dialog box and begins the search. The settings in the dialog box are remembered and are displayed when the Find command is invoked again. The Find button is not available until you complete certain required text fields.

To learn more about finding text, see [“Searching for Selected Text” on page 180](#).

Replace button

The **Replace** button is one of the buttons in the Find dialog box, shown in [Figure 6.1 on page 171](#).

When you enter text in the [Find text box](#) and click the **Find** button, the CodeWarrior IDE will search for matching text according to the control settings checked at the bottom of the Find dialog box. If a match is found for the text in the Find text box, the **Replace** button can be clicked to replace the found text with that shown in the [Replace text box](#).

To learn more about searching and replacing text, see [“Replacing Found Text” on page 185](#).

Replace & Find button

The **Replace & Find** button is shown in [Figure 6.1 on page 171](#). This button behaves much like the [Replace button](#), but also initiates another [Find](#) operation after the text substitution is performed.

To learn more about searching and replacing text, see [“Finding and Replacing Text in a Single File” on page 182](#) and [“Finding and Replacing Text in Multiple Files” on page 189](#).

Replace All button

The **Replace All** button is shown in [Figure 6.1 on page 171](#). This button behaves much like the [Replace button](#), but replaces *every* occurrence of the text shown in the [Find text box](#) with the text shown in the [Replace text box](#).

To learn more about searching and replacing text, see [“Replacing Found Text” on page 185](#).

Batch checkbox

The **Batch** checkbox is shown in [Figure 6.1 on page 171](#). Selecting this checkbox causes the results of the **Find** command to appear in a Search Results message window ([Figure 6.7 on page 188](#)).

To learn more about the role of the Batch checkbox in searching, see [“Using Batch Searches” on page 187](#).

Wrap checkbox

The **Wrap** checkbox is shown in [Figure 6.1 on page 171](#). Selecting this checkbox allows a search that reaches one end of a file or group of files to continue from the opposite end.

To learn more about this feature, consult [“Controlling Search Range in a Single File” on page 184](#).

Ignore Case checkbox

The **Ignore Case** checkbox is shown in [Figure 6.1 on page 171](#). This checkbox causes the CodeWarrior IDE to disregard the case (uppercase or lowercase) of the text entered into the [Find text box](#).

To learn more about this feature, consult [“Controlling Search Parameters” on page 184.](#)

Entire Word checkbox

The **Entire Word** checkbox is shown in [Figure 6.1 on page 171](#). This checkbox causes the CodeWarrior IDE to ignore matching text that occurs within words.

To learn more about this feature, consult [“Controlling Search Parameters” on page 184.](#)

Regexp checkbox

The **Regexp** checkbox is shown in [Figure 6.1 on page 171](#). This checkbox causes the CodeWarrior IDE to interpret the text in the [Find text box](#) as a regular expression.

CodeWarrior’s regular expressions are similar to the regular expression for `grep` in UNIX™. To learn more about this feature, refer to [“Using Regular Expressions \(grep\)” on page 198.](#)

Multi-File Search Disclosure triangle

- ▶ The Multi-File Search Disclosure triangle is shown in [Figure 6.1 on page 171](#). Click this triangle to show the [Multi-File Search Section](#) of the [Find](#) window, as shown in [Figure 6.3 on page 175](#) (Windows) or [Figure 6.4 on page 177](#) (Mac OS and Solaris).

To learn more about multi-file searching with the [Find](#) window, see [“Finding and Replacing Text in Multiple Files” on page 189.](#)

Multi-File Search button

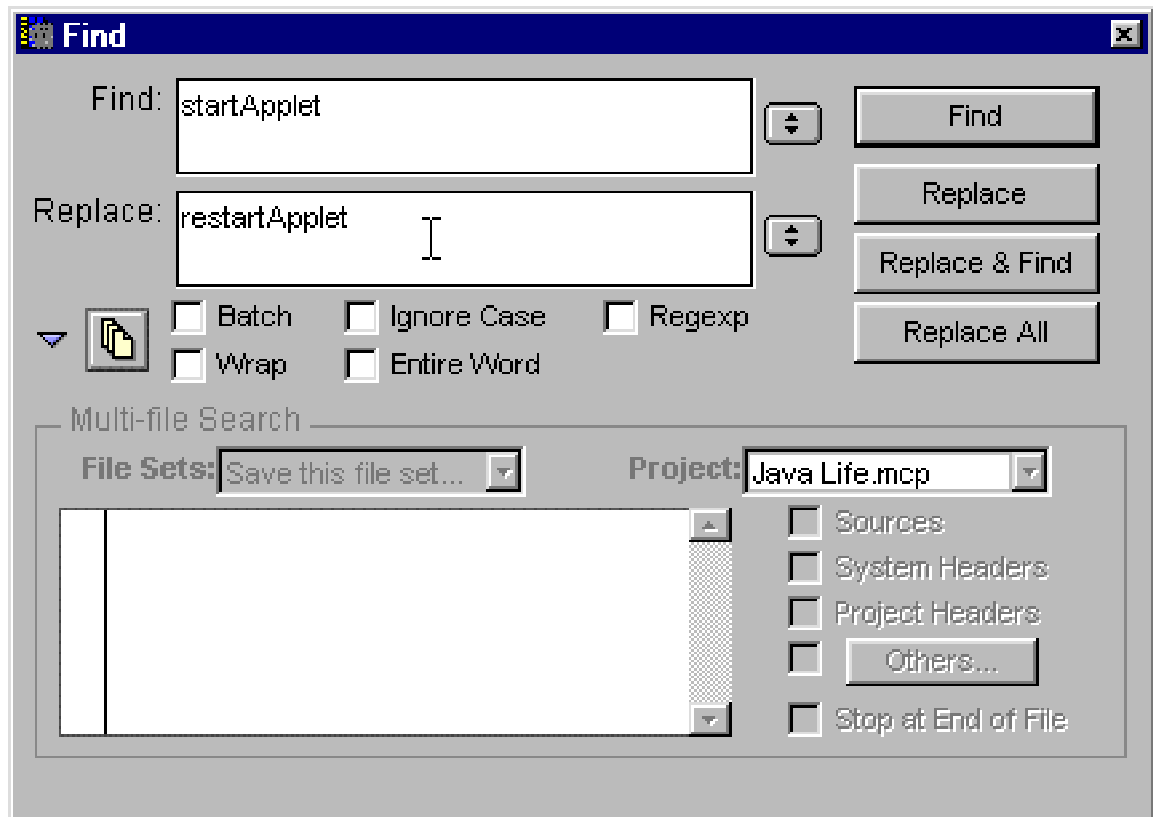


The Multi-File Search button is shown in [Figure 6.3](#). Click this button to enable or disable the options in the [Multi-File Search Section](#) of the [Find](#) window, shown in [Figure 6.4 on page 177](#).

When the Multi-File Search button is not depressed, as shown in the dialog box of [Figure 6.3](#), the items in the [Multi-File Search Section](#) of the [Find](#) window are dimmed.

To learn more about the Multi-File Search Button, see [“Activating Multi-File Search” on page 189](#).

Figure 6.3 Multi-File Search button not selected



Multi-File Search Section

This section describes the user interface items in the [Multi-File Search Section](#) of the [Find](#) window, shown in [Figure 6.4](#). The user interface items are:

- [File list](#)
- [File Sets pop-up menu](#)
- [Project pop-up menu](#)
- [Stop at End of File checkbox](#)
- [Sources checkbox](#)
- [System Headers checkbox](#)

- [Project Headers checkbox](#)
- [Others button](#)

File list

The File list is shown in [Figure 6.4](#). This is a list of the files that will be searched in a Multi-file search. You add files to this list by enabling the **Sources**, **System Headers**, **Project Headers**, and **Others** controls. You can also drag and drop groups or files from the Project window into the list.

For more information about adding files and removing files in file sets, see [“Choosing Files to be Searched” on page 190](#).

File Sets pop-up menu

The File Sets pop-up menu is shown in [Figure 6.5](#). This pop-up menu is used with Multi-file searches. Use this pop-up menu to select, add, and remove saved sets of files to search and replace.

You can build up sets of files, such as collections of header or interface files, that will be available whenever you want to search through the files for text.

For more information about Multi-file sets, see [“Choosing Files to be Searched” on page 190](#).

Project pop-up menu

The Project pop-up menu, shown in [Figure 6.6](#), allows you to choose the project file with which you want to perform your search. Since the CodeWarrior IDE can have multiple projects open at a time, this menu provides a way to perform the same search in different projects.

Figure 6.4 The Find Dialog Box for a multiple file search



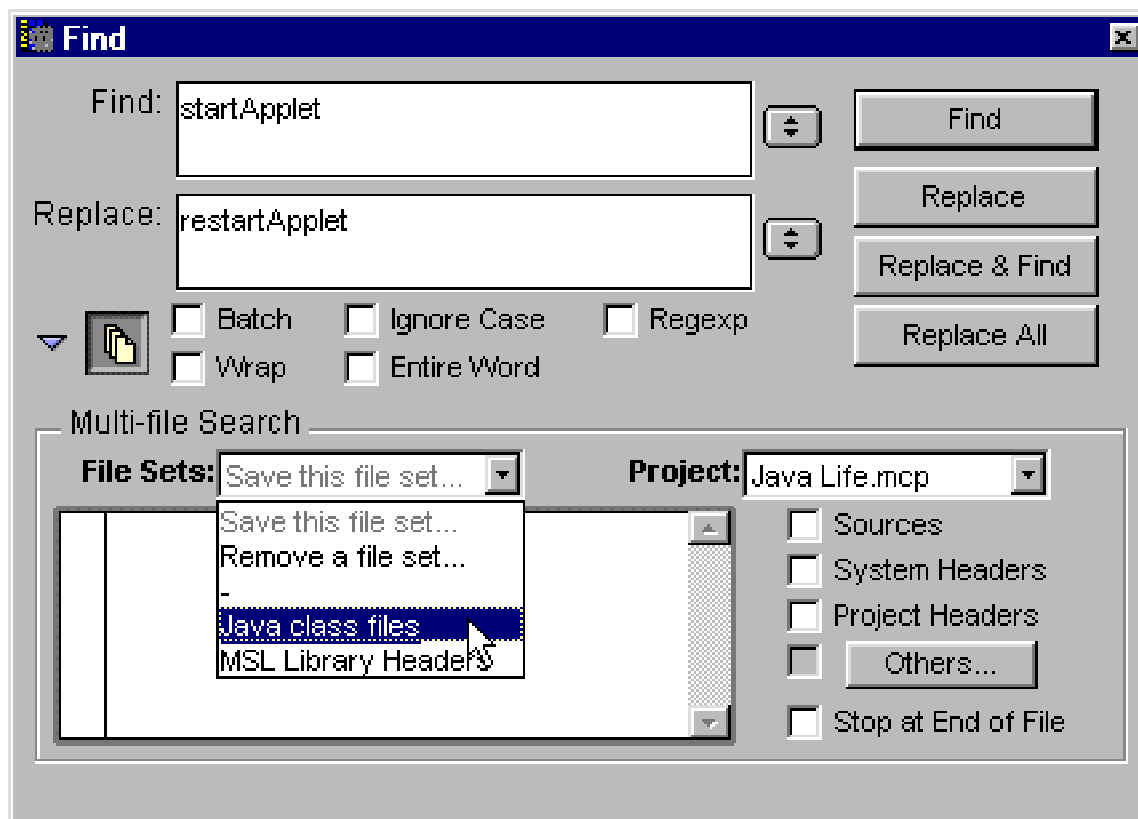
Stop at End of File checkbox

If you turn off the **Stop at End of File** checkbox, all the files in the [File list](#) are searched as one large file. When the CodeWarrior IDE reaches the end of one file, it starts searching the next. When it reaches the end of the last file to search, it beeps.

To search each file individually, enable the **Stop at End of File** checkbox. When the CodeWarrior IDE reaches the end of a file, it stops searching and beeps. You must choose [Find in Next File](#) from the [Search Menu](#) to continue the search.

For more information about using the Stop at End of File checkbox, see [“Controlling Search Range in a Single File” on page 184](#).

Figure 6.5 File Sets pop-up menu



Sources checkbox

The **Sources** checkbox is shown in [Figure 6.4 on page 177](#). This checkbox adds all the source files from the current project to the [File list](#).

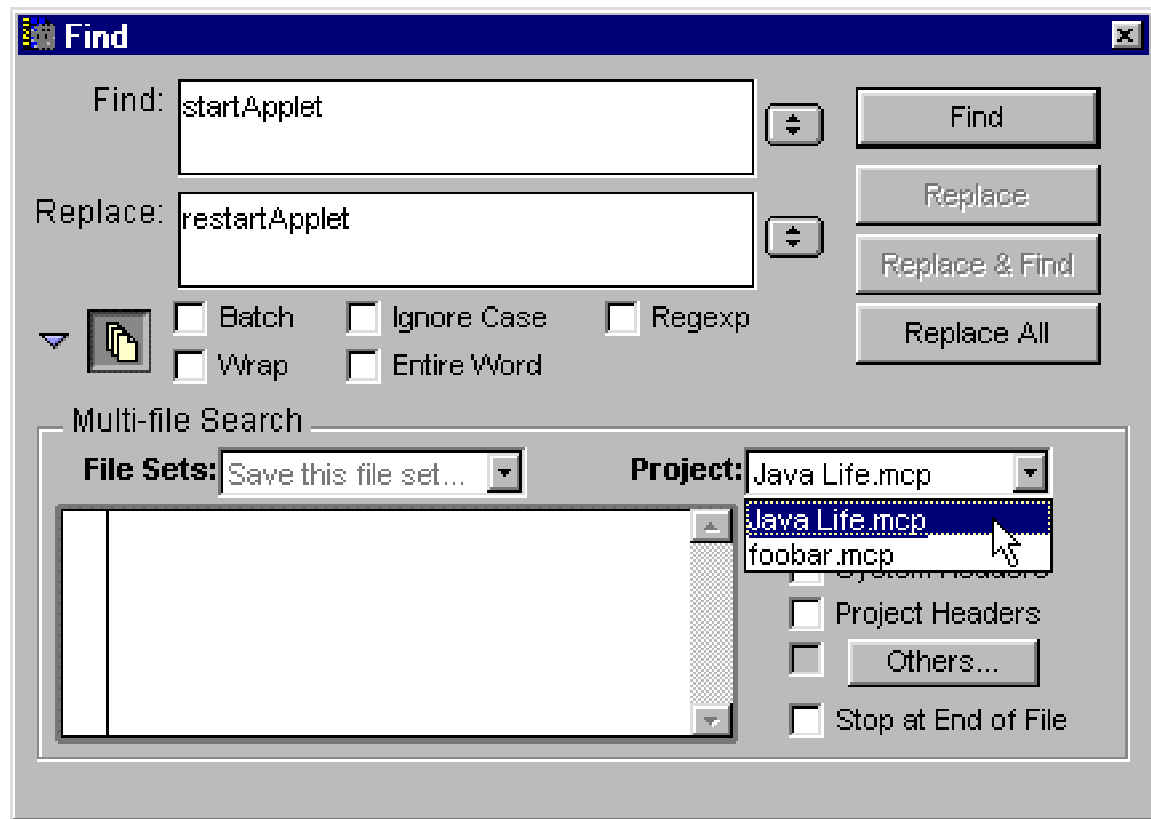
For more information about source files in file sets, and their role in Multi-file searches, see [“Adding project source files” on page 190](#).

System Headers checkbox

The **System Headers** checkbox is shown in [Figure 6.4 on page 177](#). This checkbox adds all system header or interface files from the current project to the [File list](#).

For more information about system headers in file sets, and their role in multi-file searches, see [“Adding system header files” on page 191](#).

Figure 6.6 Project Pop-up Menu



Project Headers checkbox

The **Project Headers** checkbox is shown in [Figure 6.4 on page 177](#). This checkbox adds all the project header or interface files from the current project to the [File list](#).

For more information about project headers in file sets, and their role in Multi-file searches, see [“Adding project header files” on page 191](#).

Others button

The **Others** button is shown in [Figure 6.4 on page 177](#). This button and its checkbox allows you to add one or many additional files to the [File list](#).

For more information about adding file to file sets, see [“Adding and removing arbitrary files” on page 191](#).

Searching for Selected Text

The IDE provides two ways of searching for text without using the Find dialog box. In both of these methods, you select text in a window, and the CodeWarrior IDE finds the text for you without displaying the Find dialog box.

When you search for selected text, the CodeWarrior IDE uses the option settings that you last chose in the [Find](#) window. To change these option settings, you must use the Find dialog box.

You should know how to select text in the editor window before reading this section. If you don't know how to select text, refer to ["Selecting Text" on page 152](#).

Finding text in the active editor window

This method is useful if you want to find additional occurrences of a text string in the same open editor window that you're working with.

First, select an instance of the text you want to find. After selecting your text, choose [Find Selection](#) from the [Search Menu](#).

The CodeWarrior IDE looks for the next occurrence of your text string in the current file only.

To search toward the end of the file for the next occurrence of the text string, click the [Find](#) button or choose [Find Next](#) from the [Search Menu](#). You can also press the keyboard shortcut shown to the right of the Find Next command in the Search menu.

To search toward the beginning of the file for the previous occurrence of the text string, use the [Find Previous](#) command:

Windows This command is available by pressing the key binding Shift-F3.

Mac OS and Solaris This command is available when you hold down the Shift key while using the [Search Menu](#).

The CodeWarrior IDE finds the previous occurrence of the text string and selects it. If the string is not found, the CodeWarrior IDE beeps.

Search for more occurrences of the text string by continuing to use [Find](#), [Find Next](#), or [Find Previous](#) on the [Search Menu](#).

Finding text in another window

This method is useful when your text string is in one file and you want to search for the same text string in another file.

First, select an instance of the text you want to find. After selecting your text, choose [Enter 'Find' String](#) from the [Search Menu](#). The editor enters the text in the [Find text box](#) of the Find dialog box.

Now make the window you want to search active. Then, choose [Find Next](#) from the [Search Menu](#) to search forwards in the active editor window for the next occurrence of your text string.

Use the [Find Previous](#) command to search backwards in the active editor window for the next occurrence of your text string:

Windows This command is available by pressing the key binding Shift-F3.

Mac OS and Solaris This command is available when you hold down the Shift key while using the [Search Menu](#).

The CodeWarrior IDE looks for the [Find text box](#) string in the active editor window, starting from the location of the text insertion point in that window.

If you want to search toward the end of the file for the next occurrence of the [Find text box](#) string, click the [Find](#) button or choose [Find Next](#) from the [Search Menu](#). You can also press the keyboard shortcut shown to the right of the Find and Find Next commands in the Search menu.

To search toward the beginning of the file for the previous occurrence of the Find string, use the [Find Previous](#) command.

Search for more occurrences of the [Find text box](#) string by continuing to use [Find](#), [Find Next](#), or [Find Previous](#) from the [Search Menu](#).

Finding and Replacing Text in a Single File

The [Find](#) window allows you to search for text patterns in the active editor window. When you find the text you are interested in, you can change it or look for another occurrence of it.

This section discusses how to use the Find dialog box to locate specific text you want to replace in the active editor window.

If you don't yet have a window open, see ["Opening an Existing File" on page 110](#).

If you haven't yet created a file, see ["Creating a New File" on page 109](#).

The topics in this section are:

- [Finding Search Text](#)
- [Controlling Search Range in a Single File](#)
- [Controlling Search Parameters](#)
- [Searching with Special Characters](#)
- [Replacing Found Text](#)
- [Using Batch Searches](#)

Finding Search Text

To enter text in the [Find text box](#), bring up the [Find](#) window using the **Find** command in the Search menu. Type a text string into the [Find text box](#) on the dialog box, or choose a string from the [Recent Strings pop-up menu](#), as shown in [Figure 6.2 on page 172](#).

NOTE To learn how to enter search text containing a Return or Tab character, refer to ["Searching with Special Characters" on page 185](#).

Before searching, you can set other search options that control the range of your search.

The search range defines whether you want to search the entire file or just from the text insertion point in one direction. To set up the

range of your search, see [“Controlling Search Range in a Single File” on page 184](#).

The search parameters define whether you want to search for text regardless of text case, and whether to search partial words for the text. To set up the parameters of your search, see [“Controlling Search Parameters” on page 184](#).

Before proceeding, make sure that multi-file searching is turned off since you are only interested in searching the active editor window. To learn about how to determine whether multi-file searching is turned off, refer to [“Activating Multi-File Search” on page 189](#).

Click the **Find** button in the [Find](#) window to search forward from the text insertion point in the file, or choose **Find** or **Find Next** from the [Search Menu](#). CodeWarrior now searches for the [Find text box](#) string in the active editor window.

Windows Press the key binding Shift-F3 to perform a [Find Previous](#) command, which searches backwards from the text insertion point in the file.

Mac OS and Solaris Hold down the Shift key and choose [Find Previous](#) from the [Search Menu](#) if you want to search backwards from the text insertion point in the file.

To continue searching toward the end of the file for the next occurrence of the [Find text box](#) string, click the **Find** button or choose **Find Next** from the Search menu.

To continue searching toward the beginning of the file for the previous occurrence of the [Find text box](#) string, use the [Find Previous](#) command.

The editor finds and selects the [Find text box](#) string. If the string is not found, the editor beeps.

Search for more occurrences of the [Find text box](#) string by continuing to use [Find](#), [Find Next](#), or [Find Previous](#).

From this point, you can replace some or all of the text you find with a new text string.

To replace text, see [“Replacing Found Text” on page 185](#).

Controlling Search Range in a Single File

The **Wrap** checkbox option in the Find dialog box controls the flow of the search when you reach the beginning or end of a file.

For example, suppose that you enable the [Wrap checkbox](#) option. When the text insertion point is somewhere in the middle of a source file in the active editor window, and you choose **Find Next** on the [Search Menu](#), the CodeWarrior IDE searches from the insertion point to the end of the file. When the search reaches the end of the file, the search continues from the beginning of the file to the insertion point. In other words, the search “wraps” around the ends of the file.

The **Find Previous** command operates in a similar fashion. When the search reaches the beginning of the file and the Wrap checkbox option is enabled, the search continues from the end of the file.

If you have the [Wrap checkbox](#) option unchecked, and you choose **Find Next** on the [Search Menu](#), the search stops when it reaches the end of the file.

If you’re searching multiple files with the [Wrap checkbox](#) option checked, the CodeWarrior IDE searches from the first file in the file list after it reaches the last file.

Controlling Search Parameters

The **Ignore Case** checkbox and the **Entire Word** checkbox are two easily-accessible options in the CodeWarrior IDE that you can use for matching text.

Ignore Case checkbox

The **Ignore Case** checkbox is shown in [Figure 6.1 on page 171](#). When selected, the CodeWarrior IDE treats uppercase and lowercase text in the [Find text box](#) as identical. When deselected, uppercase and lowercase text are distinct.

For example, when you select the **Ignore Case** checkbox and enter Foobar in the [Find text box](#), the IDE finds occurrences of foobar, FOOBAR, and other possible combinations of uppercase and lowercase characters.

Entire Word checkbox

The **Entire Word** checkbox is shown in [Figure 6.1 on page 171](#). This checkbox causes the CodeWarrior IDE to find only complete words (delimited by punctuation or white-space characters) matching the text in the [Find text box](#). If deselected, the IDE finds occurrences of the search text embedded within larger words. For example, suppose you enter Word in the [Find text box](#). If the **Entire Word** checkbox is selected, the IDE finds only the occurrences of Word in the file. If the **Entire Word** checkbox is deselected, the IDE matches Word even when it is embedded inside other text, such as Words, WordCount, and BigWordCount.

Searching with Special Characters

To enter a Tab or Return character in the Find or Replace fields, use one of the following methods:

- Copy and paste your selected text with the Tab or Return characters into the Find or Replace field, or
- If drag and drop is supported, drag the text from the editor window directly into a field, or
- Enable the **Regex** option and enter `\t` for Tab or `\r` for Return into the field.

Mac OS and Solaris To directly enter a Tab character, press Option-Tab. To enter a Return character, press Option-Return.

WARNING!

Using **Regex** will alter the manner in which CodeWarrior locates a string match. See [“Using Regular Expressions \(grep\)” on page 198](#) for more information on using **Regex**.

Replacing Found Text

When you find an occurrence of text you are interested in, you can either replace one occurrence at a time, or you can replace all occurrences in the entire file.

Selective Replace

To selectively replace text, first enter some text to find, then choose the **Find** operation on the [Search Menu](#), or click the [Find button](#) in the Find dialog box. You can read more about how to find text by referring to [“Finding Search Text” on page 182](#).

Next, enter the replacement text string in the [Replace text box](#) field of the Find dialog box.

Type the string in the [Replace text box](#) field or choose a string from the [Recent Strings pop-up menu](#) of the [Replace text box](#) by clicking the arrow icon just to the right. The [Recent Strings pop-up menu](#) ([Figure 6.2 on page 172](#)) contains the last five strings you have used.

Now choose whether to replace the string you found. For convenience, there are three buttons in the Find dialog box for doing this, the **Replace** button, the **Replace & Find** button, and the **Replace All** button. Each button performs a different operation.

To replace the string and see the results, click the [Replace button](#) in the Find dialog box or choose **Replace** from the [Search Menu](#). The editor replaces the text that was found with the [Replace text box](#) string.

To continue searching forward, choose **Find Next** from the [Search Menu](#), or click the [Find button](#) in the Find dialog box.

To continue searching backward, use the **Find Previous** command, or press the Shift key and click the [Find button](#) in the Find dialog box.

To replace the string and find the *next* occurrence, choose **Replace & Find Next** from the [Search Menu](#), or click the [Replace & Find button](#) in the Find dialog box. The editor replaces the selected text with the [Replace text box](#) string and finds the next occurrence of the [Find text box](#) string. If the editor can't find another occurrence, it beeps.

To replace the [Find text box](#) string and find the *previous* occurrence, use the **Replace & Find Previous** command:

Windows Press the key binding Ctrl-Shift-L, or press the Shift key as you click the **Replace & Find button** in the Find dialog box.

Mac OS hold down the Shift key as you choose [Replace & Find Previous](#) from the [Search Menu](#), or press the Shift key as you click the [Replace & Find button](#) in the Find dialog box.

The editor replaces the selected text with the [Find text box](#) string and searches for a previous occurrence of the [Find text box](#) string. If the CodeWarrior IDE can't find another occurrence, it beeps.

Replace All

To replace text, first enter some text to find in the [Find text box](#), then choose the [Find](#) operation on the [Search Menu](#), or click the [Find button](#) in the Find dialog box. You can read more about how to find text by referring to [“Finding Search Text” on page 182](#).

Next, enter the replacement text string in the [Replace text box](#) field of the Find dialog box.

To replace all the occurrences of the [Find text box](#) string, click the [Replace All button](#) in the Find dialog box, or choose [Replace All](#) from the [Search Menu](#).

WARNING!

Be careful when you use the [Replace All](#) command, since **Undo** is not available for this operation.

TIP

If you are going to perform a **Replace All** operation on a single source file, make sure to save the source file before executing the replace operation. In the event that you should change your mind, and before you save any changes, use **Revert** to replace the modified file in memory with the saved version on disk. This technique will not work across multiple files.

Using Batch Searches

The CodeWarrior IDE gives you a way to collect all matching descriptions of your text search in one window for easy reference.

If the [Batch checkbox](#) option is checked in the Find dialog box, and the **Find** button is clicked, the CodeWarrior IDE searches for all

Searching and Replacing Text

Finding and Replacing Text in a Single File

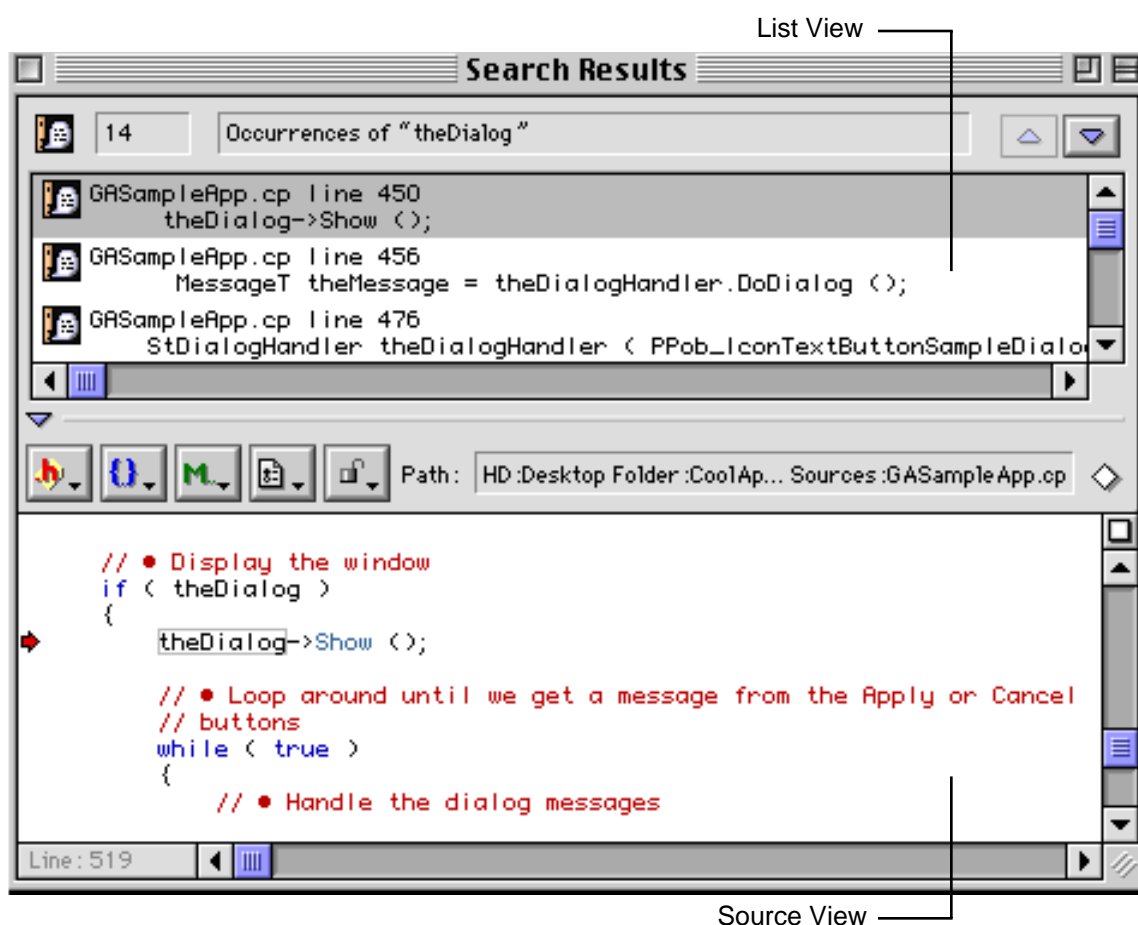
occurrences of the [Find text box](#) string and lists them in the Search Results message window, as shown in [Figure 6.7](#).

The Search Results window shown in [Figure 6.7](#) has a List View and a Source View.

To go to a particular occurrence of the [Find text box](#) string, so that it is shown in the Source View pane of the window, double-click on its entry in the List View.

To learn more about the features of this window, refer to the discussion of the Message Window in [“Guided Tour of the Message Window” on page 379](#).

Figure 6.7 Batch search results



Finding and Replacing Text in Multiple Files

The CodeWarrior IDE allows you to search multiple files for the occurrence of text strings.

In this section you will learn how to do text searches through multiple files.

Another way to quickly access information and search in multiple files is with the Browser's **Go Back** and **Go Forward** commands on the Search menu. To learn about how to use these commands, refer to [“Go Back and Go Forward” on page 246](#).

The topics in this section are:

- [Activating Multi-File Search](#)
- [Choosing Files to be Searched](#)
- [Saving a File Set](#)
- [Removing a File Set](#)
- [Controlling Search Range in Multiple Files](#)

Activating Multi-File Search

To configure the CodeWarrior IDE to search through multiple files, you need to activate multi-file searching in the Find dialog box.



When the [Multi-File Search button](#) is on, the button appears to be depressed.

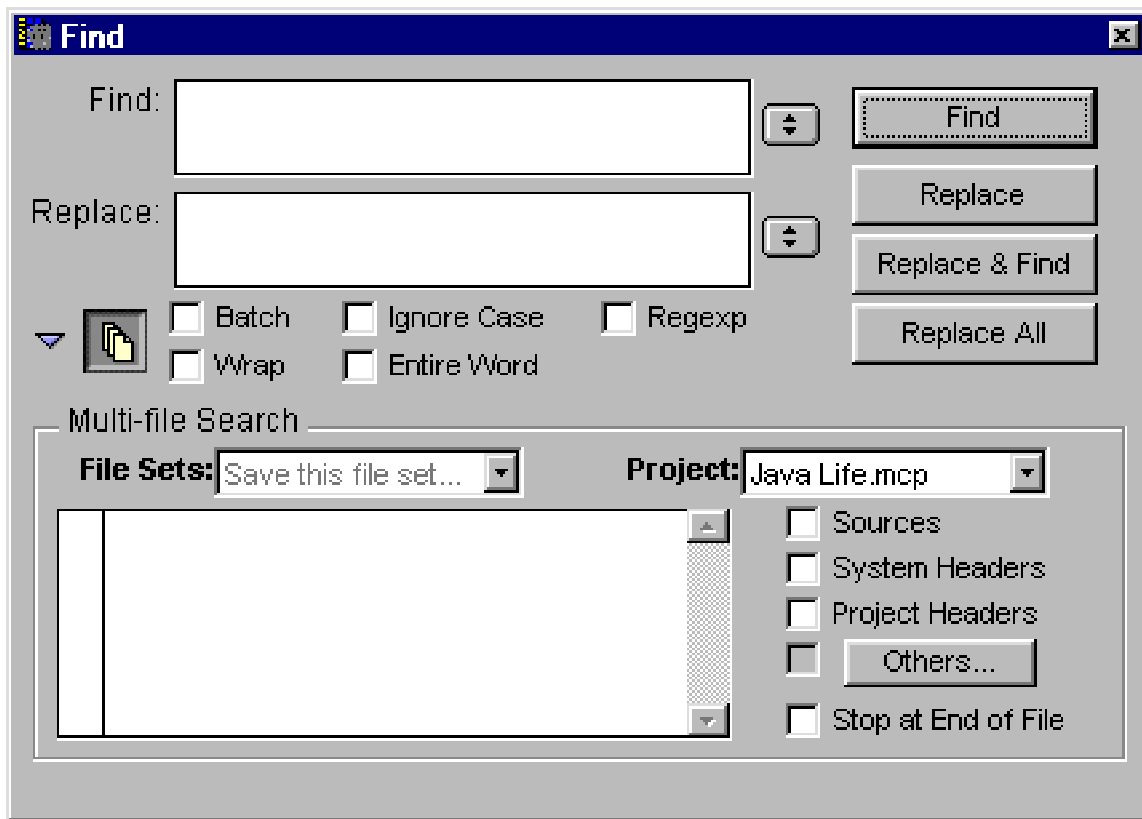


When the [Multi-File Search button](#) is off, the button looks three-dimensional.

Click the [Multi-File Search Disclosure triangle](#) to the left of the [Multi-File Search button](#), shown in [Figure 6.1 on page 171](#), so that the triangle points down.

The CodeWarrior IDE displays the [Multi-File Search Section](#), an extension of the Find dialog box, as shown in [Figure 6.8 on page 190](#). When the [Multi-File Search button](#) is on (as shown above), the [Multi-File Search Section](#) is enabled.

Figure 6.8 The Find dialog box with multi-file search options



To learn about configuring the [Multi-File Search Section](#) of the Find dialog box, refer to [“Choosing Files to be Searched” on page 190](#), [“Saving a File Set” on page 194](#), [“Removing a File Set” on page 195](#), and [“Controlling Search Range in Multiple Files” on page 196](#).

Choosing Files to be Searched

There are several ways to choose files for a search.

Adding project source files

To add all the source files from the current project, turn on the **Sources** checkbox. When you turn off the [Sources checkbox](#), the CodeWarrior IDE removes all associated files from the file list.

To include only some of the files, turn on the [Sources checkbox](#) and delete the files you don't want by selecting them and pressing Backspace/Delete after clicking on the file name.

If turning on this option doesn't add any files, update your project's internal list of header and interface files with the Make command. To learn how to do this, refer to ["Making a Project" on page 366](#).

Adding project header files

To add all the project header or interface files from the current project, turn on the **Project Headers** checkbox. When you turn off the [Project Headers checkbox](#), the CodeWarrior IDE removes all associated files from the file list.

To include only some of the files, turn on the [Project Headers checkbox](#) and delete the files you don't want by selecting them and pressing Backspace/Delete.

If turning on this option doesn't add any files, update your project's internal list of header or interface files with the Make command. To learn how to do this, refer to ["Making a Project" on page 366](#).

Adding system header files

To add all the system header or interface files from the current project, turn on the **System Headers** checkbox. When you turn off the [System Headers checkbox](#), the CodeWarrior IDE removes all associated files from the file list.

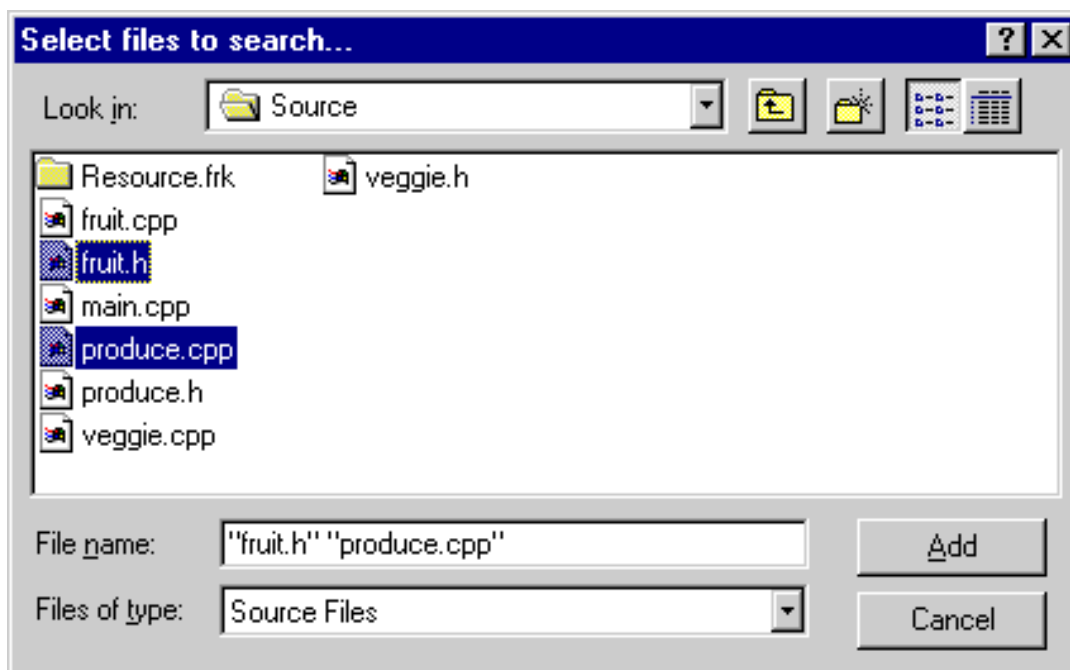
To include only some of the files, turn on the [System Headers checkbox](#) and delete the files you don't want by selecting them and pressing Delete.

If turning on this option doesn't add any files, update your project's internal list of header or interface files with the Make command. To learn how to do this, refer to ["Making a Project" on page 366](#).

Adding and removing arbitrary files

For multi-file searches, you can manually add and remove files to your file set. Adding files to a file set enables you to search in files not included in your current project.

Figure 6.9 The Select Files To Search dialog box (Windows)



First, click the [Others button](#) in the [Multi-File Search Section](#) of the Find dialog box. The Select Files To Search dialog box will be displayed as shown in [Figure 6.9](#) (Windows), [Figure 6.11](#) (Mac OS), and [Figure 6.12](#) (Solaris).

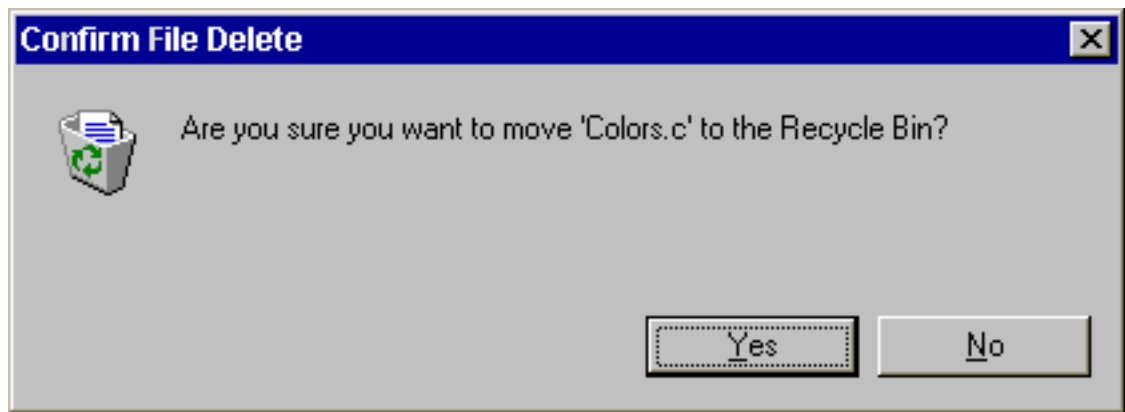
The Select Files To Search dialog box displays a list of the files in the current directory that may be added to the file set. To add a file to the file set, select it and click the **Add** button. If you change your mind, click **Cancel** and the file set will not be changed.

Windows You can select multiple files in this list by pressing the Control key and clicking a file simultaneously. When you're finished choosing files, click **Add**. All files in the Select Files To Search list will appear in the file set. To remove files from the file set, select them in the Find dialog box ([Figure 6.8 on page 190](#)) and press Backspace or Delete.

WARNING! (Windows) If you select a file's icon in the Select Files To Search dialog box ([Figure 6.9](#)) and press the Delete key, the Confirm File Delete dialog box ([Figure 6.10](#)) will appear. This dialog box warns

you that you are attempting to delete the selected file from your hard drive. Click **No** to return to the Select Files To Search dialog box. Instead of selecting the file's icon, select the file's name (including the quote marks) in the File Name text box and press Backspace or Delete. This is the correct way to remove a selected file from the Select Files To Search dialog box.

Figure 6.10 Confirm File Delete dialog box (Windows)



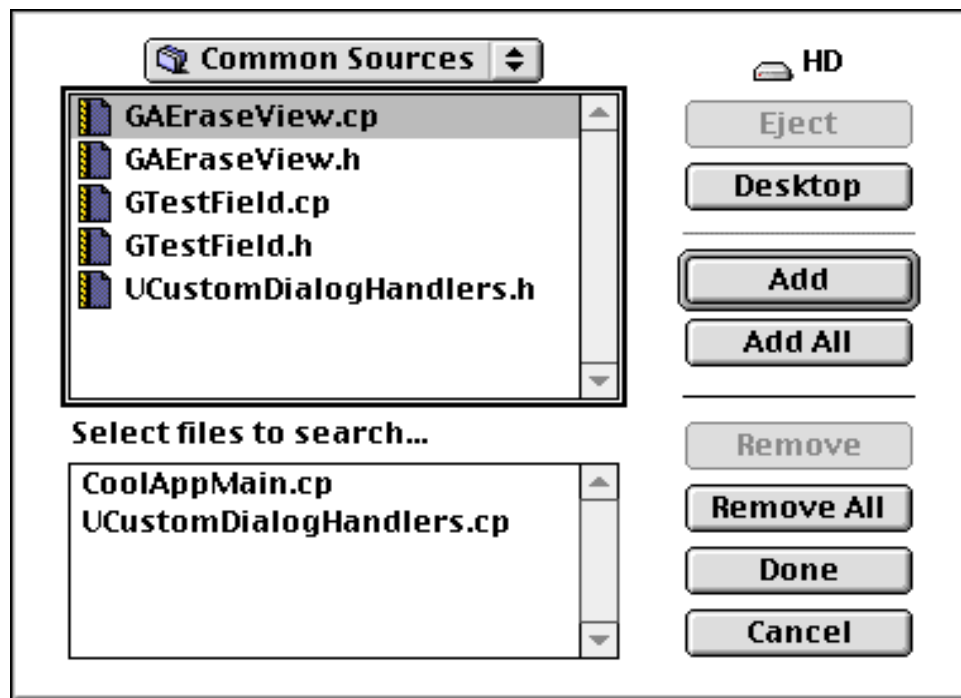
Mac OS and Solaris To add all files in the list, click the **Add All** button. The files are removed from the top file list and reappear in the Select Files To Search list at the bottom of the dialog box. To remove files from the Select Files To Search list, select them and click the **Remove** button. To remove all the files from the Select Files To Search list, click the **Remove All** button. When you're finished choosing files, click **Done**. All files in the Select Files To Search list will appear in the file set.

To add more files later, just click the [Others button](#) in the Find dialog box and repeat the process.

Choosing a file set

To select a previously-saved file set to include in your search, click on the [File Sets pop-up menu](#) and choose a file set from the menu, as shown in [Figure 6.5 on page 178](#). The files then appear in the File Sets list.

Figure 6.11 The Select Files To Search dialog box (Mac OS)



Saving a File Set

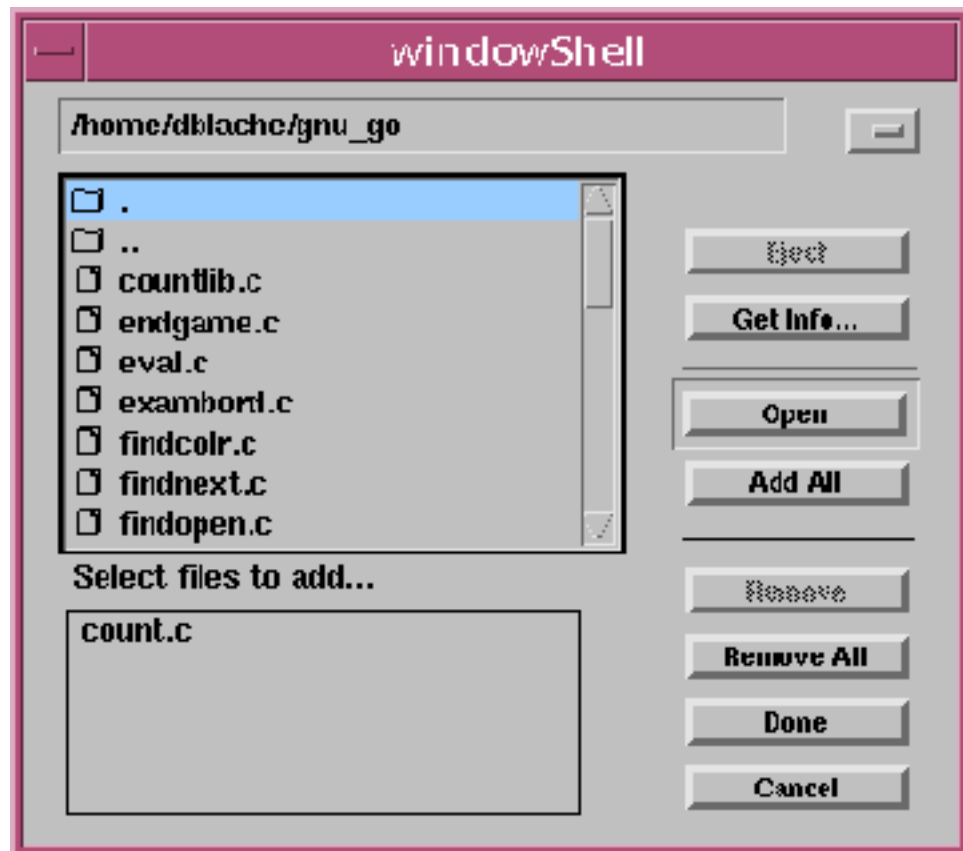
To save a file set for use in future multi-file searches, choose **Save this File Set** from the File Sets pop-up menu. The IDE displays the Save File Set dialog box shown in [Figure 6.13 on page 196](#).

Name the file set by entering a name in the Save File Set As text field.

You can choose the scope of projects which can use this file set. There are two scopes to choose from: **Specific** and **Global**. Click the radio button next to the scope you want to use with this file set.

If you plan to use this file set only with the current project, click **Specific to this project**. The CodeWarrior IDE stores the file set in the project.

Figure 6.12 The Select Files To Search dialog box (Solaris)



If you think you'll use this file set with other projects, click **Global, for all projects**. The CodeWarrior IDE stores the file set in its preferences file, so all projects (even existing projects) can use it.

After making your selection and naming the file set, click the **OK** button. If you change your mind and don't want to save the file set, click **Cancel**.

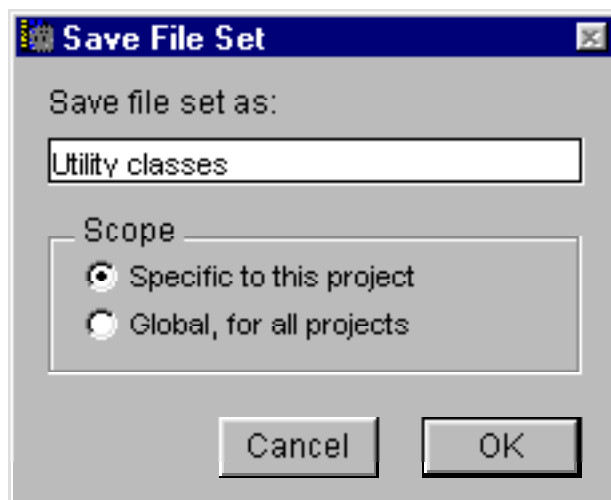
Removing a File Set

To remove a previously-saved file set, choose **Remove a file set** from the [File Sets pop-up menu](#) in the Find dialog box. The CodeWarrior IDE displays the dialog box shown in [Figure 6.14](#).

Select the file set you want to remove, then click the **Remove** button. The CodeWarrior IDE removes the file set so that future searches do not use the deleted file set. When you are finished removing file

sets, click the **Done** button to return to the Find dialog box. If you change your mind about removing the file set, click **Cancel** instead.

Figure 6.13 The Save File Set dialog box



Controlling Search Range in Multiple Files

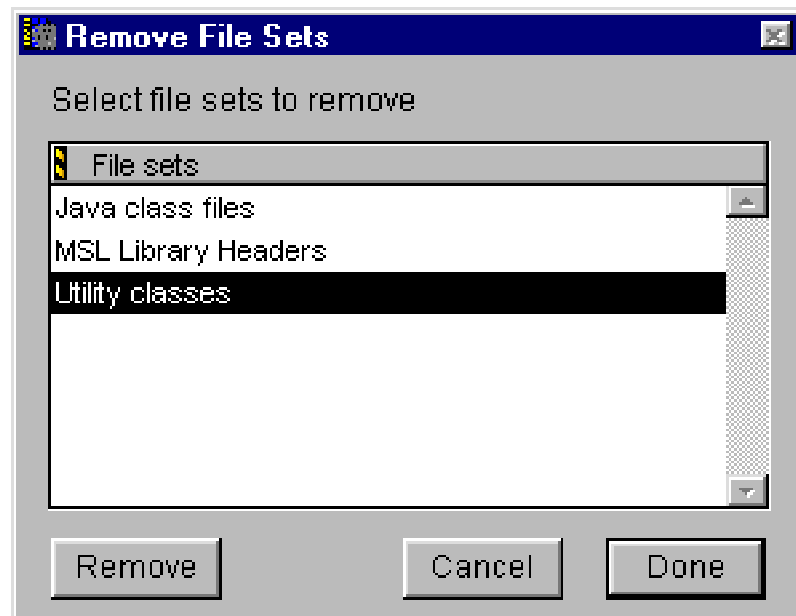
The CodeWarrior editor lets you search any number of files for a string. The files can be in the current project or any text file on disk. If you frequently search a particular set of files, just save that set and restore it later.

You can choose to stop searching at the end of each file or to search all files without stopping.

To treat all the files in the file set as one large file, turn off the [Stop at End of File checkbox](#). When the editor reaches the end of one file, it starts searching the next file until the selected text is found. The editor beeps when it reaches the end of the last file to search. After text is found, you may resume searching for the next occurrence using the [Find](#), [Find Next](#), or [Find Previous](#) menu commands.

To search each file individually, enable the [Stop at End of File checkbox](#). The editor beeps when it reaches the end of a file. The arrow to the left of the file set indicates the file the editor is currently searching.

Figure 6.14 Remove File Sets dialog box



You must choose [Find in Next File](#) from the [Search Menu](#) or use the [Find in Previous File](#) command to continue the search. To start the search from a particular file, just select the file and click in the column to its left.

Windows The Find in Previous File command is available by pressing the key binding Ctrl-Shift-T.

Mac OS The Find in Previous File command is available by pressing the Shift key while using the [Search Menu](#).

After choosing your option, proceed just as you would if you were searching only one file.

TIP You can search in the previously-searched file in the file list by using the **Find in Previous File** command. This effectively allows you to go backwards in your search into previously-searched files.

To learn more about text searching, see [“Searching for Selected Text” on page 180](#), or [“Finding and Replacing Text in Multiple Files” on page 189](#).

Using Regular Expressions (*grep*)

A regular expression is a text substring that is used as a mask for comparing text in a file. Regular expressions can be formed from single characters or from more complicated strings. When the regular expression is compared with the text in your file, the CodeWarrior IDE analyzes whether the text matches the regular expression you have entered.

This section discusses special operators in regular expressions that the CodeWarrior IDE recognizes and how they can be used to find and replace text. CodeWarrior's regular expressions are similar to UNIX's *grep* commands.

NOTE Make sure the **Regexp** checkbox is selected in the Find dialog box.

This section consists of the following topics:

- [Special Operators](#)
- [Using Regular Expressions](#)

Special Operators

[Table 6.1](#) shows characters that have special meanings based on their placement in the regular expression. For more information, see ["Using Regular Expressions" on page 200](#).

Table 6.1 Descriptions of metacharacters

Metacharacters	Description
"."	The match-any-character operator matches any single printing or non-printing character except newline and null.
"*"	The match-zero-or-more operator repeats the smallest preceding regular expression as many times as necessary (including zero) to match the pattern.

Metacharacters	Description
" + "	The match-one-or-more operator repeats the preceding regular expression at least once and then as many times as necessary to match the pattern.
" ? "	The match-zero-or-one operator repeats the preceding regular expression once or not at all.
" \n "	The back-reference operator is used in the replace string to refer to a specified group in the find string. Each group must be enclosed within parentheses. The digit n must range between 1 and 9. The number identifies a specific group, starting from the left side of the regular expression.
" "	The alternation operator matches one of a choice of regular expressions. If you place the alternation operator between any two regular expressions, the result matches the largest union of strings that it can match.
" ^ "	The match-beginning-of-line operator matches the string from the beginning of the string or after a newline character. When it appears within brackets the " ^ " represents a "not" action.
" \$ "	The match-end-of-line operator matches the string either at the end of the string or before a newline character in the string.
[...]	List operators enable you to define a set of items to use as a match. The list items must be enclosed within square brackets. Note that you cannot define an empty list.
(...)	Group operators define sub-expressions that can be used elsewhere in the regular expression as a single unit.
" - "	The range operator defines the characters that fall between the start and ending characters within the list.

Using Regular Expressions

You can create powerful regular expressions to search for text and perform replace operations on found text. This section will discuss the following topics:

- [Matching simple expressions](#)
- [Matching any character](#)
- [Repeating expressions](#)
- [Grouping expressions](#)
- [Choosing one character from many](#)
- [Matching the beginning or end of a line](#)
- [Using the Find string in the Replace string](#)
- [Remembering sub-expressions](#)
- [References](#)

To give you a better idea of how regular expressions can improve search and replace functions, look over the following example code. Each concept discussed will refer to the example code shown in [Listing 6.1](#).

Listing 6.1 Example Code

```
#include <iostream>

#define var1 10;
#define var2 20;

using namespace std;

int result = 0;

int main(void)
{
    cout << "This example provides information about" << endl;
    cout << "the use of regular expressions in the" << endl;
    cout << "CodeWarrior IDE." << endl << endl;
    cout << "Refer to this code when reading the" << endl;
    cout << "sections in the manual dealing with" << endl;
    cout << "the use of regular expressions." << endl << endl;
}
```



```
cout << "The value of var1 is " << var1;
cout << " out of 50, and" << endl;
cout << "the value of var2 is " << var2;
cout << " out of 50." << endl;
cout << "$" << var1;
cout << " + $" << var2;
cout << " = $" << ;
result += var1;
result += var2;
cout << result;
cout << endl << endl;
return 0;
}
```

Matching simple expressions

Most characters match themselves. For example, “a” matches all occurrences of the letter “a” in the code. The only exceptions are called special characters: the asterisk (*), plus sign (+), backslash (\), period (.), caret (^), square brackets ([and]), dollar sign (\$), and ampersand (&). To match a special character, precede it with a backslash, like this: *

For example, refer to the example code shown in [Listing 6.1 on page 200](#). If you want to find every occurrence of the letter “m” in the code, you would type m into the Find text box. The CodeWarrior IDE will match this regular expression with the “m” in `iostream`. Additional searches will match with the “m” in `namespace`, `main(void)`, `example`, `information`, and `manual`.

If you want to find every occurrence of a dollar sign in the code, you would type \\$ in the Find text box. The backslash tells the IDE to interpret the dollar sign as a normal character rather than a special character. The IDE will match the regular expression with the first dollar sign in the following line from [Listing 6.1](#):

```
cout << "$" << var1;
```

Additional searches will match with the second and third dollar signs in the next two lines.

Matching any character

A period (.) matches any character except a newline character. Use the period when you want to be more flexible in your search.

For example, refer to the example code shown in [Listing 6.1 on page 200](#). If you want to find four-character expressions in the code that begin with `var`, type `var .` in the Find text box. The period following the `var` tells the IDE to look through the code for `var` immediately followed by a single character. The IDE will match this regular expression with `var1` and `var2` in the code.

Repeating expressions

The asterisk and plus sign are two special operators that allow you to “repeat” expressions in your search string:

- A regular expression followed by an asterisk (*) matches *zero* or more occurrences of that regular expression. If there is any choice, the editor chooses the longest, left-most matching string in a line.
- A regular expression followed by a plus sign (+) matches *one* or more occurrences of that regular expression. If there is any choice, the editor chooses the longest, left-most matching string in a line.
- A regular expression followed by a question mark (?) matches *zero or one* occurrences of that regular expression. If there is any choice, the editor chooses the left-most matching string in a line.

For example, refer to the example code shown in [Listing 6.1 on page 200](#). Suppose you type `s*ion` in the Find text box. The `s*` tells the IDE to match zero or more occurrences of the letter `s` just before the occurrence of `ion`. In [Listing 6.1](#), this regular expression will match with the `ion` in both `information` and `sections` from the following lines:

```
cout << "This example provides information about" << endl;  
cout << "sections in the manual dealing with" << endl;
```

The same regular expression will also match with the `ssion` in expressions from the following lines:

```
cout << "the use of regular expressions in the" << endl;  
cout << "the use of regular expressions." << endl << endl;
```

If you want to find matches that have at least one letter *s* preceding *ion*, you would type *s+ion* in the Find text box. The plus sign tells the IDE to match at least one occurrence of the letter *s* just before the occurrence of *ion*. This regular expression will match with the *ssion* in expressions from [Listing 6.1](#). Notice that *s+ion* does not match with the *ion* in *information* or *sections*, since there is not at least one letter *s* preceding *ion* in either case.

If you want to find expressions in the code that contain the number zero, followed by one period or no periods at all, you would type *0\.?* in the Find text box. The backslash tells the IDE to treat the period as a normal character, and the *?* special operator acts on the normal period character. This expression will match with each occurrence of the number zero in [Listing 6.1](#). This regular expression will also match with the *0.* from the following line:

```
cout << " out of 50." << endl;
```

As shown in these examples, the asterisk and plus sign usually refer to a single character. However, it is possible to refer to more than one character at a time by grouping expressions. See the following section, [Grouping expressions](#), for more information.

Grouping expressions

If an expression is enclosed in parentheses (*(* and *)*), the editor treats it as a single unit and applies any asterisk (***) or plus (*+*) to the whole expression.

For example, refer to the example code shown in [Listing 6.1 on page 200](#). If you want to find expressions in the code that match *is*, you could simply type *is* in the Find text box. However, you could also use *(i)s* as the regular expression. Notice that *(i)s* tells the IDE to look for the letter *s* preceded by both a space and the letter *i*. Whereas *is* matches the *is* within *This*, *this*, and *is*, *(i)s* will match only with *is* in the following two lines from [Listing 6.1](#):

Searching and Replacing Text

Using Regular Expressions (*grep*)

```
cout << "The value of var1 is " << var1;  
cout << "The value of var2 is " << var2;
```

Choosing one character from many

A string of characters enclosed in square brackets (`[]`) matches any one character in that string. To match any character that is *not* in the string enclosed within the square brackets, precede the enclosed expression with a caret (`^`) like this: `[^abc]`

For example, refer to the example code shown in [Listing 6.1 on page 200](#). If you want to find expressions in the code that contain the letters x, y, or z, you would type `[xyz]` in the Find text box. The IDE will match this regular expression with the letter x in `example` and `expressions`. If instead you want to find expressions in the code that do *not* contain the letters x, y, or z, type `[^xyz]` in the Find text box. The IDE will match this regular expression with every character in the example code except the letter x in `example` and `expressions`.

Placing a minus sign (`-`) within square brackets indicates a range of consecutive ASCII characters. For example, `[0-9]` is the same as `[0123456789]`. If the minus sign is the first or last character within the enclosed string, that minus sign loses its special meaning and is treated as an ordinary character. For example, `[-bc]` represents the minus sign and the letters b and c.

If a right square bracket is immediately after a left square bracket, it does not terminate the string but is considered to be one of the characters to match. For example, `[]0-9]` tells the IDE to search for the right square bracket as well as any digit in the code. If any special character, such as backslash (`\`), asterisk (`*`), or plus sign (`+`), is immediately after the left square bracket, that special character is treated as an ordinary character. For example, `[.]` tells the IDE to search for periods in the code.

You can use square brackets in a similar manner as parentheses. The IDE will treat the information in the square brackets as a single unit. For example, `[bsl]ag` tells the IDE to search for occurrences of `bag`, `sag`, or `lag` in the code. Typing `[aeiou][0-9]` in the Find text box tells the IDE to search for a vowel followed by a number, such as `a1`.

Matching the beginning or end of a line

You can specify that a regular expression match only the beginning or end of the line.

- If a caret (^) is at the beginning of the entire regular expression, it matches the beginning of a line.
- If a dollar sign (\$) is at the end of the entire regular expression, it matches the end of a line.
- If an entire regular expression is enclosed by a caret and dollar sign (^like this\$), it matches an entire line.

For example, refer to the example code shown in [Listing 6.1 on page 200](#). If you want to find expressions in the code that match `cout` and occur only at the beginning of a line, you would type `^([\t]*cout)` in the Find text box. The `[\t]*` in the regular expression allows zero or more spaces and tabs to precede `cout`, as is the case with the code shown in [Listing 6.1](#).

Using the Find string in the Replace string

You can include the contents of the Find string in the Replace string by using an ampersand (&) in the Replace string. For example, refer to the example code shown in [Listing 6.1 on page 200](#). Suppose the Find string is `var[0-9]` and the Replace string is `my_&`. The editor will match the Find string with `var1` and `var2` in the code. Clicking the Replace button in the Find dialog box will replace `var1` with `my_var1` or `var2` with `my_var2`.

To use an ampersand in the Replace without any special meaning, use `\&`. An ampersand has no special meaning in the Find string.

Remembering sub-expressions

You can remember sub-expressions of a regular expression in a Find string and recall those sub-expressions in the Replace string. You can create up to nine remembered sub-expressions for each Find string. Each sub-expression must be enclosed within parentheses. To recall these sub-expressions when typing the replace string, use `\n`, where *n* is a digit that specifies which sub-expression to recall. Determine *n* by counting sub-expressions from the left side of the Find string.

Searching and Replacing Text

Using Regular Expressions (*grep*)

For example, refer to the example code shown in [Listing 6.1 on page 200](#). If you want to change `#define` declarations into `const` declarations, you could perform a search for `#define` and manually change the line of code to a `const` declaration. However, you can take advantage of remembered sub-expressions to perform the same process automatically. Begin by typing the following regular expression in the Find text box:

```
\#define[ \t]+(.+)[ \t]+([0-9]+);
```

This regular expression tells the IDE to search for the following string, in the exact order given: `#define`, one or more spaces or tabs, one or more characters, one or more spaces or tabs, one or more digits, and a semicolon. Starting from the left side of the regular expression, the first sub-expression is `(. +)` and the second sub-expression is `([0-9]+)`. These two sub-expressions are recalled in the following Replace string:

```
const int \1 = \2;
```

The `\1` refers to the first sub-expression and the `\2` refers to the second sub-expression from the Find string. These two sub-expressions recall the variable name and its value from the original `#define` declaration. Notice that the replace string changes the `#define` declaration into a `const` declaration by using references to the two sub-expressions. Thus, the editor will find `#define var1 10;` and change it into `const var1 = 10;` in the code shown in [Listing 6.1](#). It will change the next `#define` statement in the same manner.

References

For a comprehensive book on using regular expressions, get *Mastering Regular Expressions*, by Jeffrey E.F. Friedl, published by O'Reilly & Associates, Inc., from your local bookstore.

Browsing Source Code



This chapter describes CodeWarrior's class browser, a tool you use to examine your project source code from various perspectives.

The CodeWarrior browser creates a database of all the symbols in your code, and provides you with a user interface to access the data quickly and easily, regardless of language.

Historically, programmers have used browsers primarily with object-oriented code, but the CodeWarrior IDE browser works with both procedural and object-oriented code. It works with most compilers, including C, C++, Pascal, and Java.

To help you understand the browser, we are going to look at it from three perspectives: high-level architecture, user interface, and functionality.

The topics in this chapter include:

- [Activating the Browser](#)
- [Understanding the Browser Strategy](#)
- [Guided Tour of the Browser](#)
- [Browser Wizards](#)
- [Using the Browser](#)

Activating the Browser

To learn more about how to activate the browser, refer to:

- [“Configuring IDE Options” on page 253](#) to learn how to configure IDE preferences related to the browser
- [“Activate Browser” on page 338](#) for details about the target-specific preference settings that activate the browser.

When the browser is activated, the compiler generates the browser database information.

For more information on browser settings and options, see [“Setting Browser Options” on page 245](#).

To learn how to use contextual menus in the browser, see [“Browser Contextual Menu” on page 213](#).

Understanding the Browser Strategy

When you activate the browser, the CodeWarrior IDE compilers generate a database of information about your code. This database includes data not only about your code, but also about the relationships between various parts of your code, such as inheritance hierarchies.

The browser is a user interface that allows you to sort and sift through this information in ways that suit your needs.

Like any good database access program, the browser does not dictate how you should look at your information. It gives you a variety of tools to suit your working style.

There are three principal ways of looking at the information available to you in the browser:

- [Contents View](#)—a comprehensive view of all data
- [Browser View](#)—a class-based view
- [Hierarchy View](#)—an inheritance-based view

These sections take a brief look at each option. [“Guided Tour of the Browser” on page 212](#) discusses the user interface in detail.

The browser also implements instant access to information. By right-clicking (Windows) or clicking and holding the mouse (Mac OS, Solaris, and Linux) on any symbol for which there is information in the database, you get instant access to related source code. [“Browser Contextual Menu” on page 213](#) discusses this feature.

In addition, the browser lets you decide the scope of the view. You can look at data in all your classes, or you may wish to focus on one class.

Within the browser and hierarchy views, you can look at multiple classes or single classes. [Table 7.1](#) summarizes the various major choices you have when using the browser.

Table 7.1 Browser viewing options

Viewing Style	Wide Focus	Narrow Focus
comprehensive	contents	not applicable
inheritance-based	multi-class hierarchy	single-class hierarchy

The browser-related menu commands in the [Window Menu](#)—[Browser Contents](#), [Class Hierarchy Window](#), and [New Class Browser](#)—display wide-focus views. Once you have the wide view, you can focus on a particular class.

No matter what viewing style or focus you happen to be in at any given moment, the browser has simple and intuitive mechanisms for switching to another kind of view.

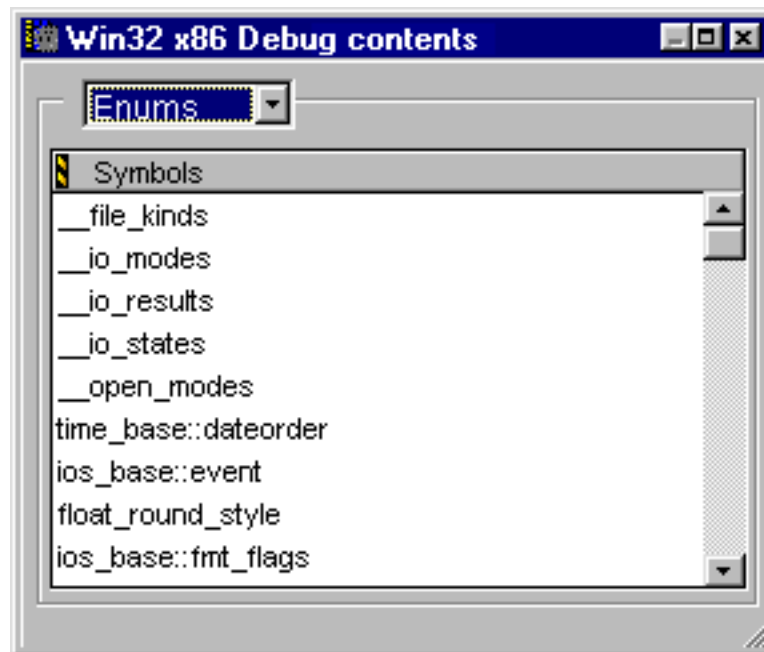
Contents View

The Contents view lets you see all of your data sorted by category into alphabetical lists. [Figure 7.1](#) shows a Contents view.

You select the particular category you want to examine. You can focus on classes, constants, enumerations, routines, global variables, macros, routine templates, and type definitions.

See [“Contents Window” on page 215](#) for details on the Contents window interface.

Figure 7.1 A Contents view



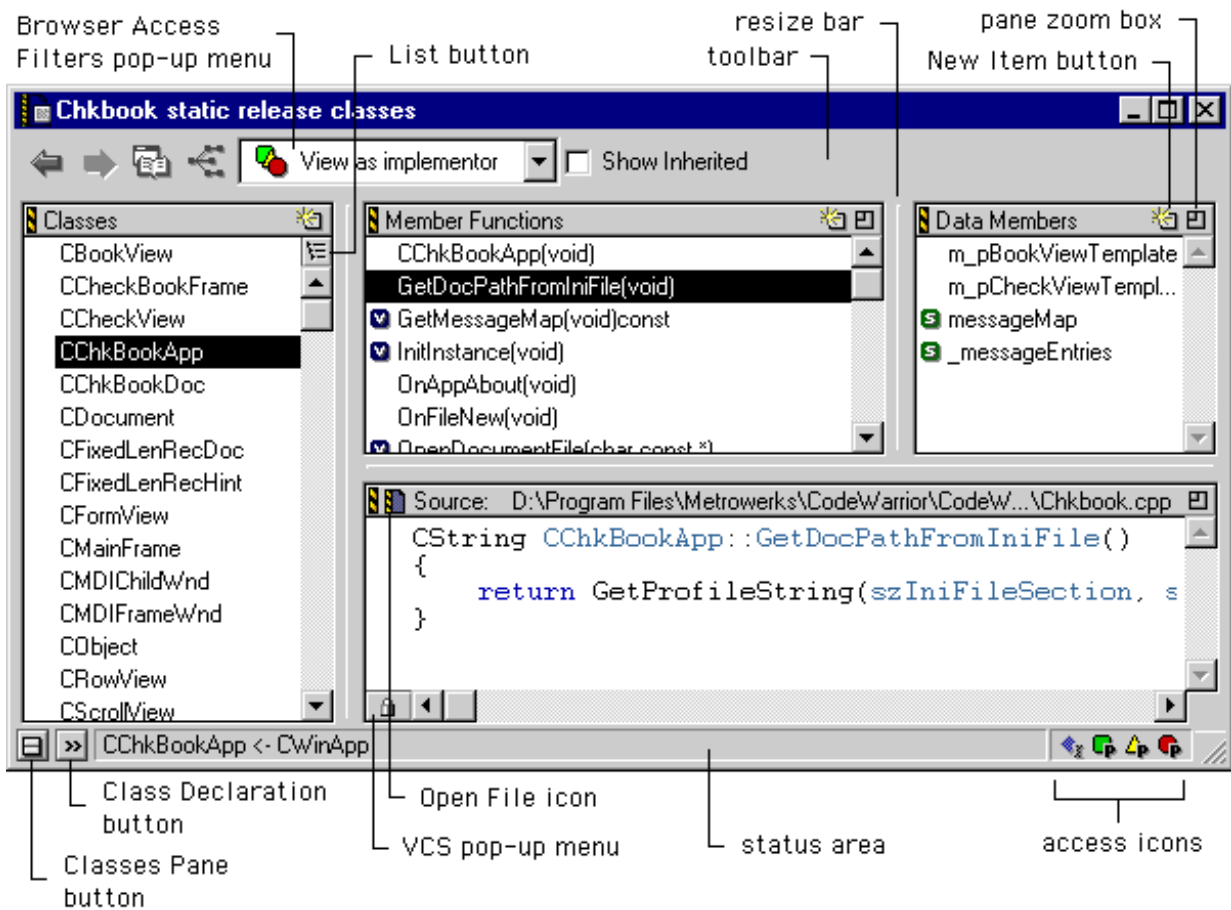
Browser View

The Browser view is like a traditional class browser. You use this view to look at your data from a class-oriented perspective. [Figure 7.2](#) shows what a Browser view looks like.

In the Browser view, you have a list of classes. For the selected class in that list, you see all of its member routines and its data members. When you select an item, the source code related to that item is displayed in the Source pane.

See [“Browser Window” on page 216](#) for details on the user interface elements in the Browser view.

Figure 7.2 A Browser view

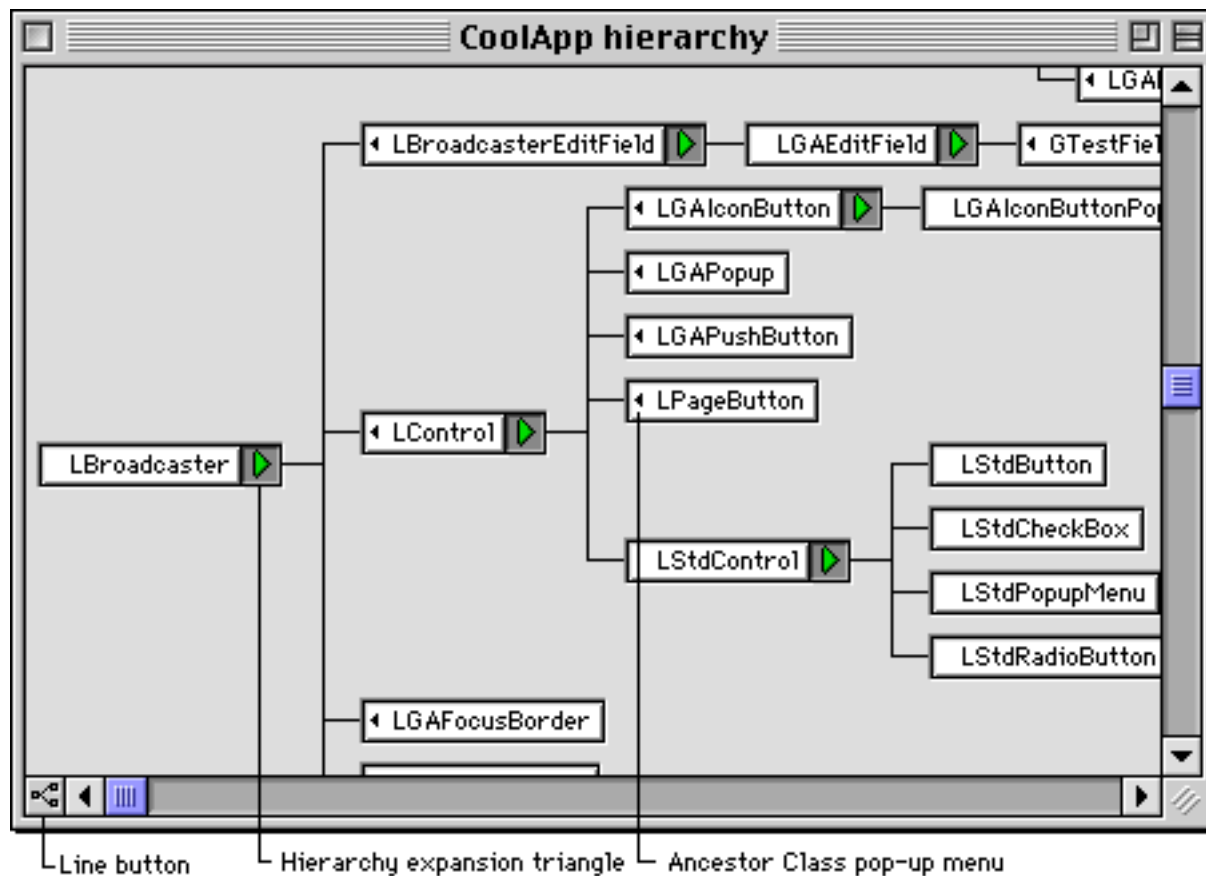


NOTE The Browser view has a toolbar. To learn how to use and customize the toolbar, see [“Customizing the IDE” on page 296](#). The Browser view also has a tab control when browsing rapid application development (RAD) projects. For more information about the tab control, refer to [“RAD Browsing” on page 579](#).

Hierarchy View

The Hierarchy view is a graphical view of your class hierarchy. You use this view to understand or follow class relationships. [Figure 7.3](#) illustrates a multi-class hierarchy view for some classes.

Figure 7.3 A Hierarchy view



The Hierarchy view gives you a real feel for the way your classes are connected with each other. You can expand and collapse a hierarchy at will.

See [“Multi-Class Hierarchy Window” on page 224](#) and [“Single-Class Hierarchy Window” on page 226](#) for details on the Hierarchy view interface.

Guided Tour of the Browser

At first glance, the browser interface is quite complicated. There are multiple windows filled with controls and information. However, there are really only three kinds of views: Contents, Browser, and Hierarchy.

In addition, the browser contextual menu is a fundamental and vital feature of the browser interface. The contextual menu is available for any symbol for which the browser database has data. The menu commands describe various actions that you can perform on the selected item.

For object-oriented code (member functions only), you have the opportunity to see the declaration or definition of any routine with that name. You can also open up a symbol browser that lists every implementation of the routine. Depending on the nature of the symbol (class name, routine name, enumeration, and so forth), the contextual menu lists different destinations appropriate for the item. This gives you instant access to the source code related to any symbol.

This section examines each window used by the browser and the browser controls. The sections are:

- [Browser Contextual Menu](#)
- [Contents Window](#)
- [Browser Window](#)
- [Multi-Class Hierarchy Window](#)
- [Single-Class Hierarchy Window](#)
- [Symbol Window](#)
- [Browser Menu](#)

Browser Contextual Menu

When the browser is active, right-click (Windows) or click and hold (Mac OS, Solaris, and Linux) on any symbol for which there is data in the browser database. When you do, the IDE displays a contextual menu with a variety of items.

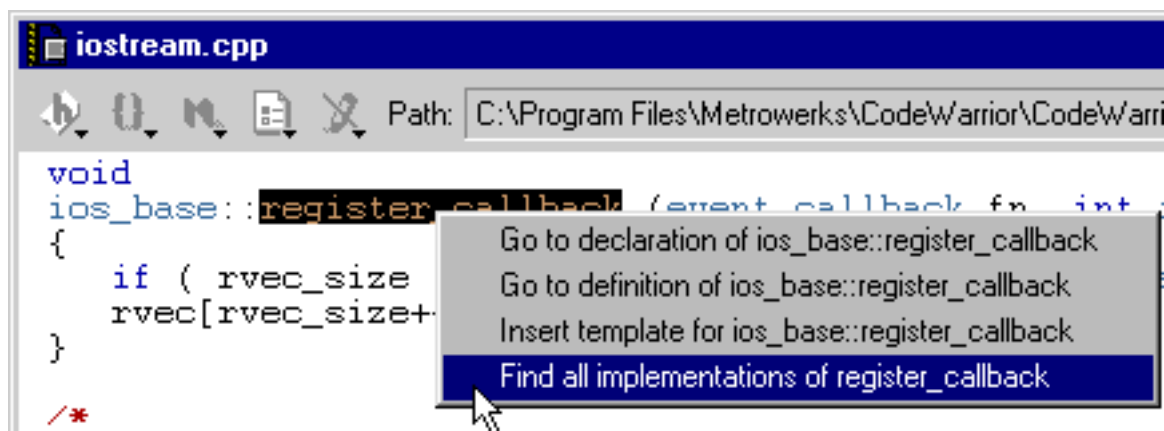
The commands displayed in the menu depend on the nature of the symbol you are investigating. Some commands let you open browser windows, while other commands direct you to a location in code.

You can also use the contextual menu to perform common debugging tasks. For example, you can choose commands to set and clear breakpoints in the active editor window.

In effect, every symbol in your code—routine name, class name, data member name, constant, enumeration, template, macro, type definition, and so forth—becomes a hypertext link to other locations in your source code.

For example, if you select class name and use the browser contextual menu, you can open the class declaration, a [Browser Window](#), or a [Single-Class Hierarchy Window](#). If you use the contextual menu for a routine name, you get different choices, as shown in [Figure 7.4](#).

Figure 7.4 A browser contextual menu for a routine



You can even insert a routine template at the location if you wish. See [“Browser Display” on page 272](#) for more information.

Other menus for other kinds of symbols have similar commands.

TIP To find and enter a browser item for a piece of text you have selected or just entered, invoke the browser contextual menu for that text. The menu offers a list of matching items. Choose one of these items to enter it for you. See [“Completing Symbols” on page 247](#) for other ways to type browser items.

TIP The contextual menu features work not only in browser windows, but also in the CodeWarrior source code editor! This is a great

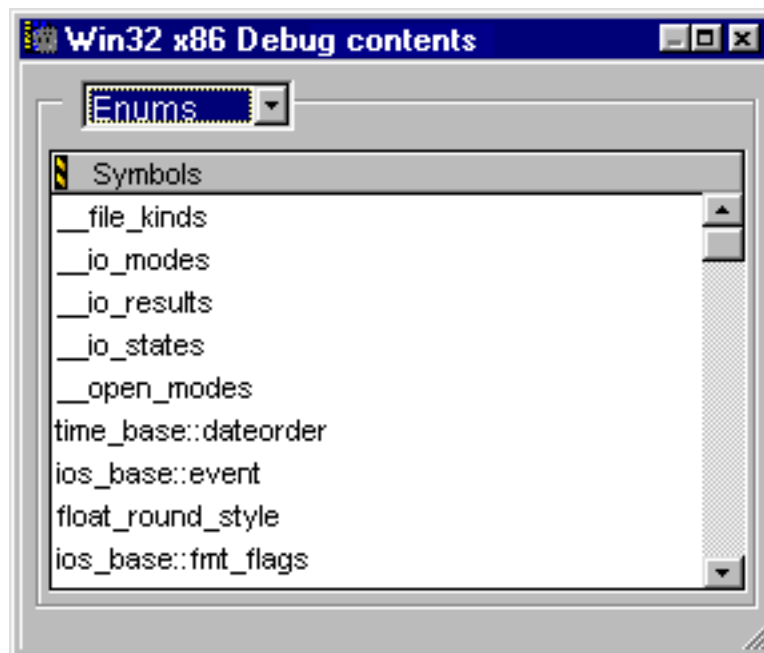
reason for always having the browser enabled, even if you do not use the browser windows.

Of particular note in the contextual menu for routine names is the **Find all implementations of** command. When you choose this command for a routine that has multiple definitions, a [Symbol Window](#) opens in the browser.

Contents Window

The Contents window displays browser data sorted by category into alphabetical lists. Choose [Browser Contents](#) from the [Window Menu](#) to display the Contents window, shown in [Figure 7.5](#).

Figure 7.5 A Contents window



The items in this window are:

- [Category pop-up menu](#)
- [Symbols pane](#)

Category pop-up menu

The Category pop-up menu select the type of information displayed in the [Symbols pane](#). The currently selected item is highlighted (Windows) or checked (Mac OS, Solaris, and Linux).

Symbols pane

The Symbols pane displays every item in the browser database that is a member of the currently selected category. The items are listed alphabetically.

NOTE Routines are listed alphabetically by *routine* name, but the class name is displayed first. As a result, it may seem that the routines are not listed alphabetically.

Browser Window

The Browser window ([Figure 7.2 on page 211](#)) gives you a class-based view of the information in the browser database. The window has several panes displaying lists of information.

You can open Browser window using several techniques:

- Choose [New Class Browser](#) from the [Window Menu](#).
- Use the [Browser Contextual Menu](#) in the [Contents View](#) when classes or functions are displayed.
- Double-click a class name in a [Multi-Class Hierarchy Window](#)
- Double-click a class name in a [Single-Class Hierarchy Window](#)

TIP The Browser window has a toolbar. To learn how to use and customize the toolbar, see [“Customizing the IDE” on page 296](#). When browsing rapid application development (RAD) projects, the Browser window has a tab control. To learn how to use the tab control, refer to [“RAD Browsing” on page 579](#).

The [Classes pane](#), [Member Functions pane](#), and [Data Members pane](#) are lists of their respective data. Clicking within a particular pane

makes it the active pane. You can also use the Tab key to rotate through all the panes (except for the Source pane).

TIP Using the Tab key can be mildly hazardous to your source code. If the Source pane is active and you press the Tab key, you enter a tab into your source code. Once you are in the Source pane, you cannot press the Tab key to get out of it.

Windows Use the mouse to select a different pane.

Mac OS, Solaris, and Linux Use the mouse or press Option-Tab to move to a different pane. A focus outline surrounds the active pane.

You can click an item in any list to select it, or navigate through the items in the active list by typing or using the arrow keys. You can also select items by typing their names. The selection changes to most closely match the characters you type.

Mac OS, Solaris, and Linux Use the Control-Tab key combination to cycle through list items alphabetically.

The items in this window include:






- [Browser toolbar](#)
- [Browser Access Filters pop-up menu](#)
- [Pane zoom box](#)
- [Resize bar](#)
- [Classes pane](#)
- [List button](#)
- [Classes Pane button](#)
- [Class Declaration button](#)
- [Member Functions pane](#)
- [Data Members pane](#)
- [Identifier icon](#)
- [Source pane](#)
- [Open File icon](#)
- [VCS pop-up menu](#)

- [Status area](#)

Browser toolbar

The toolbar provides easy single-click access to many CodeWarrior commands. The buttons that are displayed in the Browser window's toolbar are listed in [Table 7.2](#).

Table 7.2 Browser toolbar buttons

Button	Description
	Go Back
	Go Forward
	Show Browser Contents window
	Show Multi-Class Hierarchy window
	Show Single-Class Hierarchy window

Browser Access Filters pop-up menu







The Browser Access Filters pop-up menu controls the display of member functions and data members in the Browser view. The pop-up menu's commands "filter" the display according to public, private, and protected access types. A mark is displayed next to each enabled filter. The available choices in the pop-up menu are described in [Table 7.3](#).

The first three choices represent shortcuts to multiple filters. For example, instead of using the Browser Access Filters pop-up menu three times in succession to enable public, private, and protected filters, choose **View as implementor** to enable all three filters simultaneously.

After you enable a particular filter, the access icons change in the bottom-right corner of the Browser window. Darkened icons, as



shown in [Table 7.3](#), indicate enabled filters. Grayed-out icons represent disabled filters.

Table 7.3 Browser Access Filters pop-up menu commands

Command	Access Icon	Show items with...
View as implementor		public, private, and protected access
View as subclass		public and protected access
View as user		public access
Show public		public access
Show protected		protected access
Show private		private access

Pane zoom box

The pane zoom box enlarges and shrinks panes in the browser.

-  Click this box to enlarge a pane to fill the Browser window.
-  Click this box to shrink a pane to its original size.

Resize bar

A resize bar is located between each pair of panes, as illustrated in [Figure 7.2 on page 211](#). To resize two panes, drag the resize bar located between them.

TIP When a source file is displayed in the [Source pane](#), pressing Control/Command-` opens the associated interface file. If the file in the [Source pane](#) is an interface file, you open the related source code file, and vice versa.

Classes pane

The Classes pane lists all the classes in the browser database.

You can view the list alphabetically or by class hierarchy. Toggle the view by clicking the [List button](#) at the top right of the pane, as shown in [Figure 7.2 on page 211](#).

In the Hierarchy view, hierarchy expansion triangles are displayed next to class names that have subclasses, as shown in [Figure 7.3 on page 212](#). Click the expansion triangle to show or hide subclasses.

TIP In the Classes pane, Alt/Option click a hierarchical control to open all subclasses at all levels. This is a “deep” disclosure. Ctrl/Command click to open a single level of subclass in a class and all of its siblings at the same level. This is a “wide” disclosure. Ctrl/Command-Alt/Option-click to open both wide and deep.

When you select a class in the Classes pane, the [Multi-Class Hierarchy Window](#) selection changes too. The Hierarchy view scrolls to the newly selected class if necessary.

NOTE The Classes pane does not display information about classes or structures that do not have any member functions, base classes, or subclasses. Thus, structures and classes that just have fields and data members are not displayed.

List button

The List button at the top right of the [Classes pane](#) controls the listing of classes. You can toggle between an alphabetical list or a hierarchical list.

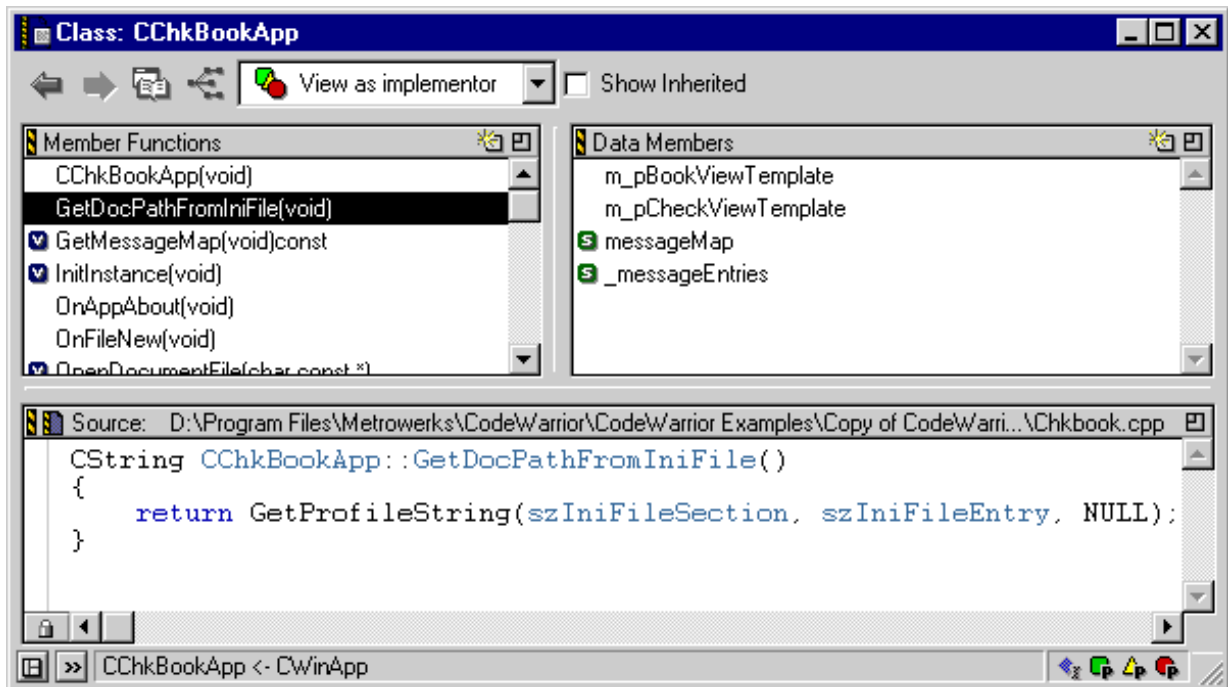


Click this button to switch to an alphabetical list.



Click this button to switch to a hierarchical list.

Figure 7.6 The Browser window with the Classes pane hidden



Classes Pane button

The Classes Pane button at the bottom left of the Browser window controls the display of the [Classes pane](#).



Click this button to hide the Classes pane ([Figure 7.6](#)).



Click this button to display the Classes pane ([Figure 7.2](#)).

Class Declaration button


The Class Declaration button at the bottom left of the Browser window lets you access the declaration of the current class.



Click this button to show the declaration of the current class in the [Source pane](#). The [Status area](#) of the Browser window shows the name of the current class.


Member Functions pane

The Member Functions pane lists all member functions defined in the selected class. Constructors and destructors are at the top of the list. After that, all other entries are listed in alphabetical order.

You can also display inherited member functions by enabling the **Show Inherited** checkbox in the toolbar. The Inherited access icon in the Browser window () darkens to indicate that inherited member functions are currently displayed.

TIP If you select a member function in the [Member Functions pane](#) in the [Browser Window](#), you can use the Enter/Return key to open an editor window and view the definition of the selected function.

Data Members pane

The Data Members pane lists all data members defined in the selected class. You can also display inherited data members by enabling the **Show Inherited** checkbox in the toolbar. The Inherited access icon in the Browser window () darkens to indicate that inherited data members are currently displayed.




The entries in the Data Members pane are listed in alphabetical order. If inherited members are displayed, data members are listed by superclass, but alphabetically within each class.

TIP If you select a data member in the [Data Members pane](#) in the [Browser Window](#), you can use the Enter/Return key to open an editor window and view the declaration of the selected data member.

Identifier icon

A routine or data member can have an identifier icon beside its name. The following table describes the icons.

Table 7.4 Browser identifier icons

Icon	Meaning	The member is...
	static	a static member
	virtual	a virtual function that you can override, or an override of an inherited function
	pure virtual or abstract	a member function that you must override in a subclass if you want to create instances of that subclass

Source pane

The Source pane displays the source code for the selected item. If the item is a class, the pane shows the class declaration. If the item is a routine, the pane shows the routine definition. If the selected item is a data member, the pane shows the data member declaration from the interface file.

NOTE If you Alt/Option click an item in the [Member Functions pane](#) or the [Data Members pane](#), that item is entered into the [Source pane](#) text at the current insertion point. This is an efficient way to enter routine calls or variable names into the [Source pane](#).

The text in the Source pane is fully editable. The path to the file that contains the code on display is shown at the top of the Source pane.

Open File icon



The Open File icon is located at the top of the Source pane. Click this icon to open the file that contains the code displayed in the Source pane. The source code is displayed in a new editor window.

VCS pop-up menu



This pop-up menu lets you use revision control with the source file you are viewing. To learn more about how to use this feature, refer to the discussion in [“Common VCS Operations” on page 613](#).

Status area

The status area displays various status messages and other information. For example, when you select a class from the Classes pane, the status area displays the base class(es) for the selected class.

Multi-Class Hierarchy Window

The Multi-Class Hierarchy window displays a complete graphical map of the classes in the browser database. Each class name is displayed in a box, and related classes are connected to each other by lines. Choose [Class Hierarchy Window](#) from the [Window Menu](#) to display the Multi-Class Hierarchy window. [Figure 7.7](#) shows the Multi-Class Hierarchy window.

Use the arrow keys to change the selected class “geographically.” The up and down keys work on siblings. The left and right keys work on ancestors and descendants.

You can use the keyboard to change the selection. The current selection most closely matches the characters you type. Use the Tab key to change the selected class alphabetically.

TIP If you select a class in the [Classes pane](#) in the [Browser Window](#), the selection in the [Multi-Class Hierarchy Window](#) changes too.

If you double-click a class entry, or select the entry and press the Enter/Return key, you open a [Browser Window](#) for that class.

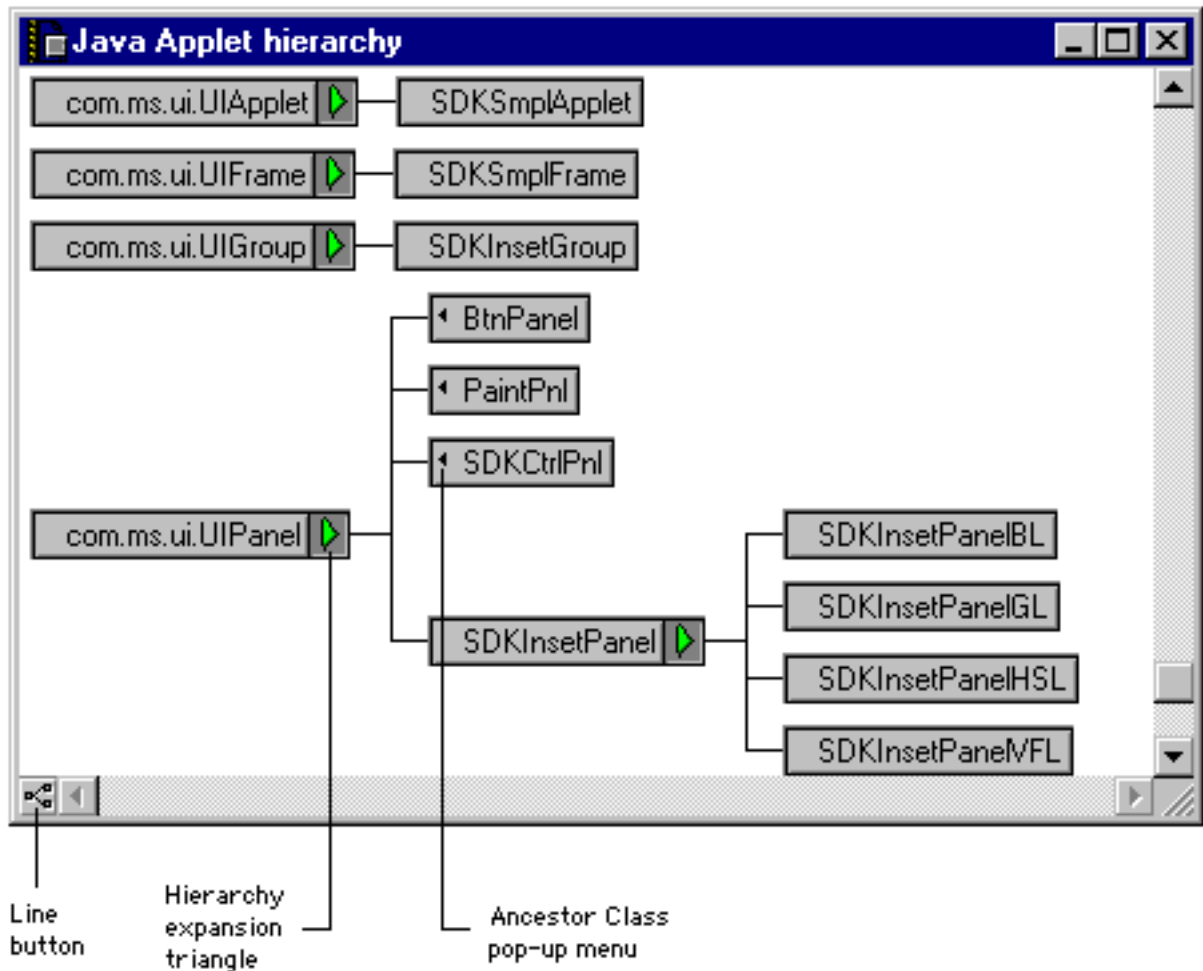
In addition to the entry for each class, the Multi-Class Hierarchy window has three items:

- [Line button](#)
- [Hierarchy expansion triangle](#)
- [Ancestor Class pop-up menu](#)

Line button

The Line button controls the display of the lines that connect related classes. You can toggle between diagonal lines and straight lines. The choice is entirely aesthetic.

Figure 7.7 The Multi-Class Hierarchy window



Hierarchy expansion triangle

The hierarchy expansion triangle controls the display of subclasses.

If you click this triangle, the next level of subclasses is displayed or hidden from view. To be more precise, the expanded state restores to what it was the last time this class was open.

NOTE Alt/Option click a disclosure triangle to open all subclasses at all levels. This is a “deep” disclosure. Ctrl/Command click to display subclasses for a class and all of its siblings. This is a “wide” disclosure. Ctrl-Alt-click (Windows) or Command-Option-click

(Mac OS, Solaris, and Linux) to open both wide and deep. You can use Ctrl/Command-Alt/Option-click to expand or collapse an entire map if you click the expansion triangle for a base class that has no ancestors.

Ancestor Class pop-up menu

Click on the Ancestor Class triangle, shown in [Figure 7.7](#), to display a pop-up menu. This menu lists immediate ancestors. When you choose an item from the pop-up menu, you jump to that class in the map. If the item is not currently visible, the computer beeps.

This control is displayed only for classes that have multiple base classes.

Single-Class Hierarchy Window

The Single-Class Hierarchy window displays a complete graphical map for a single class in the browser database. The map displays *all* immediate ancestors of the class, and all of its descendants. (The [Multi-Class Hierarchy Window](#) only shows one base class.)

[Figure 7.8](#) shows the Single-Class Hierarchy window, displaying multiple base classes and subclasses. The underlined class name is the focus of the window.

You can open a single class hierarchy view using several techniques:

- Use the [Browser Contextual Menu](#) in the [Contents Window](#)
- Use the [Browser Contextual Menu](#) in the [Multi-Class Hierarchy Window](#)
- Use the [Browser Contextual Menu](#) in the [Browser Window](#)
- Use the [Show Multi-Class Hierarchy window](#) button in the [Browser toolbar](#)

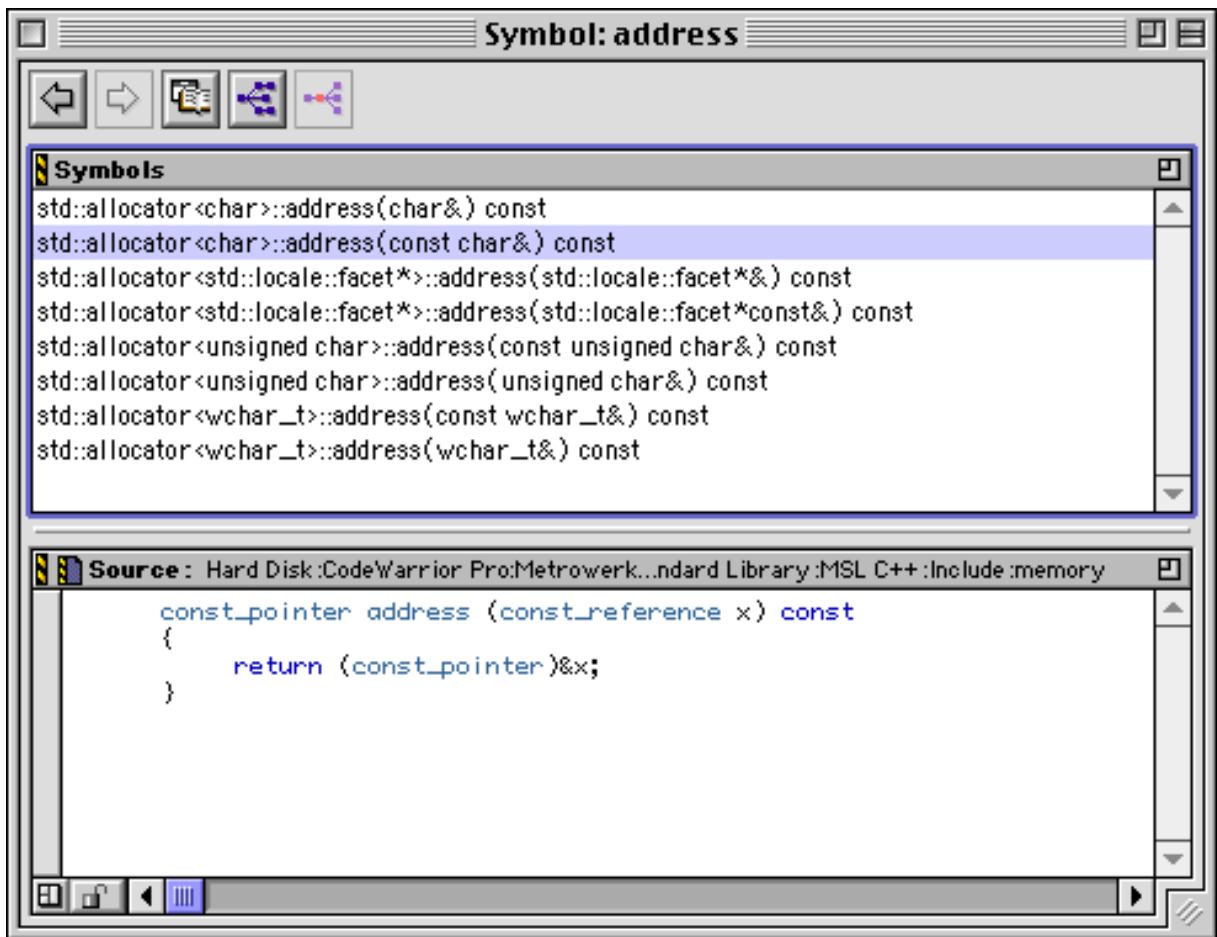
Figure 7.8 The Single-Class Hierarchy window

The [Single-Class Hierarchy Window](#) is identical to the [Multi-Class Hierarchy Window](#), except that it displays a limited map. For information about how this window behaves, see [“Multi-Class Hierarchy Window” on page 224](#).

Symbol Window

The Symbol window lists all implementations of any symbol that has multiple definitions. Most commonly, these symbols are multiple versions of overridden functions in object-oriented code. However, the Symbol window works for *any* symbol that has multiple definitions in the browser database.

Figure 7.9 The Symbol window



After selecting an implementation in the Symbol window list, you see its definition in the Source pane. [Figure 7.9](#) shows the Symbol window.

You open a Symbol window by right-clicking (Windows) or clicking and holding (Mac OS, Solaris, and Linux) on a symbol name in any

browser or editor window for which there is information in the browser database. When you do, a [Browser Contextual Menu](#) is displayed. If the item has multiple implementations, choose **Find all implementations of**. When you choose that item, the Symbol window displays.

TIP In a [Source pane](#) or editor window, Alt/Option-double-click or Ctrl/Command-double-click a function or other symbol name to find all implementations and open the Symbol window without using the contextual menu.

Most of the items in this window are identical to the [Browser Window](#). For a discussion of the window in general, see “[Browser Window](#)” on page 216. That topic includes discussion of these items, also found in the Symbol window:

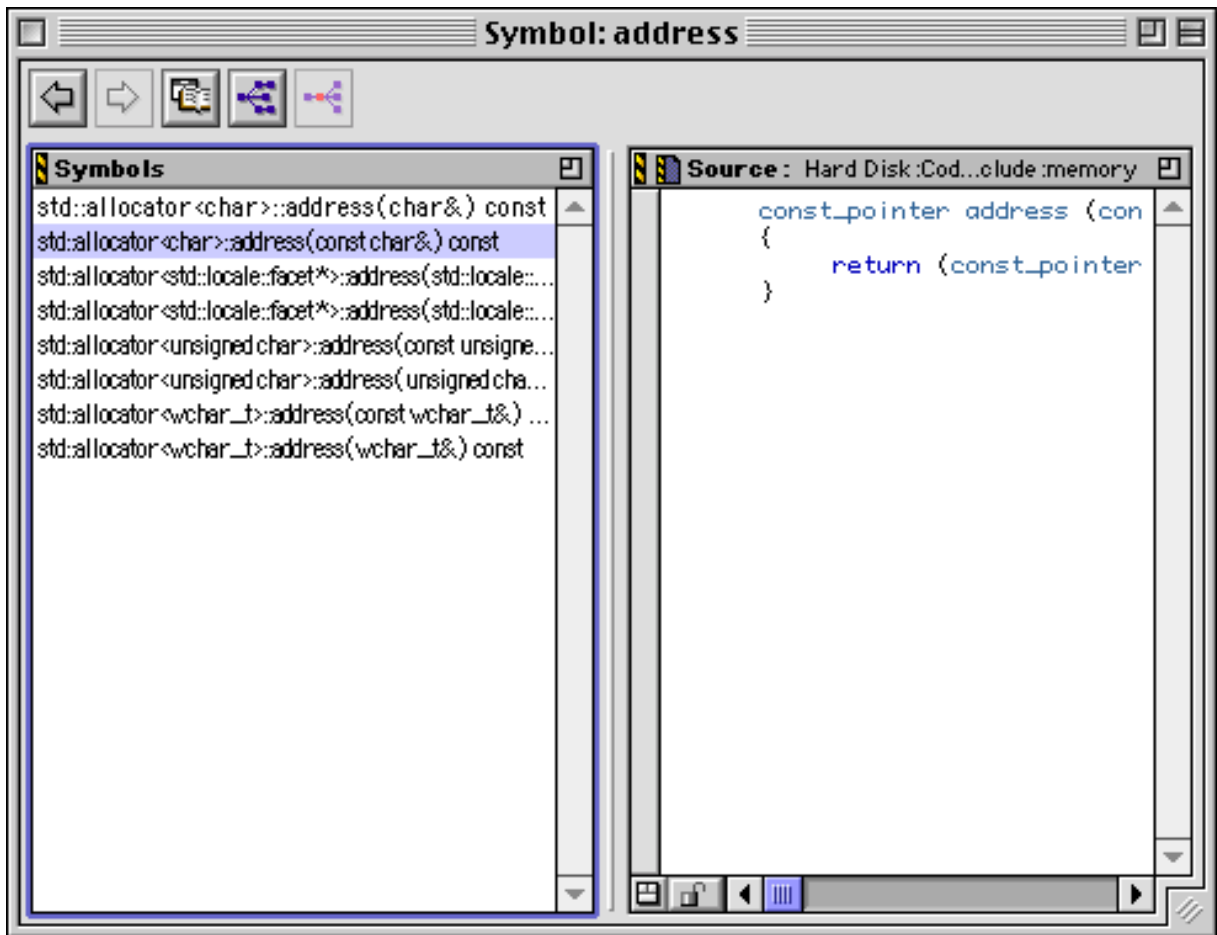
- [Browser toolbar](#)
- [Resize bar](#)
- [Source pane](#)
- [Open File icon](#)
- [VCS pop-up menu](#)

The Symbol window also has two unique items: the Symbols pane and the Orientation button.

The Symbols pane has the focus outline in [Figure 7.9 on page 227](#). This pane lists all versions of a symbol in the database. When you select an item in the Symbols pane, that item’s definition appears in the [Source pane](#).

The Orientation button toggles the position of the Symbols pane and the Source pane. The position toggles between [Figure 7.9](#) and [Figure 7.10](#).

Figure 7.10 The Symbol window in vertical orientation



Browser Menu

When a [Contents View](#), [Browser View](#), or [Hierarchy View](#) is visible, the Browser menu is displayed in CodeWarrior's menu bar. You use the commands in this menu to create new items. The IDE opens a wizard based on the command you select. The wizard presents various options to help you create a new class, member function, or data member.

The following list describes the commands in the Browser menu:

- **New Class**—opens the [New Class Wizard](#) to help you create a new class. The wizard lets you specify the name, location, file type, and modifiers for the new class.

- **New Member Function**—opens the [New Member Function Wizard](#) to help you create a new member function for a selected class. The wizard lets you specify the name, return type, parameters, modifiers, and other optional information for the new member function.
- **New Data Member**—opens a the [New Data Member Wizard](#) to help you create a new data member for a selected class. The wizard lets you specify the name, type, initializer, modifiers, and other optional information for the new data member.

Browser Wizards

When creating a new class, member function, or data member, you select a Browser wizard. The wizard helps you complete the item-creation process. This section describes the following wizards:

- [New Class Wizard](#)
- [New Member Function Wizard](#)
- [New Data Member Wizard](#)

For beginners

Each wizard assumes you have a basic understanding of the associated programming language. If the items in the wizard are unfamiliar to you, consult additional programming language references for more information.

The Browser wizards include the following navigation buttons:

- **Back**—return to the previous step
- **Next**—proceed to the subsequent step
- **Finish**—display a summary of current information
- **Cancel**—discard all changes

Each wizard is divided into a series of steps. You progress through these steps in sequence, and each step builds on the information provided in previous steps. When you supply enough information in a particular step, click **Next** to continue.

You can modify the default settings in each step. To accept all current settings in the wizard, click **Finish**. The wizard displays a summary of all the current settings for the new project. Click

Generate to accept the current settings and create the new RAD item, or click **Cancel** to return to the wizard and continue modifying settings.

New Class Wizard

The New Class wizard is divided into the following steps:

1. [Describe the name and location for the new class.](#)
2. [Specify base classes and methods for the new class.](#)
3. [List #include files for the new class.](#)
4. [Assign the new class to the project's build targets.](#)

To use the New Class wizard, follow these steps:

1. **Describe the name and location for the new class.**

This section of the wizard, shown in [Figure 7.11](#), lets you specify the name, declaration, and location for the new class. Other options are available to further describe the class.

This section includes the following parts:

- [Class Name](#)
- [Declaration File](#)
- [Namespace](#)
- [Use separate file for member definitions](#)

Class Name

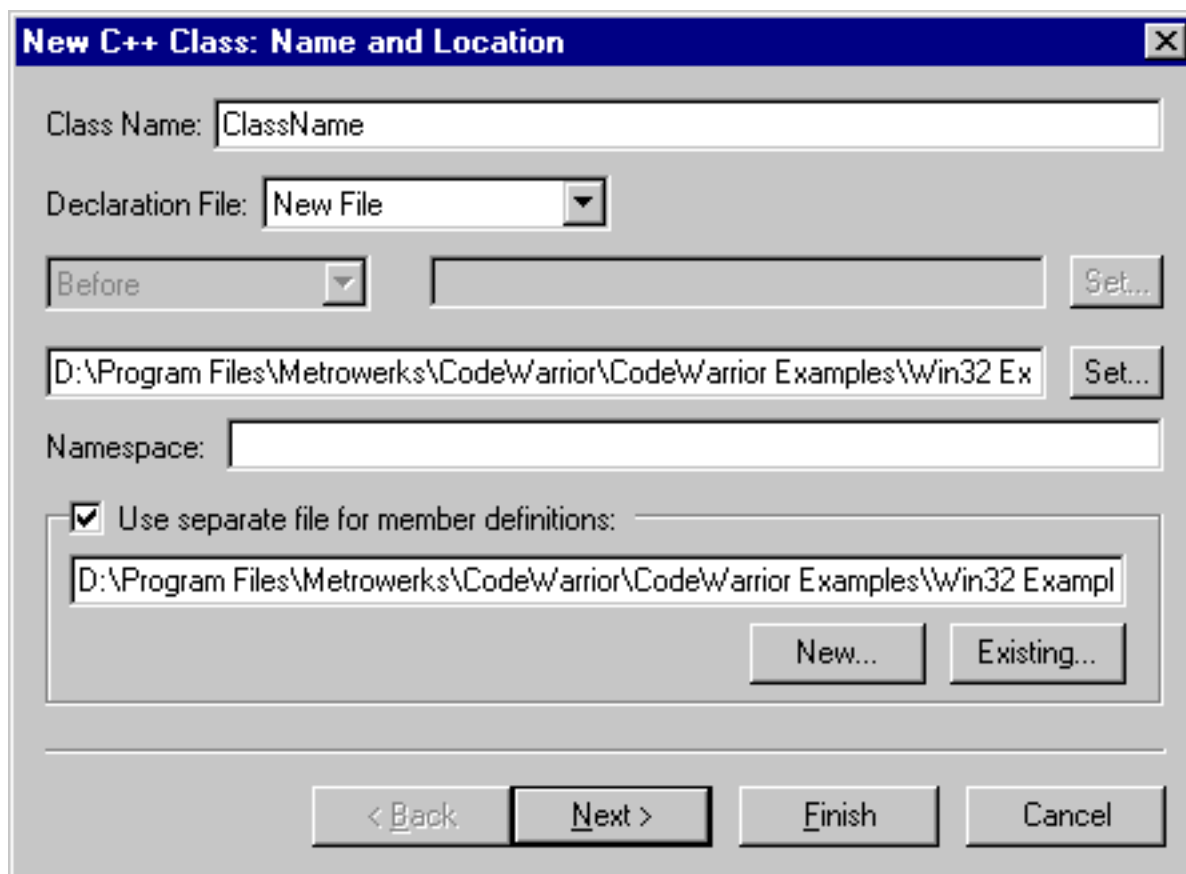
Enter a name for the class in this field.

Declaration File

This pop-up menu lets you specify the type of declaration file:

- **New File**—a declaration file
- **Relative to class**—a declaration file that depends on an existing file in the project

Figure 7.11 New Class wizard - Name and Location



Depending on the option you choose, different fields become enabled below the **Declaration File** pop-up menu. The enabled fields for the **New File** option are shown in [Figure 7.11](#), and the enabled fields for the **Relative to class** option are shown in [Figure 7.12](#).

If you choose the **New File** option, type in the enabled field the path to which you want to save the file. Alternatively, click **Set** next to the field to save the file using a standard Save window.

If you choose the **Relative to class** option, the pop-up menu and the **class** field beside it become enabled, as shown in [Figure 7.12](#). In the **class** field, type the name of the class you want to relate to the new class. Alternatively, click **Set** next to the **class** field, select a class in the window that opens, and then click **Select**. Next, use the pop-up

menu to place the new class **Before** or **After** the class in the **class** field.

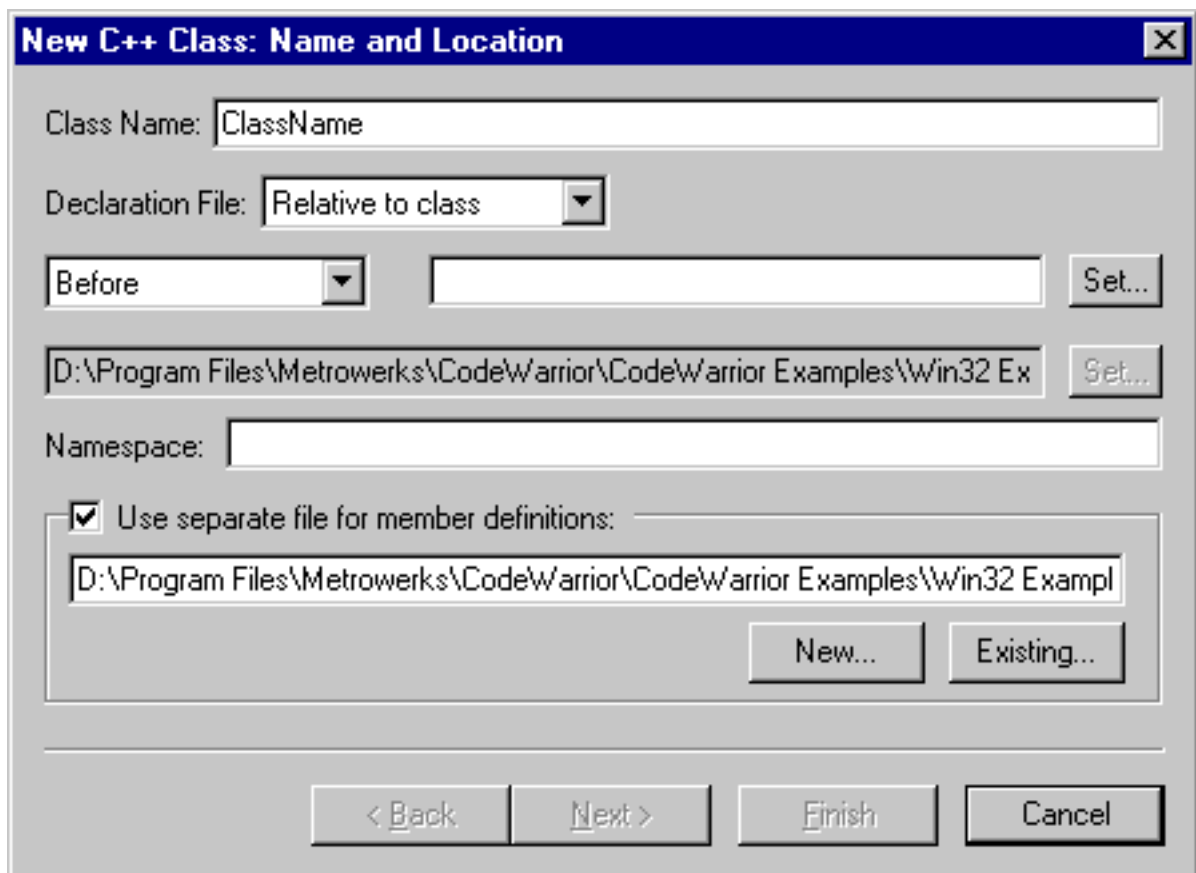
Namespace

In this field, type a namespace for the new class. Each name in the namespace should be unique.

Use separate file for member definitions

Enable this checkbox if you want to use a separate file to define the members of the new class. Type the path to the separate file in the field below the checkbox. Alternatively, click **Existing** to select the file using a standard dialog box. To create a new separate file, click **New** and save the new file to a location on your hard disk.

Figure 7.12 Choosing the Relative to class option



2. **Specify base classes and methods for the new class.**

This section of the wizard, shown in [Figure 7.13](#), lets you define base class, methods, and additional information for the new class.

This section includes the following parts:

- [Base Classes](#)
- [Generate Constructor and Destructor](#)
- [Namespaces required for base classes and constructor parameters \(optional\)](#)

Base Classes

Enter in this field a list of base classes for the new class. Some example base classes are provided.

Figure 7.13 New Class wizard - Base Class and Methods

The screenshot shows a Windows-style dialog box titled "New C++ Class: Base Classes and Methods". It contains several input fields and checkboxes. The "Base Classes:" field has the text "public FirstBaseClass" and an example "Example: public FirstBaseClass, virtual SecondBaseClass, ...". Below this is a checkbox labeled "Generate Constructor and Destructor" which is checked. Underneath is an "Access:" dropdown menu set to "Public". The "Constructor parameters:" field has the text "int inSuperParam1" and an example "Example: int inSuperParam1, bool inNewParam, ...". Below this is another checked checkbox labeled "Virtual destructor". At the bottom, the "Namespaces required for base classes and constructor parameters (optional):" field has the text "nspace1" and an example "Example: nspace1, nspace2, ...". At the very bottom are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Generate Constructor and Destructor

Enable this checkbox to generate a constructor and destructor for the new class. After enabling this checkbox, the following options become available:

- **Access**—Choose an access type for the new class from this pop-up menu. Three access types are available: **Public**, **Protected**, and **Private**.
- **Constructor parameters**—If desired, type a list of parameters for the constructor in this field. Example parameters are listed above the field.
- **Virtual destructor**—If you wish, enable this checkbox to create a virtual destructor for the new class.

Namespaces required for base classes and constructor parameters (optional)

If you wish, you can enter in this field required namespaces for the base classes from the [Base Classes](#) field and the constructor parameters in the [Generate Constructor and Destructor](#) part of the wizard.

3. List `#include` files for the new class.

This section of the wizard, shown in [Figure 7.14](#), lets you specify additional header `#include` files for the new class.

This section includes the following parts:

- [Include files that will automatically be added for base classes](#)
- [Additional header include files](#)

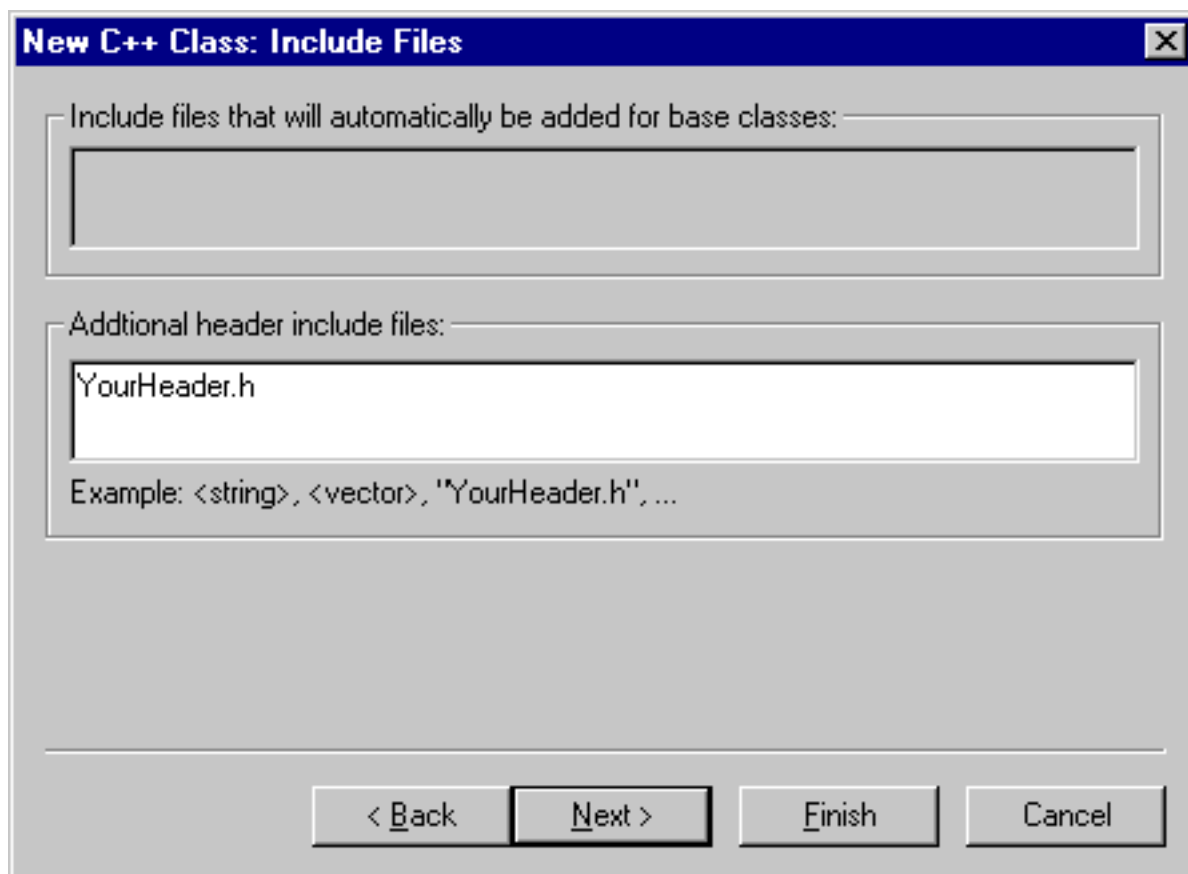
Include files that will automatically be added for base classes

This field shows you a list of `#include` files that will be automatically added to the base classes.

Additional header include files

Enter in this field a list of other `#include` files for the new class, in addition to those listed in the previous field. Separate each file in the list with a comma. Sample files are listed below the field.

Figure 7.14 New Class wizard - Include Files



4. **Assign the new class to the project's build targets.**

This section of the wizard, shown in [Figure 7.15](#), lets you assign the new class to particular build targets within the active project.

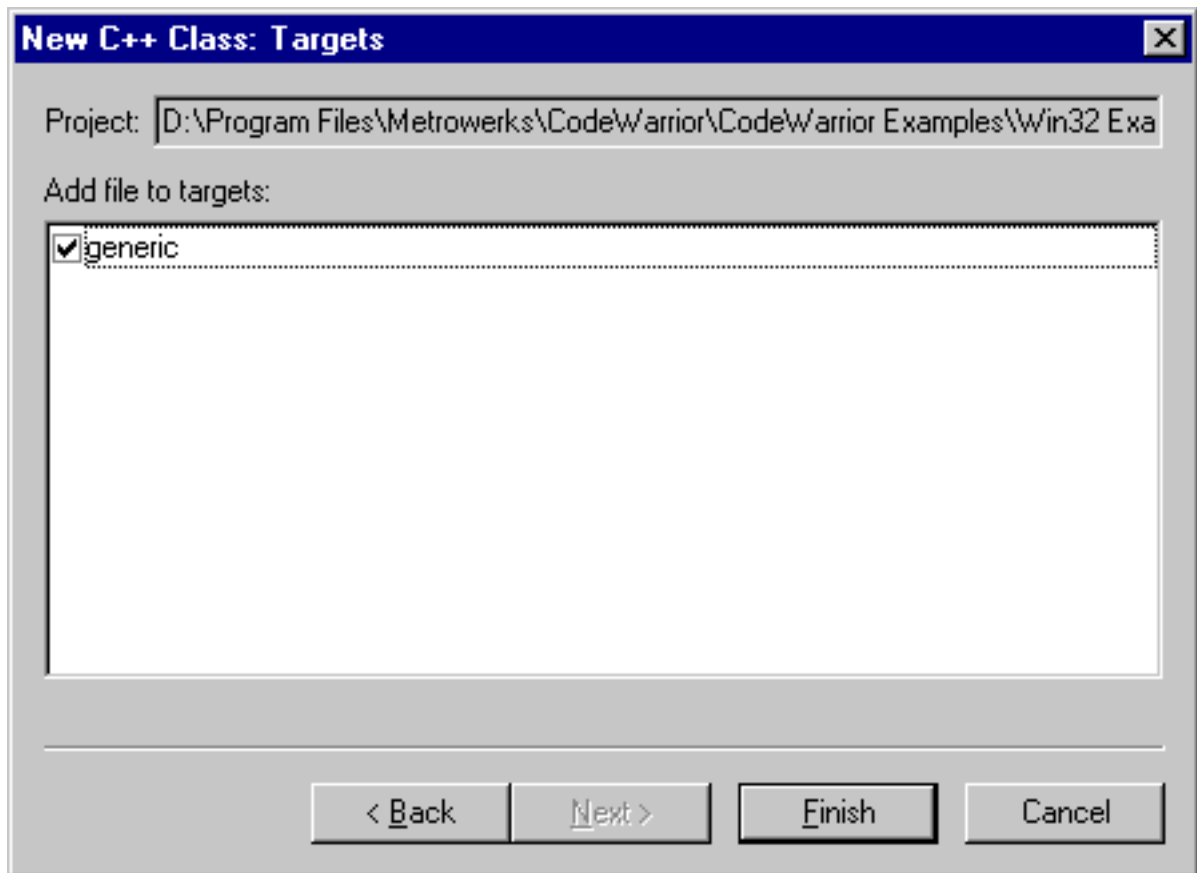
This section includes the following parts:

- [Project](#)
- [Add files to targets](#)

Project

This field shows you the path to the project whose build targets are listed in the next field.

Figure 7.15 New Class wizard - Targets



Add files to targets

To assign the class to a specific build target, enable the checkbox next to the build target's name in this list. For example, in [Figure 7.15](#), the new class is assigned to the **generic** build target.

New Member Function Wizard

The New Member Function wizard is divided into the following steps:

1. [Describe the declaration of the new member function.](#)
2. [Specify file locations for the new member function.](#)

Figure 7.16 New Member Function wizard - Member Function Declaration

The screenshot shows a Windows-style dialog box titled "New C++ Member Function". Inside, there's a tab labeled "Member Function Declaration". The dialog contains several input fields and options:

- Name:** A text box containing "MemberFunctionName".
- Return type:** A text box containing "void".
- Parameters:** A text box containing "int inParam1". Above this box is an example: "Example: int inParam1, bool, char& outParam3, ...".
- Namespaces required for parameters (optional):** A text box containing "nspace1". Above this box is an example: "Ex: nspace1, n2, ...".
- Modifiers:** A section containing:
 - Access:** A dropdown menu set to "Public".
 - Specifier:** A dropdown menu set to "None".
 - ☐ **Inline**
 - ☐ **Const**

At the bottom of the dialog are four buttons: "Cancel", "Back", "Next", and "Finish".

To use the New Member Function wizard, follow these steps:

1. **Describe the declaration of the new member function.**

This section of the wizard, shown in [Figure 7.16](#), lets you specify the name, return type, and parameters for the new member function. Other options are available to further describe the function.

This section includes the following parts:

- [Name](#)
- [Return Type](#)
- [Namespaces required for parameters \(optional\)](#)
- [Namespaces required for parameters \(optional\)](#)

- [Modifiers](#)

Name

Type a name for the member function in this field.

Return Type

Enter an appropriate function return type in this field.

Namespaces required for parameters (optional)

If you wish, type a list of function parameters in this field. Example parameters are listed above the field.

Namespaces required for parameters (optional)

If desired, enter a list of namespaces required for the parameters in the [Namespaces required for parameters \(optional\)](#) field. Sample parameters are listed above the field.

Modifiers

Use the **Access** and **Specifier** pop-up menus to choose the access level and method specifier for the new method. Possible access levels include **Public**, **Protected**, and **Private**. Possible specifiers include **None**, **Virtual**, **Pure Virtual**, and **Static**. You can enable the **Inline** or **Const** checkboxes as desired to further describe the function's modifiers.

2. Specify file locations for the new member function.

This section of the wizard, shown in [Figure 7.17](#), lets you specify file locations associated with the new member function.

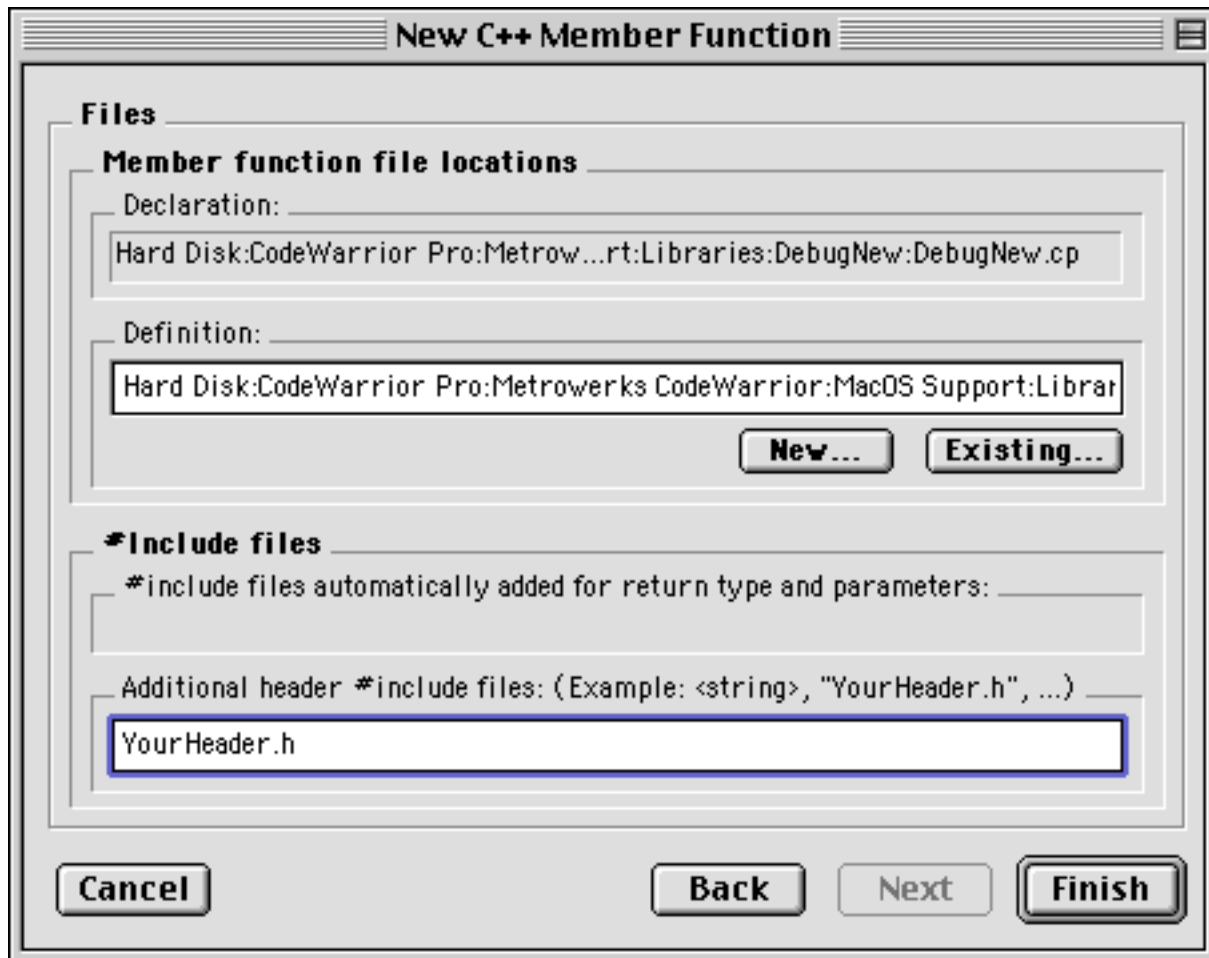
This section includes the following parts:

- [Declaration](#)
- [Definition](#)
- [Include files automatically added for return type and parameters](#)
- [Additional header include files](#)

Declaration

This field shows you the location of the file to which the member function's declaration will be added.

Figure 7.17 New Member Function wizard - File Locations



Definition

Enter in this field the path to file used for the member function's definition. Alternatively, click **Existing** to select the file using a standard dialog box. To create a new file to use for the member function's definition, click **New** and save the new file to a location on your hard disk.

Include files automatically added for return type and parameters

This field shows you a list of `#include` files that will be automatically added to the member function. These files are automatically added based on the return type and parameters you specified from the previous section.

Additional header include files

Enter in this field a list of other `#include` files for the new member function, in addition to those listed in the previous field. Some examples are provided above the field.

New Data Member Wizard

The New Data Member wizard is divided into the following steps:

1. [Describe the declaration of the new data member.](#)
2. [Specify file locations for the new data member.](#)

To use the New Data Member wizard, follow these steps:

1. **Describe the declaration of the new data member.**

This section of the wizard, shown in [Figure 7.18](#), lets you specify the name, type, and initializer for the new data member. Other options are available to further describe the data member.

This section includes the following parts:

- [Name](#)
- [Type](#)
- [Namespaces required for type \(optional\)](#)
- [Initializer](#)
- [Modifiers](#)

Name

Type a name for the data member in this field.

Type

Enter an appropriate data-member type in this field.

Namespaces required for type (optional)

If desired, enter a list of namespaces required for the type in the [Type](#) field. Sample parameters are listed above the field.

Figure 7.18 New Data Member wizard - Data Member Declaration

New C++ Data Member: Data Member Declaration

Name: DataMemberName

Type: void

Namespaces required for type (optional): std Example: std

Initializer: 100 Example: 100 or inConstructorParameterName

Modifiers:

Access: Protected Specifier: None

☐ Const ☐ Volatile

< Back Next > Finish Cancel

Initializer

If you wish, type an initial value for the data member in this field. Sample initializers are listed above this field.

Modifiers

Use the **Access** and **Specifier** pop-up menus to choose the access level and member specifier for the new data member. Possible access levels include **Public**, **Protected**, and **Private**. Possible specifiers include **None**, **Static**, and **Mutable**. You can enable the **Const** or **Volatile** checkboxes as desired to further describe the data member's modifiers.

Figure 7.19 New Data Member wizard - File Locations

New C++ Data Member: File Locations

Declaration:

Definition:

Include files:

Include file automatically added for member type:

Additional header include files: (Example: <string>, "YourHeader.h", ...)

2. Specify file locations for the new data member.

This section of the wizard, shown in [Figure 7.19](#), lets you specify file locations associated with the new member function.

This section includes the following parts:

- [Declaration](#)
- [Definition](#)
- [Include file automatically added for member type](#)
- [Additional header include files](#)

Declaration

This field shows you the location of the file to which the data member's declaration will be added.

Definition

This field is not available in this wizard.

Include file automatically added for member type

This field shows you the `#include` file that is automatically added for the data-member type.

Additional header include files

Enter in this field a list of other `#include` files for the new data member, in addition to the file listed in the previous field. Some examples are provided above the field.

Using the Browser

The browser provides multiple paths through the data related to your code. This section can give you a feel for how to work with the browser, and outline some techniques you can use to accomplish common tasks.

Topics in this section include:

- [Setting Browser Options](#)
- [Identifying Symbols in the Browser Database](#)
- [Navigating Code in the Browser](#)
- [Browsing Across Subprojects](#)
- [Completing Symbols](#)
- [Opening a Source File](#)
- [Seeing a Declaration](#)
- [Seeing a Routine Definition](#)
- [Editing Code in the Browser](#)
- [Analyzing Inheritance](#)
- [Finding Functions That Are Overrides](#)

- [Viewing MFC and PowerPlant Classes](#)
- [Saving a Default Browser](#)

Setting Browser Options

Browser-related menu items and browser-specific options become available when you activate the browser. See [“Activating the Browser” on page 207](#) for information on how to enable the browser.

When the browser is enabled for your project, browser-related menu commands are enabled. These items include the [Browser Contents](#), [Class Hierarchy Window](#), and [New Class Browser](#) commands in the [Window Menu](#).

TIP A quick way to tell whether the browser is enabled is to look in the [Window Menu](#) at these browser-related menu commands. If they are enabled, the browser is active.

In addition, there are global IDE options that relate to the browser. You control how various items are colored in browser windows, and the time delay before the [Browser Contextual Menu](#) is displayed (Mac OS, Solaris, and Linux).

To tell the IDE to include items from a project’s subprojects in its browser windows, see [“Cache Subprojects” on page 338](#).

To learn how to modify these settings, see [“Browser Display” on page 272](#) or [“Context popup delay” on page 277](#).

Identifying Symbols in the Browser Database

There is an easy way to know whether or not a symbol is in the browser database without checking to see if a [Browser Contextual Menu](#) displays: use browser coloring. If the browser is activated, symbols that are in the browser database are displayed in editor and browser windows according to the colors you select. See [“Browser Display” on page 272](#) for more information.

TIP The factory default color setting is identical for all of the eight types of browser-database symbols. You can choose a different color for each symbol type if you like. However, if syntax coloring is also enabled for your code, you may find it easier to identify symbols in the browser database using only one or two colors.

Navigating Code in the Browser

There are many ways to move around in code with the browser.

Using the Contextual Menu

Perhaps most powerful and flexible way to navigate code is to use the [Browser Contextual Menu](#). You see this menu when you right-click (Windows) or click and hold (Mac OS, Solaris, and Linux) on any symbol for which there is data in the browser database. This includes class names, routine names, global variables, class data members, and much more. To learn more, refer to [“Browser Contextual Menu” on page 213](#).

In the [Browser Window](#), simply selecting a class, routine, or data member displays the associated code in the window’s [Source pane](#).

Go Back and Go Forward

The browser fully supports the [Go Back](#) and [Go Forward](#) commands in the [Search Menu](#). Regardless of the views, windows, or code you have looked at, you can always go back to what you were viewing earlier.

These commands allow you to go backward or forward through a series of changes you made. For example, suppose you use the browser to look at your project and make changes to a file. Then, you switch files and make more changes. You may do this many times. You can use the [Go Back](#) command to go back one or more actions you have performed. Even if you did not make any changes to the file, but looked at it (or a specific class or method), you can go back to that action. Similarly, once you have gone back, you can use the [Go Forward](#) command to return to where you started.

When you add the [Go Back](#) and [Go Forward](#) commands to a CodeWarrior toolbar, you can use their associated pop-up menus,

as shown in [Figure 7.20](#). Click and hold on the [Go Back](#) or [Go Forward](#) toolbar icons to display the pop-up menu.

Figure 7.20 Go Back and Go Forward toolbar buttons

Windows Choose an action from the menu to go to that action. If you choose an action out of sequence, CodeWarrior will go to that action *without* going through any previous action.

Mac OS The underlined item is your current position in the queue. Choose an action from the menu to go to that action. If you choose an action out of sequence, CodeWarrior will go to that action *without* going through any previous action. The CodeWarrior IDE can track up to 100 actions.

NOTE **Go Back** and **Go Forward** do *not* undo any actions you performed. They allow for a more flexible method of moving around to specific places you have been in the Browser window.

Browsing Across Subprojects

The IDE normally only displays browser items for the current build target. To include browser information from the subprojects of the current build target, enable subproject caching. For more information, see [“Cache Subprojects” on page 338](#).

Completing Symbols

The IDE has commands to complete your typing for you when you enter the name of an item for which the browser has information. Use the **Find symbols with prefix**, **Find symbols with substring**, **Get next symbol**, and **Get previous symbol** keyboard commands to find and choose browser items that match the text you have selected or just entered in a source code file. These commands are only available from the keyboard. They are not available in the IDE’s menus. See [“Editor Commands” on page 671](#) for a description of these keyboard commands.

To enter the name of a browser item that has the same characters as the text you have selected or just typed, use the **Find symbols with substring** keyboard command. To enter the name of a browser item

that only has the same first characters, use the **Find symbols with prefix** keyboard command.

After using the **Find symbols with substring** and **Find symbols with prefix** commands, use the **Get next symbol** and **Get previous symbol** commands to search among the browser symbols that match the text you have selected or just entered.

After you find the browser item you want to enter, press the right arrow key to place the insertion point next to the item and continue typing.

Another way to find and enter a browser item is to select the first few characters of some text and then right-click (Windows) or click and hold (Mac OS, Solaris, and Linux) on the selection. The Browser contextual menu displays a list of matching items, as shown in [Figure 7.21](#). Choose one of these items to automatically replace the selected text. See [“Browser Contextual Menu” on page 213](#) for more information.

Figure 7.21 Automatically replacing selected text



Opening a Source File

There are some quick methods you can use to open a source file.

In the [Browser Window](#), click the [Open File icon](#) when portions of the file you wish to see are displayed in the window's [Source pane](#). See [“Open File icon” on page 223](#) for more information.

To select a file, click on a symbol used in that file. Then use the [Browser Contextual Menu](#) to open the desired file or see a particular routine.

If you are looking at a source file, and want to see the interface file, or vice versa, simply type Ctrl/Command ` to see the related file.

Seeing a Declaration

There are several methods used to view a declaration. The methods vary depending on the kind of symbol you investigate.

If you select a class name or data member name in a [Browser Window](#), and the declaration is shown in the [Source pane](#), double-click the name to open the file that contains the declaration. (If you select or double-click a routine name, you see the definition).

If you click the [Class Declaration button](#) in the Browser window, you see a class declaration in the [Source pane](#).

When you right-click (Windows) or click and hold (Mac OS, Solaris, and Linux) on a name in any window, use the [Browser Contextual Menu](#) to open the declaration. This technique lets you see a routine declaration.

Seeing a Routine Definition

There are several methods you can use to see a routine definition:

- In the [Browser Window](#), select the routine in the [Member Functions pane](#). The definition is displayed in the [Source pane](#). To open the file that contains the definition, double-click the routine name in the Member Functions pane.
- Right-click (Windows) or click and hold (Mac OS, Solaris, and Linux) on the routine name in any editor or browser window. Then use the [Browser Contextual Menu](#) to jump to the particular routine definition.
- Choose the [Go Back](#) command from the [Search Menu](#).
- Press Alt/Option or Ctrl/Option, then double-click a function name in any source view. The [Symbol Window](#) displays for functions with multiple definitions to show all implementations of that function.
- In the [Multi-Class Hierarchy Window](#) or the [Single-Class Hierarchy Window](#), right-click (Windows) or click and hold (Mac OS, Solaris, and Linux) on a class name. Then use the

[Browser Contextual Menu](#) to jump to the particular routine definition.

Editing Code in the Browser

Any code visible in a [Source pane](#) is fully editable. Locate the definition with which you wish to work. When the code is displayed in the [Source pane](#), use the same techniques you learned in the editor window to edit your code.

For more information about the CodeWarrior editor, see [“Editing Source Code” on page 137](#).

Analyzing Inheritance

Use the [Multi-Class Hierarchy Window](#) or the [Single-Class Hierarchy Window](#) to analyze inheritance in your source code. Look for the small triangle to the left of a class name, as shown in [Figure 7.7](#). This triangle indicates that the class has multiple ancestors. Use the associated [Ancestor Class pop-up menu](#) to jump to any ancestor class to study its descendants (or ancestors). Use the [Hierarchy expansion triangle](#) to expose or conceal subclasses.

For more information, see [“Multi-Class Hierarchy Window” on page 224](#).

Finding Functions That Are Overrides

Use the [Browser Window](#) to find functions that are overrides. To find these overrides, disable the **Show Inherited** checkbox in the [Browser Window](#) toolbar. Disabling the checkbox disables the display of unchanged inherited functions. However, it does not disable functions you have overridden.

Next, look for functions that are marked as virtual with an [Identifier icon](#). Most of these functions are likely to be overrides of inherited functions, although some could be functions you declared in the class that were not inherited from an ancestor.

To open a [Symbol Window](#) for any routine with multiple definitions, right-click (Windows) or click and hold (Mac OS, Solaris, and Linux) on a routine name, and then choose **Find all**

implementations of. Combined with a Hierarchy view, you see precisely who overrides the routine, and where the routines are found in the class hierarchy.

Viewing MFC and PowerPlant Classes

The browser lets you view framework classes, such as the Microsoft Foundation Classes (MFC) for Windows and the PowerPlant classes for the Mac OS. If you want the browser to display these framework classes, you must include framework headers in such a way that the compiler sees them.

You should use precompiled headers to speed compilation time. However, if you simply use the precompiled headers directly, the compiler does not see the framework classes and does not generate information for the symbol database.

To resolve this problem, include a renamed copy of the source file that creates the precompiled headers in your project. These source files are named with extensions that identify precompiled headers, like `.pch` or `.pch++`. Alternatively, include as a subproject the project that creates the precompiled headers.

Including these files creates a project-specific precompiled header. CodeWarrior builds this precompiled header from within your project, exposing framework symbols to the compiler, which generates information for the browser database.

The reason you want to include a renamed copy is to avoid problems associated with multiple projects that use the same precompiled headers. When one project updates the headers, it will be marked as changed for all other projects, which then rebuild the precompiled headers. This rebuilding step defeats the purpose of the precompiled headers.

Saving a Default Browser

Browser windows have various settings that you can modify. You can preserve these modifications as your default settings.

First, set up a browser window to your liking. For example, set the size of each pane in the Browser window, and the size and location

of the window itself. Then choose [Save Default Window](#) from the [Window Menu](#). The next time you open a Browser window, it takes on the attributes you saved.

You can do the same thing for any of the browser windows. However, you must save each window's setup individually, while that window is active.

To learn about saving editor windows, see ["Saving Editor Window Settings" on page 148.](#)

Configuring IDE Options



This chapter discusses the options available in the CodeWarrior IDE [Preferences](#) window. In addition, this chapter discusses IDE window toolbars and their configuration.

You can customize CodeWarrior's features through the IDE Preferences window. These global preference settings affect the way the IDE works in all projects.

To view the IDE Preferences window, choose the [Preferences](#) command from the [Edit Menu](#). The preferences are organized into a series of panels devoted to a particular topic. For example, one panel controls the font and tab settings in the CodeWarrior editor.

The IDE's toolbars are another powerful and flexible feature. You can fully customize the toolbars to fit your own working style and improve your efficiency.

The topics in this chapter include:

- [Preferences Guided Tour](#)
- [Choosing Preferences](#)
- [Customizing the IDE](#)

Preferences Guided Tour

To open the IDE Preferences window, choose the [Preferences](#) command from the Edit menu.

The topics in this section are:

- [Preference Panels](#)
- [Dialog Box Buttons](#)

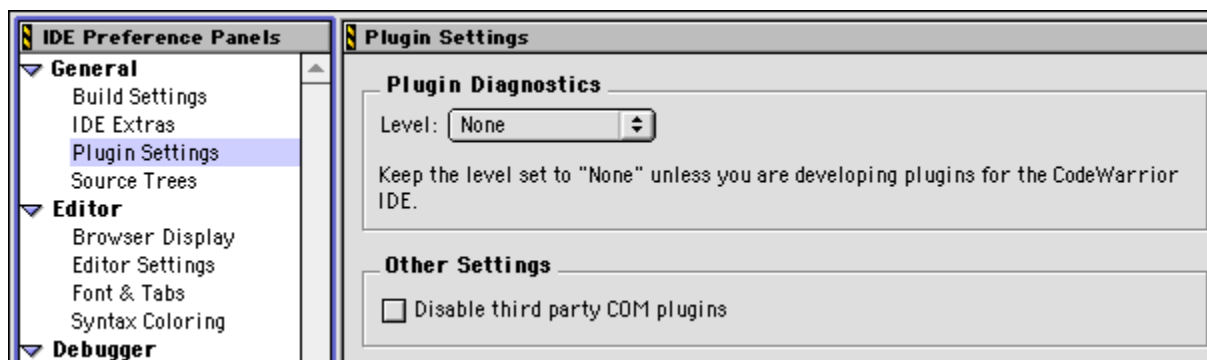
Preference Panels

The IDE Preferences window is divided into two parts. The left side of the window has a hierarchical list of available panels. The right side of the window displays the information from a selected panel. The preference panels available to you vary, depending on the CodeWarrior product you are using.

The preference panels affect the IDE as a whole and apply to all projects. To display a particular panel, select its name in the list. You can use the arrow keys or click the name of the panel. [Figure 8.1](#) shows a selected panel in the IDE Preferences window.

Each panel consists of a series of related options that you can modify. After changing these options, you can save them, discard them, restore them, or reset them. See [“Dialog Box Buttons”](#) for more information.

Figure 8.1 **Selecting a preference panel**



Dialog Box Buttons

There are several dialog box buttons in the IDE Preferences window that control how a panel's settings are used and applied.

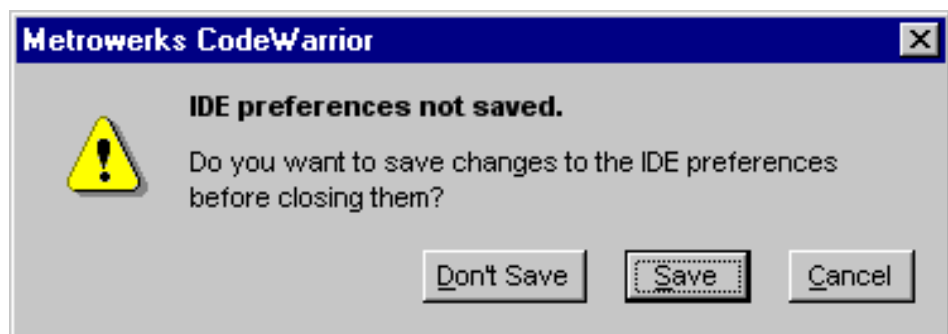
The topics in this section are:

- [Discarding changes](#)
- [Factory Settings button](#)
- [Revert Panel button](#)
- [Save button](#)

Discarding changes

If you change the settings in the IDE Preferences window and then try to close the window, the CodeWarrior IDE displays the dialog box shown in [Figure 8.2](#). To discard your changes, click **Don't Save**. To keep your changes, click the **Save** button. To discard your changes and return to the IDE Preferences window so that you can continue making changes, click **Cancel**.

Figure 8.2 Preferences Confirmation Dialog Box



Factory Settings button

The **Factory Settings** button causes the panel to revert to the settings that the CodeWarrior IDE uses as defaults. Settings in other panels are not affected by this button. Only the settings for the current panel are reset.

Revert Panel button

The **Revert Panel** button lets you undo changes in a preference panel. Clicking this button causes the currently selected preference panel to reset to its last saved state. This is useful when you decide against the changes you made.

Save button

The **Save** button commits all of the current settings in the panels. After closing the IDE Preferences window, the CodeWarrior IDE behaves according to the preferences that you saved.

Choosing Preferences

This section discusses setting preferences for the IDE as a whole. You can learn to configure preferences that affect the debugger, editor, and IDE in general.

To learn how to display a particular preference panel, see [“Preferences Guided Tour” on page 253](#).

In this section, we discuss the features of the IDE that are controlled by each preference panel. The panels are grouped into the following categories:

- [General Preferences](#)
- [Editor Preferences](#)
- [Debugger Preferences](#)

General Preferences

This section describes the preference panels that control general IDE features. These panels include:

- [Build Settings](#)
- [IDE Extras](#)
- [Plugin Settings](#)
- [Source Trees](#)

Build Settings

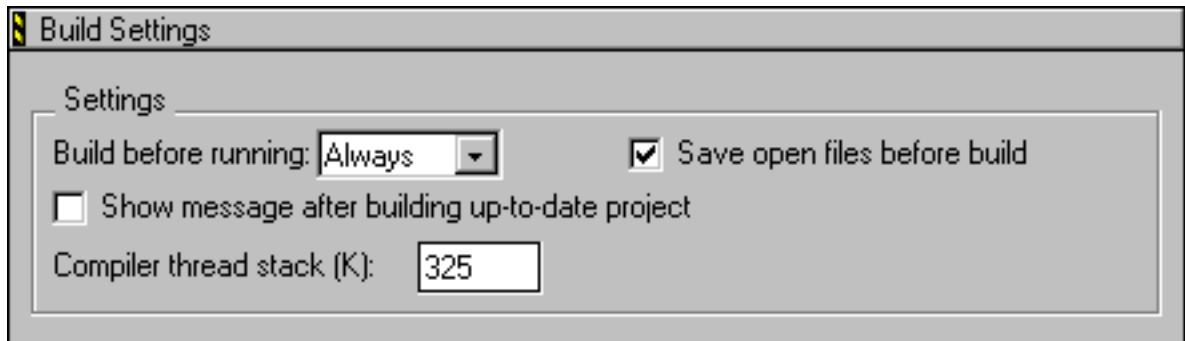
The Build Settings panel, shown in [Figure 8.3](#) (Windows) and [Figure 8.5](#) (Mac OS), provides project build customizations.

To learn how to open the IDE Preferences window and select the Build Settings panel, see [“Preferences Guided Tour” on page 253](#).

Build before running

This pop-up menu determines how the CodeWarrior IDE handles project builds. You can choose to **Always** build projects before running them, **Never** build projects before running them, or let the CodeWarrior IDE **Ask** you how to proceed.

Figure 8.3 Build Settings preference panel (Windows)



Save open files before build

Enable this checkbox if you want to save all open files automatically before a [Preprocess](#), [Precompile](#), [Compile](#), [Disassemble](#), [Bring Up To Date](#), [Make](#), or [Run](#) command is executed.

Show message after building up-to-date project

Enable this checkbox if you want the IDE to display a message after building an up-to-date project. The message appears in the Message window, as shown in [Figure 8.4](#). For more information about up-to-date projects, see [“Updating a Project” on page 366](#).

Figure 8.4 Message after building an up-to-date project

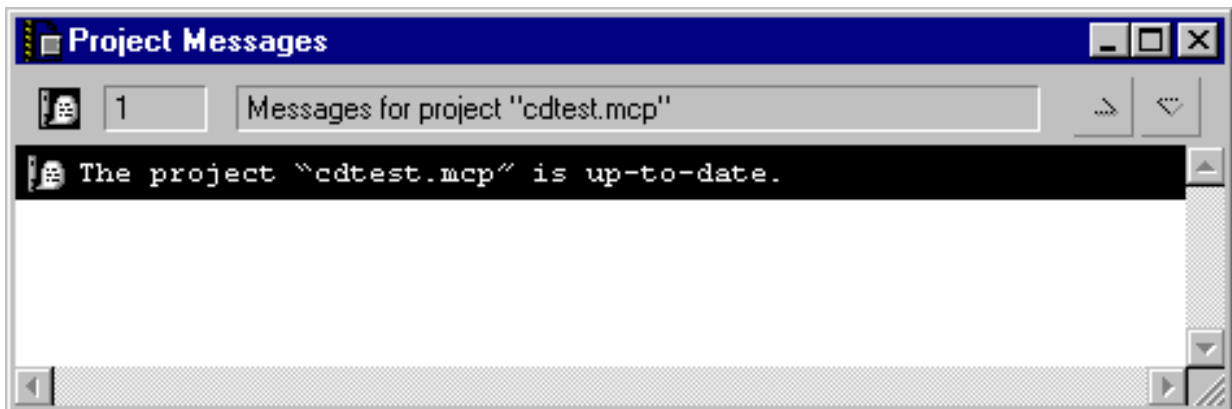


Figure 8.5 Build Settings preference panel (Mac OS)



Compiler thread stack

Use the **Compiler thread stack** field, shown in [Figure 8.3](#) (Windows) and [Figure 8.5](#) (Mac OS), to specify the upper limit of stack size allocated by the IDE for compiling and linking thread support.

All builds in CodeWarrior are threaded, with compilation and linking occurring on a thread separate from the main application thread. This setting lets you control the size of the stack allocated for the compiler thread.

Normally, you should not change this setting. However, if you have a large or very complex project, you can increase this setting to avoid compiler crashes.

Include file cache (Mac OS)

This checkbox, shown in [Figure 8.5](#), specifies the upper limit on how much memory the IDE should use for caching `#include` files and precompiled headers. If your computer has a lot of memory and you want to use that memory to speed up builds, increase the number in this field.

Play sound after 'Bring Up To Date' & 'Make' (Mac OS)

Enable this checkbox, shown in [Figure 8.5](#), to play a sound after finishing a build of your project. You can select different sounds for

both successful and unsuccessful build results by using the **Success** and **Failure** pop-up menus, respectively.

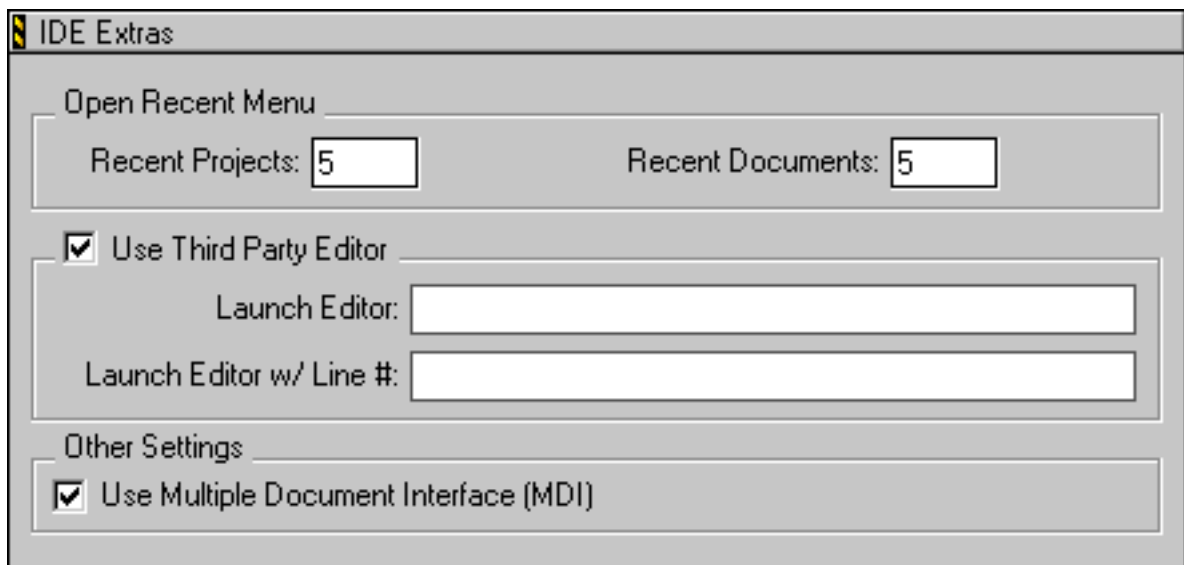
IDE Extras

The IDE Extras panel, shown in [Figure 8.6](#) (Windows) and [Figure 8.7](#) (Mac OS), has options to remember previously opened projects and text files and to configure the IDE for use with third-party editors.

To learn how to open the IDE Preferences window and select the IDE Extras panel, see [“Preferences Guided Tour” on page 253](#).

Mac OS The IDE Extras preference panel also lets you configure miscellaneous options, such as window zooming, the Script menu, automatic help, and online reference databases.

Figure 8.6 IDE Extras preference panel (Windows)



Recent Projects

Enter in this field the maximum number of recent projects that you want displayed in the [Open Recent](#) command of the [File Menu](#).

Recent Documents

Type in this field the maximum number of documents that you want displayed in the [Open Recent](#) command of the [File Menu](#).

Use Third Party Editor (Windows)

Use this checkbox to control whether the CodeWarrior text editor is used to display text files. When the checkbox is not enabled, the CodeWarrior editor is the default editor for your text files. When this checkbox is enabled, the IDE uses the third-party text editor you specify in the **Launch Editor** field to display text files.

There are two command lines used for invoking a third-party editor. The first, **Launch Editor**, specifies the text editor that the IDE uses to display text files. The second command line, **Launch Editor w/ Line #**, specifies a text editor and an initial line of text to jump to upon launch. For example, the IDE invokes this second command line when you double-click an error message to display the line in a text file that caused the error message.

You can use two variables in the command-line string: `%file` and `%line`. When the IDE encounters these variables, it expands `%file` into the full file path and `%line` into the initial line number in the file. For example, if you want to use the Emacs text editor to edit text files, then you would type the following command line in the **Launch Editor** field:

```
runemacs %file
```

If you also want the text editor to jump to a particular line in the text file, you would type the following command line in the **Launch Editor w/ Line #** field:

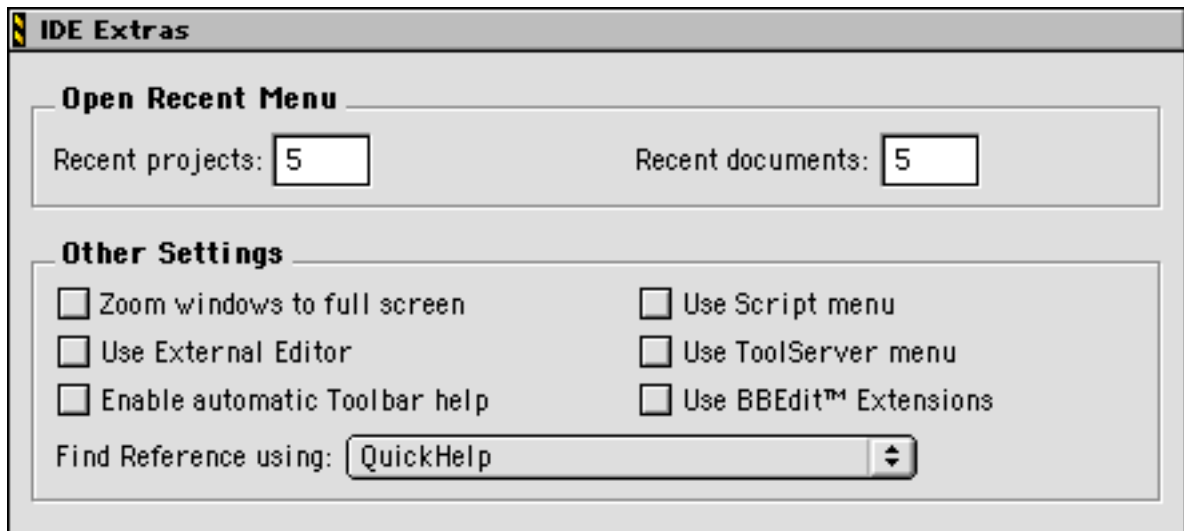
```
runemacs +%line %file
```

For more information about using command lines with a particular text editor, consult the documentation for that editor.

Use Multiple Document Interface (Windows)

Use this checkbox to set the Windows interface style used by the CodeWarrior IDE. Enable the checkbox to use MDI (Multiple Document Interface), or disable the checkbox to use FDI (Floating Document Interface).

Figure 8.7 IDE Extras preference panel (Mac OS)



Zoom windows to full screen (Mac OS, Solaris, and Linux)

This checkbox configures the behavior of the zoom box in the upper right-hand corner of all editor windows. If this checkbox is enabled and you click the zoom box of an editor window, the window resizes to fill the entire screen. If this checkbox is not enabled, clicking the zoom box of an editor window resizes that window to a standard size.

Use External Editor (Mac OS, Solaris, and Linux)

If this checkbox is enabled, the CodeWarrior IDE sends Open Document ('odoc') AppleEvents to AppleScript-compatible third-party text editors. To use this feature, create a (Helper Apps) folder (if it does not already exist) in the CodeWarrior IDE application folder. Next, place an alias inside the (Helper Apps) folder which points to your third-party editor application. Then, change the name of the alias to External Editor

If the **Use External Editor** checkbox is not enabled, the CodeWarrior editor is the default editor for your text files.

Note that the third-party editor is not used for text editing unless the file you are working with has been added to the currently open Project window.

To learn how to add files to the Project window, refer to [“Adding Files” on page 78](#).

Enable automatic Toolbar help (Mac OS)

This checkbox toggles on and off the Balloon Help for the toolbar icons. When you move the mouse over an icon and leave it there for a second or two, a balloon pops up and tells you more about the command represented by the icon.

Use Script menu (Mac OS, Solaris, and Linux)

When this checkbox is enabled, the Script Menu is displayed in the IDE’s menu bar.

For more information about the Script Menu and scripting the IDE with AppleScript, refer to *Targeting Mac OS*.

Use ToolServer menu (Mac OS)

This checkbox toggles the display of the Tools menu ([Figure 8.8](#)) in the IDE’s menu bar. For more information about using this menu, refer to *Targeting Mac OS*.

Figure 8.8 ToolServer Menu



Use BEdit™ Extensions (Mac OS)

When this checkbox is enabled, an additional icon displays in the CodeWarrior IDE menu bar, as shown in [Figure 8.9](#). This icon is useful for accessing BEdit extensions from within CodeWarrior.

For more information about BEdit, refer to its documentation.

Figure 8.9 BBEdit Extensions Menu Icon



Find Reference using (Mac OS)

This pop-up menu lets you select an online database application to look up references and definitions. The following formats are supported:

- QuickHelp-based documents (supplied with the IDE)
- Symantec THINK Reference database (not part of the CodeWarrior product)
- PalmQuest reference for the Palm connected organizer
- QuickView-based documents, such as the Macintosh Programmer's Toolbox Assistant (MPTA)

CodeWarrior documentation in QuickHelp format is located on the CodeWarrior Reference CD.

To learn more about online reference databases and how to use them, refer to [“Online References” on page 162](#).

Plugin Settings

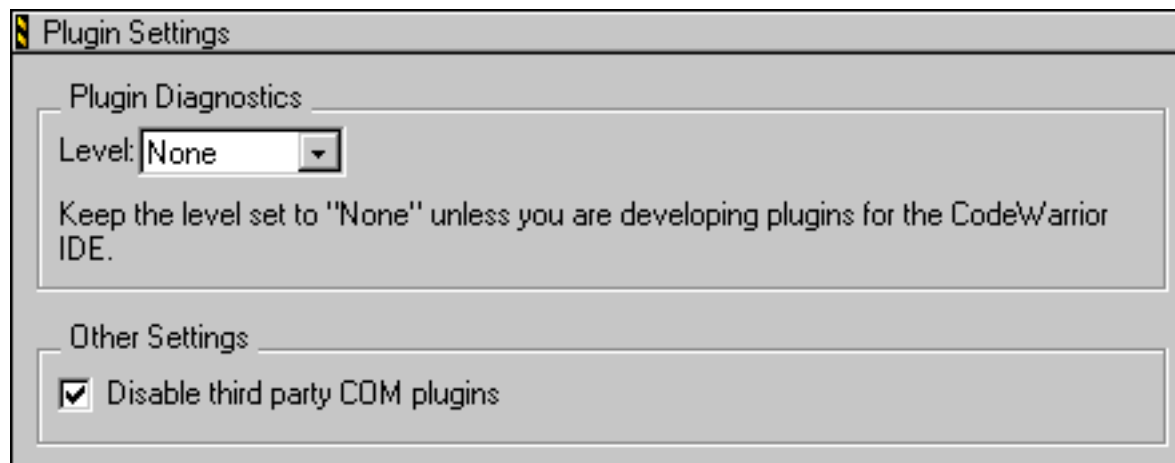
The Plugin Settings panel, shown in [Figure 8.10](#), has options to troubleshoot third-party IDE plug-ins.

To learn how to open the IDE Preferences window and select the Plugin Settings panel, see [“Preferences Guided Tour” on page 253](#).

Plugin Diagnostics

The pop-up menu in this section helps the IDE tell you more information about plug-ins that you are developing for CodeWarrior. This information is useful when you experience problems getting your plug-in to function properly, or you want more information about the properties of installed plug-ins.

Figure 8.10 Plugin Settings preference panel



There are three possible levels of plug-in diagnostics:

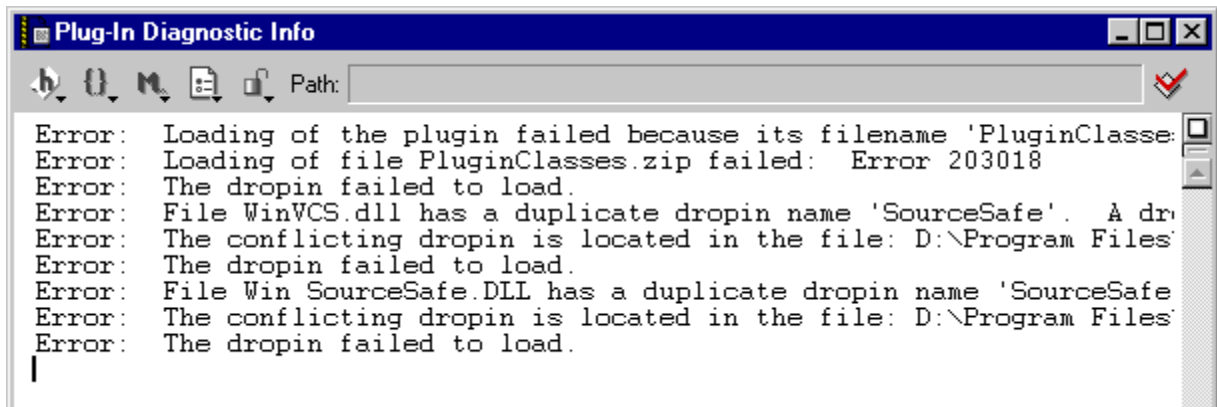
None—This is the default setting. No plug-in diagnostics are activated, and no output is produced.

- **Errors Only**—This setting causes CodeWarrior to report problems that the IDE encounters when loading plug-ins. These problems are displayed in a new text document after the IDE starts up, as shown in [Figure 8.11](#).
- **All Info**—This setting causes CodeWarrior to report detailed information for each plug-in. Problems with loading plug-ins, optional plug-in information, and plug-in properties are reported. This information is displayed in a new text document after the IDE starts up, as shown in [Figure 8.12](#). This text document also includes a complete list of installed plug-ins and their associated preference panels, compilers, and linkers.

In order to use a specific plug-in diagnostic level, select it from the Level pop-up list.

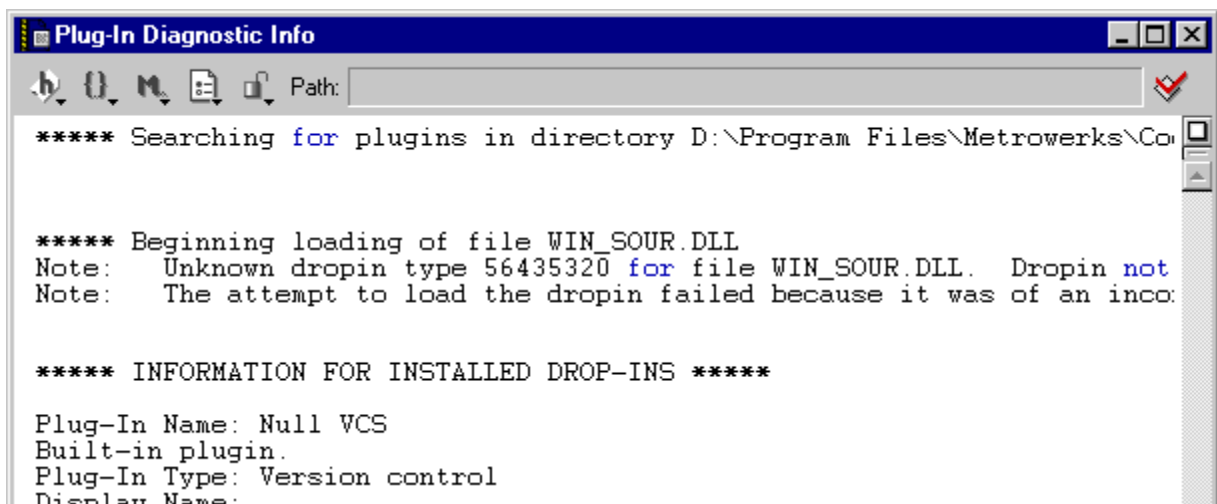
NOTE You must restart CodeWarrior for any changes to the diagnostic level to take effect. When you change a plug-in diagnostic level and save that change, CodeWarrior displays a dialog box reminding you to restart.

Figure 8.11 Plugin Diagnostics - Errors Only text file



You have the option of saving the generated text file. You can also print out the text file so that you can have a convenient error reference when troubleshooting your plug-ins. CodeWarrior provides suggestions in the generated text file for correcting general plug-in errors.

Figure 8.12 Plugin Diagnostics - All Info text file



Disable third party COM plugins

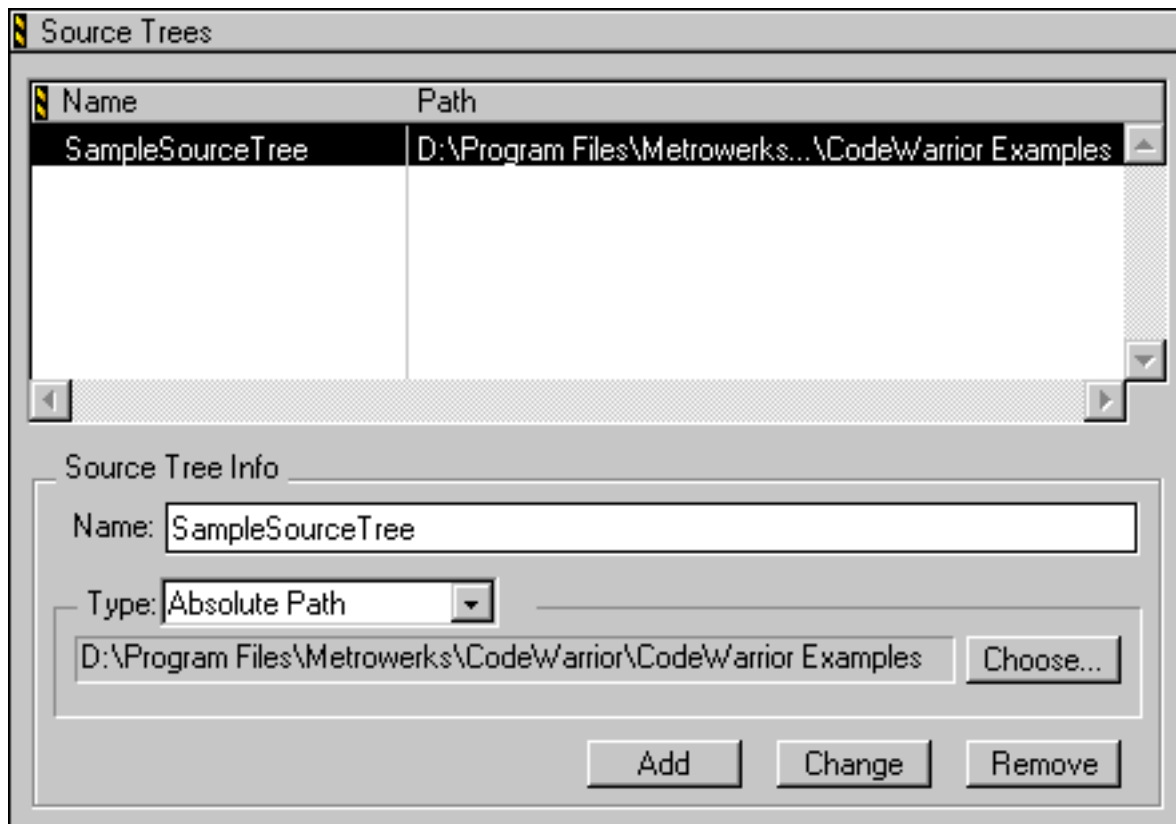
Enable this checkbox to disable all third-party Common Object Model (COM) plug-ins for the IDE. This option is useful for troubleshooting problems with the IDE.

TIP If you experience problems when using the IDE, try disabling third-party COM plug-ins. If the problem does not occur after you disable the plug-ins, then the source of the problem likely involves conflicts between the plug-ins supplied with CodeWarrior and the third-party plug-ins.

Source Trees

The Source Trees preference panel, shown in [Figure 8.13](#), lets you define global source trees (root paths) for use in your projects. You can define your project's access paths and build-target output in terms of source trees. Using this approach, you can easily share projects across various hosts. You need only make minor changes to the source trees' paths to maintain your project's functionality.

Figure 8.13 Source Trees preference panel



To learn how to open the IDE Preferences window and select the Source Trees preference panel, see [“Preferences Guided Tour” on page 253](#).

There is also a target-specific Source Trees settings panel. For more information about this settings panel, see [“Source Trees” on page 345](#). The source trees you define in the IDE Preferences window can be used for all projects. The source trees defined in the Target Settings window can only be used with the current build target in the active project. If you define the same source trees in both panels, the target-specific source trees take precedence over the global source trees.

Source Trees list

This list shows all of the source trees defined for use with the IDE. The list consists of two columns:

- **Name**—This column shows the name of each source tree. When you define access paths in terms of source trees, you use this name in your access path’s definition. For more information, see [“Add” on page 334](#).
- **Path**—This column shows the path to each source tree. You might need to modify the source trees’ paths when you transfer your project to a new host. Refer to [“Change” on page 271](#) for more information about modifying the current path.

Name

This field is part of the **Source Tree Info** section shown in [Figure 8.13](#). Use the **Name** field to enter a name for a new source tree or to change the name of an existing source tree.

Type

This field is part of the **Source Tree Info** section shown in [Figure 8.13](#). Use the **Type** field to choose one of the following types of source trees:

- **Absolute Path**—This type of source tree is based on a file path.
- **Environment Variable**—This type of source tree is based on an existing definition of an environment variable. You cannot create or modify this type of source tree on the Mac OS-hosted IDE.

- **Registry Key**— (Windows) This type of source tree is based on an existing key entry in the registry.

Add

To add a new source tree, first select the appropriate type of source tree from the **Type** pop-up menu. Next, enter a name for the new source tree in the **Name** field.

When you create an **Absolute Path** source tree, the **Choose** button is available. Click this button to select a path using a standard dialog box, as shown in [Figure 8.14](#) (Windows) and [Figure 8.15](#) (Mac OS).

Figure 8.14 Access Path Selection dialog box (Windows)

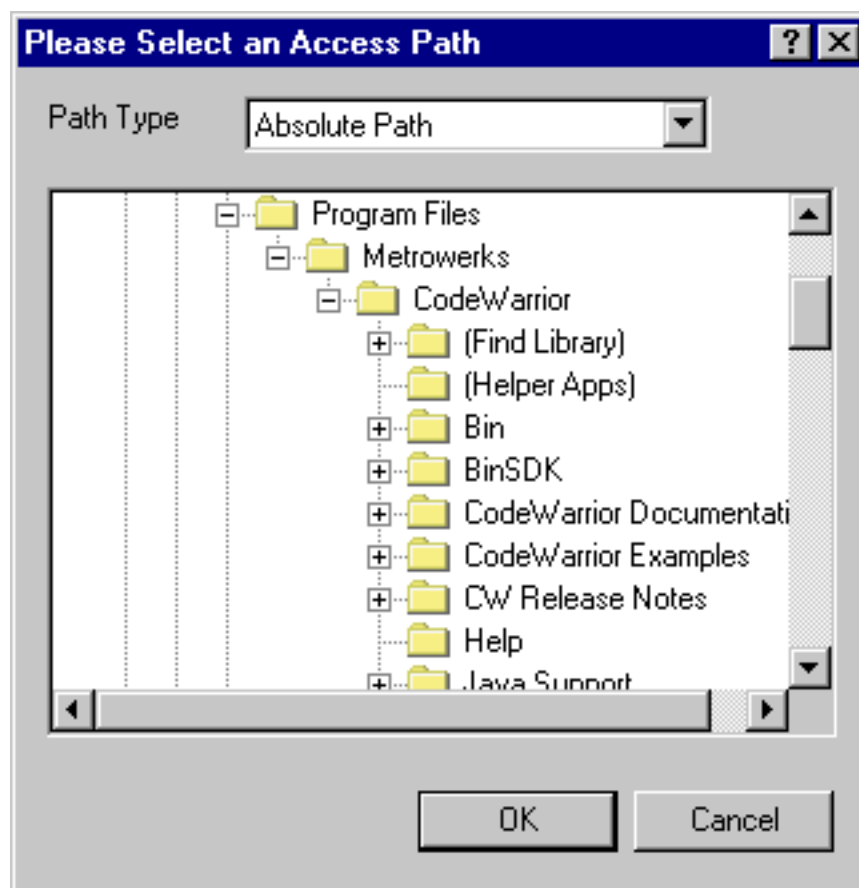
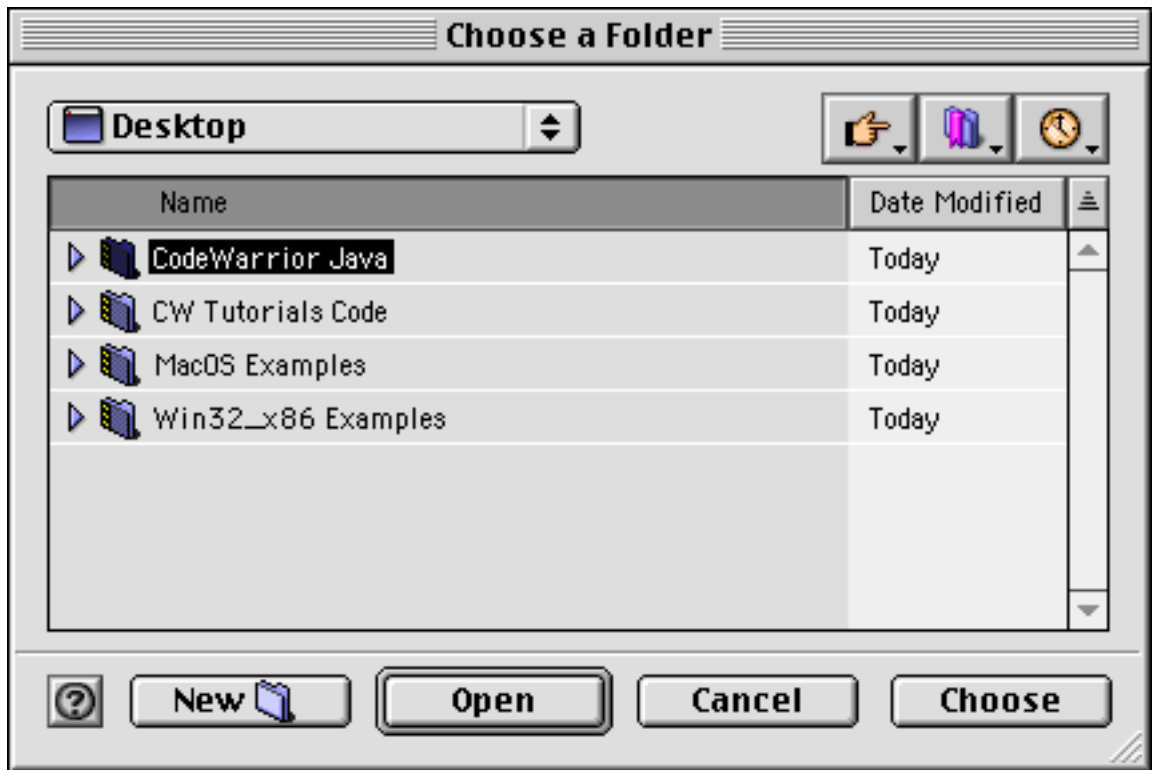


Figure 8.15 Access Path Selection dialog box (Mac OS)



On the Windows-hosted IDE, you can specify how CodeWarrior stores a path by choosing one of these path type options:

- **Absolute Path**—means that the IDE stores the path from the root level of the startup hard drive to the folder whose path you want to add, including all folders in between. You need to update absolute paths if you move the project to another system, rename the hard drive, or rename any of the folders along the path.
- **Compiler Relative**—means that the IDE stores the path from the folder that contains the CodeWarrior IDE to the folder whose path you want to add. You do not need to update compiler-relative paths if you move a project, as long as the hierarchy of the relative path is the same. You cannot create a relative path to a folder on a different hard drive than where your CodeWarrior IDE resides.
- **System Relative**—means that the IDE stores the path from the operating system's base folder to the folder whose path you want to add. You do not need to update system relative paths if

you move a project, as long as the hierarchy of the relative path is the same. You cannot create a relative path to a folder on a different hard drive than where your active operating system's base folder resides.

NOTE Relative paths allow projects to contain two or more files with identical names. However, for large projects, you might notice slower performance when adding relative paths to a project.

When you create an **Environment Variable** or **Registry Key** source tree, the **Type** field changes to the field shown in [Figure 8.16](#). Enter the path to the environment variable or registry key in this field.

Figure 8.16 Creating a registry key (Windows)

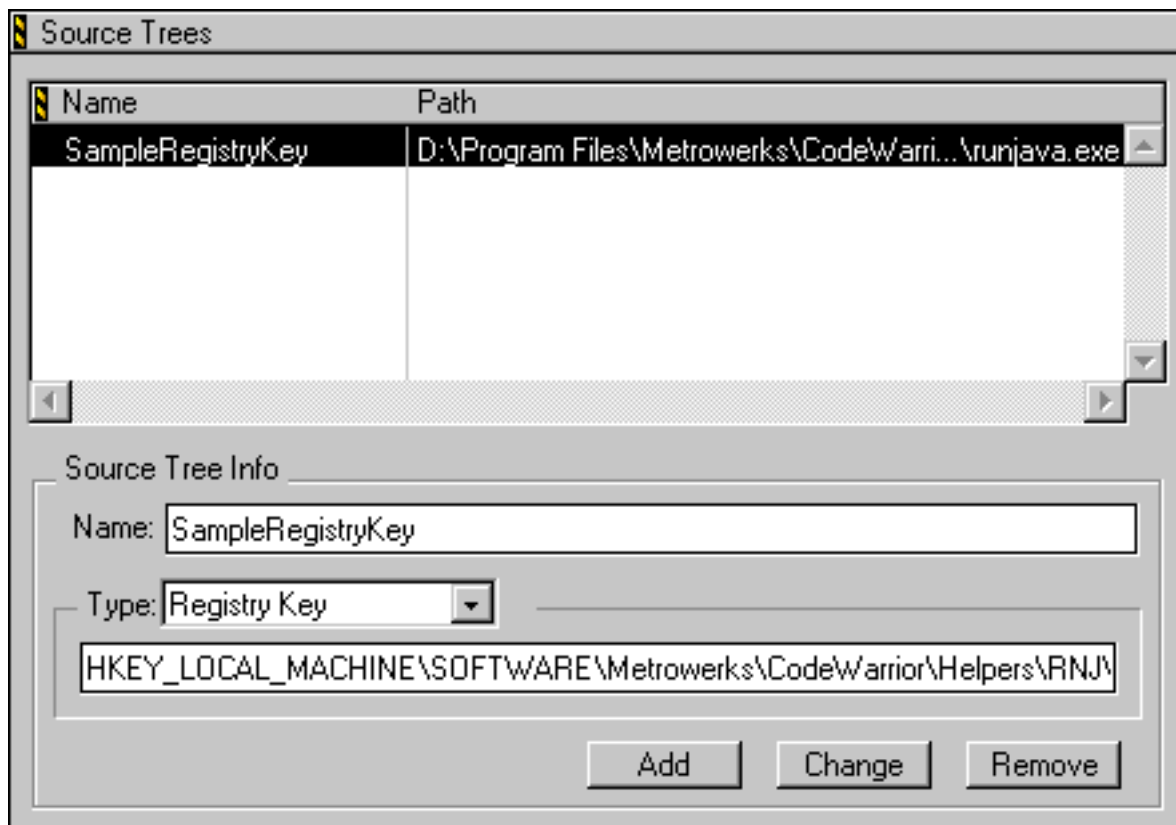
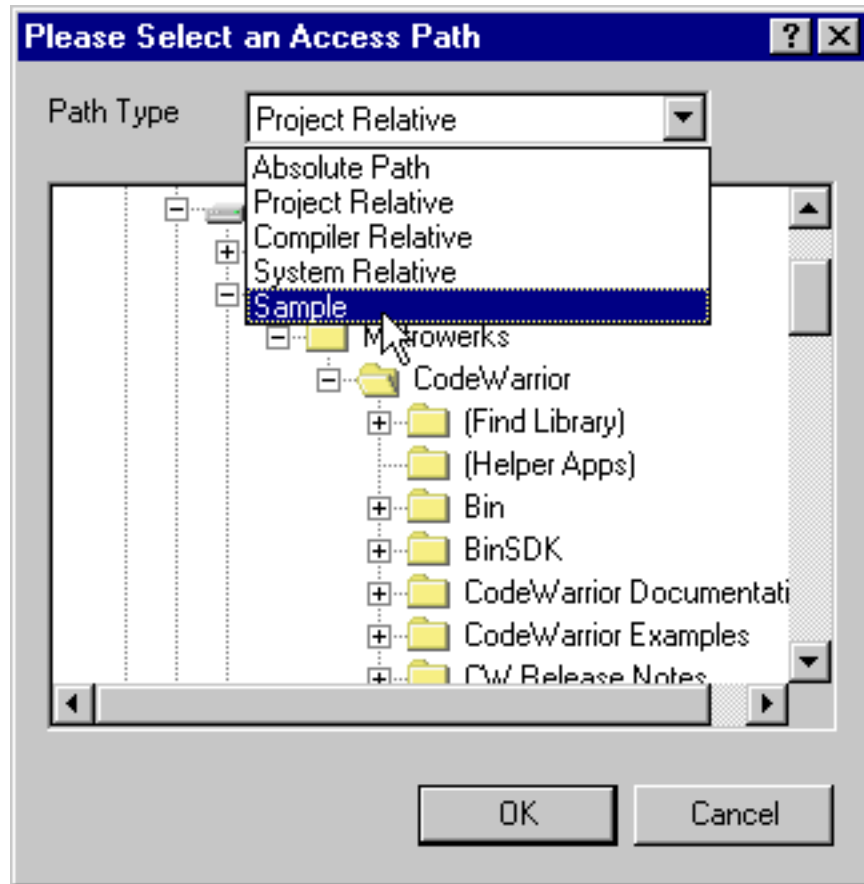


Figure 8.17 Choosing source trees in access path pop-up menus



When you finish adding source trees, click the **Save** button in the IDE Preferences window to save your changes. The source trees are then displayed in the path pop-up menus for your project. For example, if you create a source tree named **Sample**, you can use it to create an access path, as shown in [Figure 8.17](#). You can also define your project's target output in terms of the source trees by using the **Output Directory** field in the **Target Settings** panel. For more information about this settings panel, refer to ["Target Settings" on page 327](#).

Change

To change a source tree, first select it in the [Source Trees list](#). Next, modify the **Name** and **Type** fields as desired. Then click the **Change** button. When you finish modifying the source trees, click the **Save** button in the IDE Preferences window to save your changes.

NOTE After changing a source tree, you might need to modify your project so that you do not refer to the source tree in its original form. Therefore, the IDE displays a message reminding you to update your project.

If your project's files or libraries are not in the source trees, the IDE will not find them when compiling, linking, or running your project. Therefore, after you change an existing source tree, the IDE displays a message reminding you to update your project.

Remove

To remove a source tree, first select it in the Source Trees panel. Then, click the **Remove** button, and the source tree is removed. When you finish removing source trees, click the **Save** button in the IDE Preferences window to save your changes.

NOTE After removing a source tree, you might need to modify your project so that you do not refer to the removed source tree. Therefore, the IDE displays a message reminding you to update your project.

Editor Preferences

In this section we discuss preference panels that control the CodeWarrior editor's features. The Editor panels include:

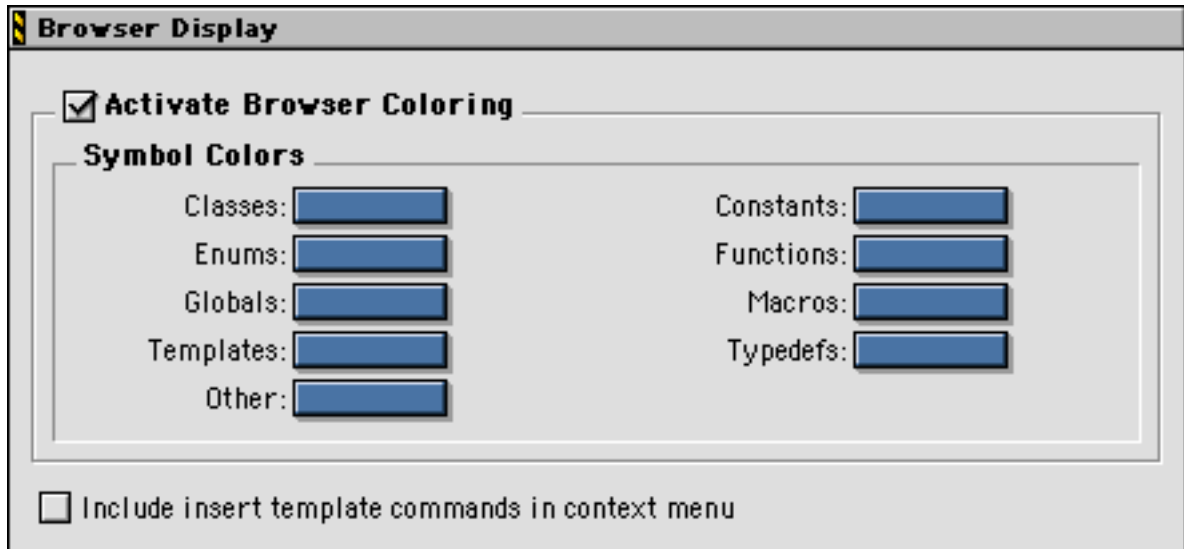
- [Browser Display](#)
- [Editor Settings](#)
- [Font & Tabs](#)
- [Syntax Coloring](#)

Browser Display

The Browser Display preferences are shown in [Figure 8.18](#).

To learn how to open the IDE Preferences window and select the Browser Display panel, see [“Preferences Guided Tour” on page 253](#).

Figure 8.18 Browser Display options



The Browser can export its lists of symbols and their types to the CodeWarrior editor. This enables the editor to use different colors for displaying various types of symbols. Enable the **Activate Browser Coloring** checkbox to use this feature. When active, the color choice for each symbol type is displayed in the Editor window and the Browser window. Click on a color sample to change its color.

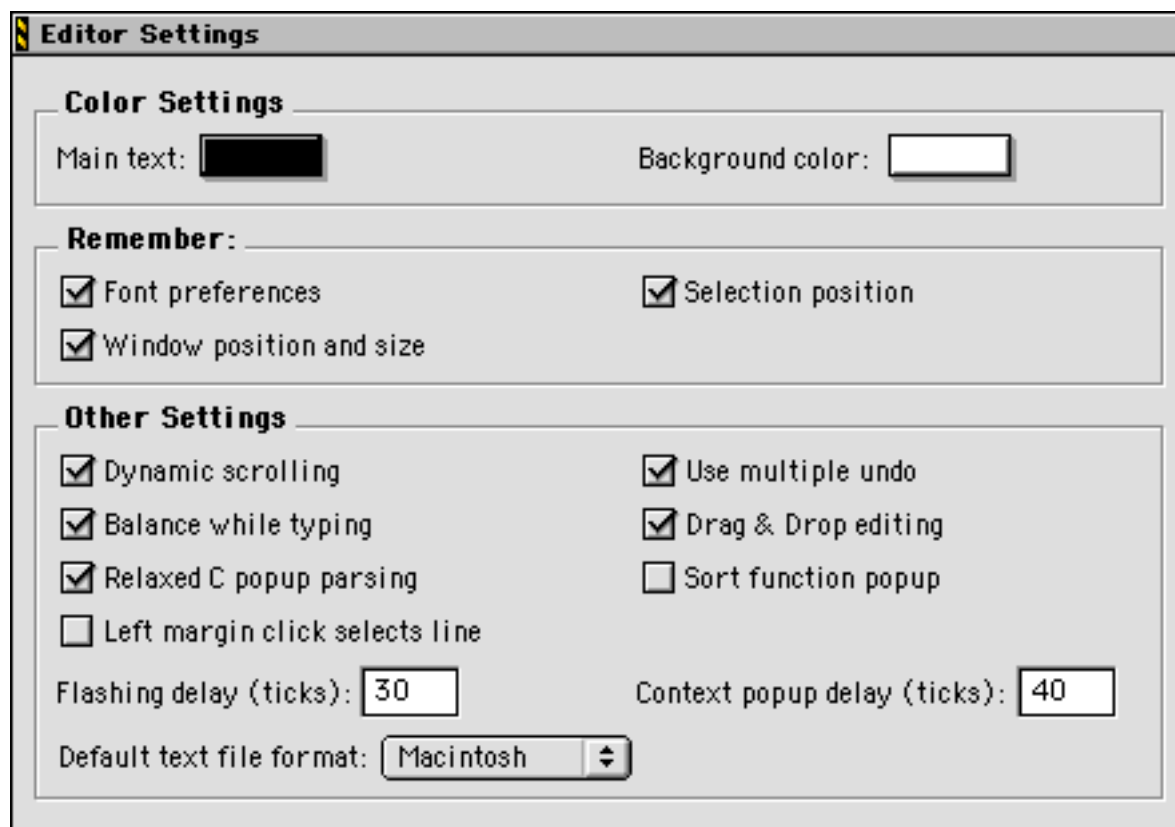
Enable the **Include insert template commands in context menu** checkbox to use the context pop-up menus to insert templates into your source code. By default, this option is disabled. When this option is enabled, there is an additional **Insert Template** command in your context pop-up menus, as shown in [Figure 7.4 on page 214](#).

Editor Settings

This section tells you how to configure the CodeWarrior editor's behavior to make your text-editing chores easier. The Editor Settings panel is shown in [Figure 8.19](#).

To learn how to open the IDE Preferences window and select the Editor Settings panel, see ["Preferences Guided Tour" on page 253](#).

Figure 8.19 Editor Settings preference panel



This preference panel has three groups of options: Color Settings, Remember, and Other Settings. The [Color Settings](#) control the main text color (non-syntax) and the background color in the Editor and Browser windows. The [Remember](#) options determine the Editor window settings that are saved from one programming session to the next. [Other Settings](#) (such as [Dynamic scrolling](#) and [Balance while typing](#)) control how the editor works.

Color Settings

The following options control the color settings for the editor.

Main text This option configures the color of any text not colored by the [Browser Display](#), [Syntax Coloring](#), or [Custom Keywords](#) color sets. Click the color sample to change the color using the operating system's standard color selection window.

Background color Click the color sample to change the background color of the Editor and Browser windows using the operating system's standard color selection window.

Remember

These options control which settings the editor remembers between programming sessions.

Font preferences You can configure the font information for an individual file if you enable this checkbox. Otherwise, all files inherit the default font settings from the CodeWarrior IDE.

Window position and size Enabling this checkbox saves each window's position and size so that files open in the same location on the screen each time. This feature requires that your editor files be writable. To learn more about writable files, refer to ["Common VCS Operations" on page 613](#).

Selection position Enabling this checkbox tells the CodeWarrior IDE to remember what part of a file's text was visible, the location of the insertion point, and any selections that were made. Disable this checkbox if you always want the editor to go to the top of the file when it opens a window. This feature requires that your files be writable. To learn more about writable files, refer to ["Common VCS Operations" on page 613](#).

Other Settings

These options control other behaviors that the editor remembers between programming sessions.

Dynamic scrolling (Mac OS, Solaris, and Linux) When this checkbox is enabled, the text in an Editor window visually scrolls while you drag the scroll box in the scrollbar. To temporarily disable this option, hold down the Option key while dragging the scroll box.

Balance while typing When this checkbox is enabled, the IDE checks for balanced parentheses, brackets, and braces as you type. Each time you type a right parenthesis, bracket, or brace, the editor attempts to locate the matching left counterpart. If the counterpart is found, the editor brings it into view, highlights it for a specified length of time called the [Flashing delay](#), and returns to where you were typing. If the counterpart is not found, the editor beeps. By default, the **Balance while typing** checkbox is enabled. To learn

more about the Flashing delay, refer to [“Flashing delay” on page 277](#).

TIP If you want to check for balanced punctuation without highlighting it, set the Flashing delay to 0.

Relaxed C popup parsing Enable this checkbox if you use style-coding conventions as described in “Reference Manual,” of *The C Programming Language, Second Edition* (Prentice Hall) by Kernighan and Ritchie (K&R). Enabling the checkbox allows the CodeWarrior IDE to recognize the K&R style coding conventions in your source code and properly display function names in the Routine pop-up menu. Disable the checkbox if you use non-standard macros that can interfere with K&R-styled code.

NOTE Some macro functions are not recognized when the **Relaxed C popup parsing** checkbox is enabled. If you encounter problems while viewing routine names, disable this checkbox and try again.

Left margin click selects line When this checkbox is enabled, moving the cursor to the left edge of an editor window changes the cursor’s icon to a right-pointing arrow. Clicking along the left margin of the window with the right-pointing arrow selects the line next to the cursor. Clicking and dragging the right-pointing arrow selects more than one line. When this checkbox is disabled, the cursor’s icon is a left-pointing arrow. The left-pointing arrow cannot select an entire line with a click.

Use multiple undo When this checkbox is enabled, you can undo and redo multiple actions. When this checkbox is disabled, you can only undo or redo the most recent action that you performed. See [“Redo, Multiple Undo, and Multiple Redo” on page 624](#) for more information.

Drag & Drop editing This checkbox enables Drag and Drop support in the editor. To learn more about this feature, refer to [“Moving Text \(Drag and Drop\)” on page 154](#).

Sort function popup Enable this checkbox if you want to alphabetically sort the items listed in the Editor window’s [Routine Pop-Up Menu](#). To learn more about this feature, refer to [“Routine Pop-Up Menu” on page 140](#).

Flashing delay The Flashing delay is the amount of time the CodeWarrior editor displays and highlights an item. It is measured in 60ths of a second. This option is for balancing punctuation. To learn more about this setting, refer to [“Balancing Punctuation” on page 154](#) and [“Balance while typing” on page 275](#).

WARNING! If you enter 0 for the Flashing delay, you disable flashing entirely.

Context popup delay (Mac OS, Solaris, and Linux) This option determines the minimum length of time to hold down the mouse button before the [Browser Contextual Menu](#) displays. The range of acceptable values is 0 to 240. Each interval represents 1/60th of a second (16.67 milliseconds). To learn more about this browser feature, refer to [“Browser Contextual Menu” on page 213](#) and [“Using the Contextual Menu” on page 246](#).

WARNING! If you enter 0 for the Context pop-up delay, you disable the pop-up menu entirely.

Default text file format This pop-up menu sets the end-of-line conventions that the CodeWarrior IDE uses to create new files. You can choose from three formats: **Macintosh**, **DOS**, and **UNIX**. To learn about saving text files in different text formats, see [“Options Pop-Up Menu” on page 142](#).

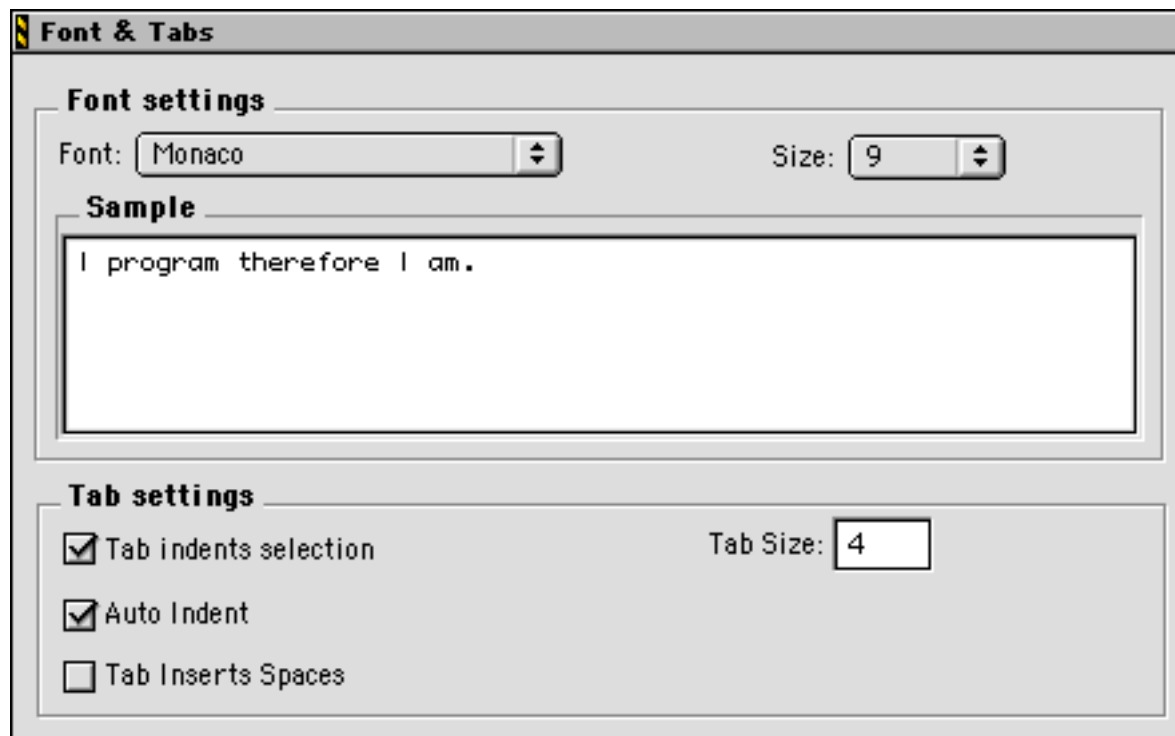
Font & Tabs

The Font & Tabs preference panel is shown in [Figure 8.20](#). This panel sets the font and tab information for the active Editor window. If no Editor window is open, the panel’s settings apply to the CodeWarrior IDE defaults.

To learn how to open the IDE Preferences window and select the Font & Tabs panel, see [“Preferences Guided Tour” on page 253](#).

To change the font and tab settings for a file, open an Editor window for that file. Then, open the IDE Preferences window, select the Font & Tabs preference panel, and make your changes. Each time you open that file in the future, the editor remembers the font and tab preferences that you set.

Figure 8.20 Font & Tabs preference panel



As long as you have write permissions on the file (that is, the file is not Read-Only), the IDE remembers your changes. To learn more about writable files, refer to [“Common VCS Operations” on page 613](#).

Font

Use this pop-up menu to choose your desired text font.

Size

Use this pop-up menu to choose your desired font size.

Sample

This field provides sample text that reflects the current settings in the [Font](#) and [Size](#) fields.

Tab Settings

These options control the use of tabs in the CodeWarrior IDE.

Tab indents selection When this checkbox is enabled, you can select some lines of text and press the Tab key to insert a tab in front of each selected line. The inserted tabs do not replace the existing text. When the checkbox is disabled, selected lines are completely replaced with tabs. The checkbox is enabled by default.

Auto Indent Enable this checkbox if you want the editor to automatically create a new line with the same indentation as the previous line.

Tab Inserts Spaces Enable this checkbox to insert spaces in the text when you press the Tab key. You can specify the number of inserted spaces in the [Tab Size](#) field.

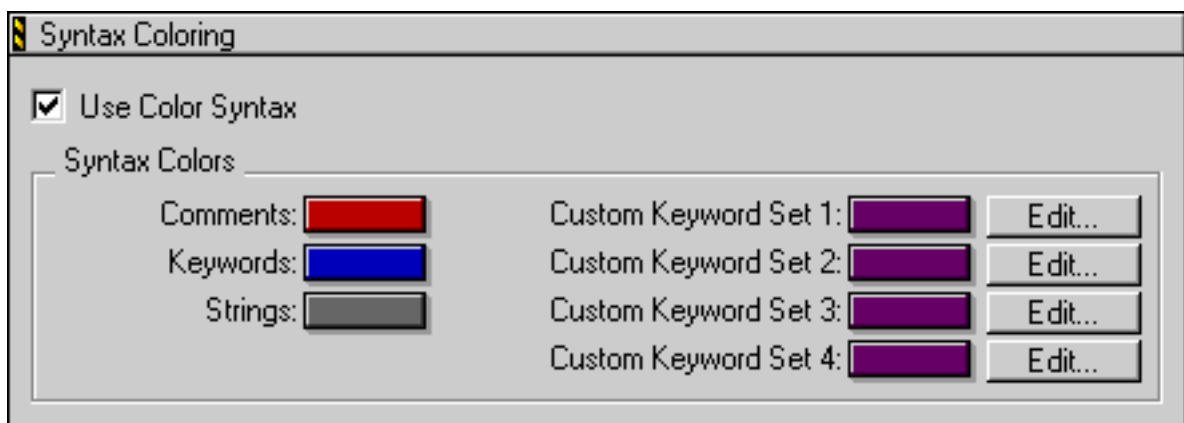
Tab Size This field represents the width of a tab, measured in spaces, when you press the Tab key.

Syntax Coloring

The Syntax Coloring preference panel, shown in [Figure 8.21](#), includes options for setting the color of comments, keywords, strings, and custom keyword sets. Enable the **Use Color Syntax** checkbox to display your text files according to the colors defined in this panel.

To learn how to open the IDE Preferences window and select the Syntax Coloring panel, see [“Preferences Guided Tour” on page 253](#).

Figure 8.21 Syntax Coloring preference panel



You can assign a particular color to four distinct sets of words. Each of these custom keyword sets can contain routine names, type names, or anything else you want to highlight in an Editor window. [Table 8.1](#) describes the text elements for which you can set a color.

Table 8.1 **Syntax coloring elements**

Element	Description
Comments	Code comments. In Java, C, or C++, comments are placed between <code>/*</code> and <code>*/</code> or from <code>//</code> to the end of the line. In Pascal, comments are place between <code>{</code> and <code>}</code> or between <code>(*</code> and <code>*)</code> .
Keywords	The programming language’s keywords. This element does not include macros, types, variables defined by system interface files, or variables that you define.
Custom Keywords	Any keyword listed in the Custom Keyword List. This list is useful for macros, types, and other names that you want to highlight.
Strings	Anything that is not a comment, keyword, or custom keyword, such as literal values, variable names, routine names, and type names.

Changing syntax highlighting colors

The CodeWarrior IDE can use different colors for each type of text. To change these colors, click the color sample beside the name. The CodeWarrior IDE displays a standard dialog box for you to select a new color. The next time you view a text file, the CodeWarrior IDE uses the new color.

Controlling syntax coloring within a window

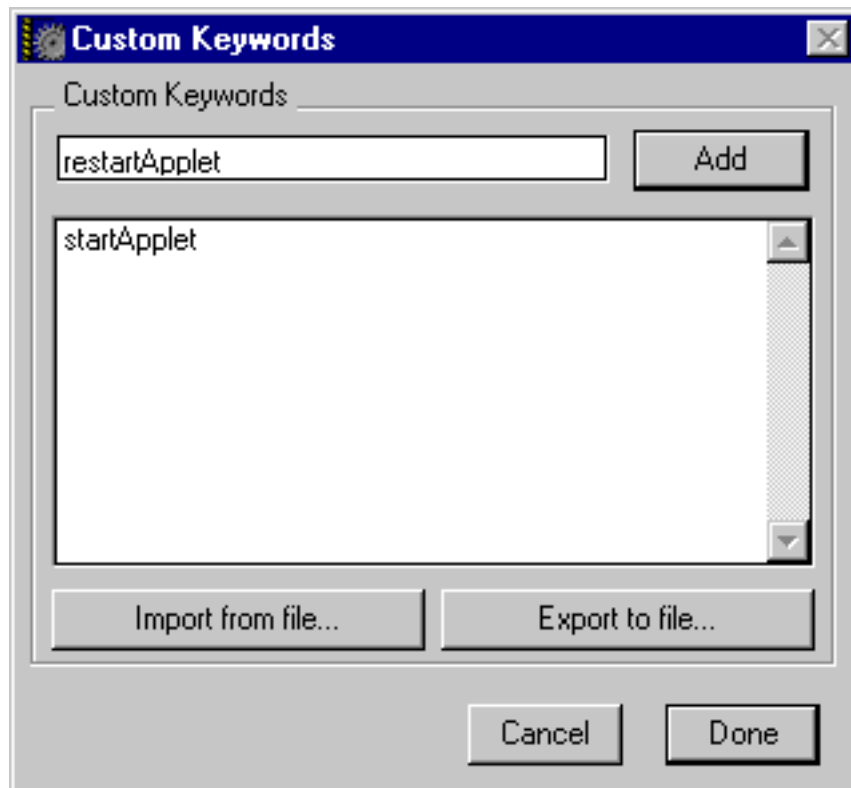
Use the **Syntax Coloring** command in the [Options Pop-Up Menu](#) to enable or disable syntax coloring as you view a particular file. For more information, see [“Options Pop-Up Menu” on page 142](#).

Using color for custom keywords

Use the Custom Keywords dialog box to choose additional words to display in color. These words can be macros, types, or other names that you want to highlight. The custom keywords are global to the CodeWarrior IDE and apply to every project.

Click the **Edit** button to the right of the custom keyword set you want to modify. The CodeWarrior IDE displays the Custom Keywords dialog box, shown in [Figure 8.22](#).

Figure 8.22 Custom Keywords dialog box



Type the custom keyword in the field at the top of the Custom Keywords dialog box, then click **Add**. The IDE adds the custom keyword to the Custom Keywords List. You can add as many custom keywords as desired. However, due to the way the IDE handles these words, you might not be able to add custom keywords to a large Custom Keywords List. If the IDE is unable to add a custom keyword to the list, it displays an error message.

To delete a custom keyword, select it and press Backspace/Delete. The CodeWarrior IDE removes the custom keyword from the Custom Keywords List.

Once you finish entering custom keywords, click **Done**. The dialog box disappears. The next time you view a source file, all of your custom keywords are highlighted in the selected color.

TIP You can also set target-specific colors for custom keywords. To learn more about this, see [“Custom Keywords” on page 356](#).

Importing or exporting custom keywords

You can import existing custom keyword sets for use in the IDE, or you can export your custom keyword sets for use in another host.

To import a custom keyword set, follow these steps:

1. **Click the Edit button next to the desired custom keyword set in the [Syntax Coloring](#) preference panel.**

The IDE displays the Custom Keywords dialog box.

2. **Click Import from file.**

A standard Open dialog box displays.

3. **Use the dialog box controls to select the custom keyword set you wish to import.**

4. **Click Open.**

The IDE adds the custom keywords from the selected set to the Custom Keywords list.

5. **Click Done.**

The imported custom keywords become part of your custom keyword set.

To export a custom keyword set, follow these steps:

1. **Click the Edit button next to the desired custom keyword set in the [Syntax Coloring](#) preference panel.**

The IDE displays the Custom Keywords dialog box.

2. Click Export to file.

A standard Save dialog box displays.

3. Use the dialog box controls to select a location to save the exported custom keyword set.

If you plan to use the exported custom keyword set on a different host, you should add a “.txt” extension (without the quotes) to the set’s name. The Windows-hosted IDE requires this extension in order to identify the custom keyword set.

4. Click Save.

The IDE exports the custom keyword set and saves it to your hard disk.

5. Click Done.

You return to the Syntax Coloring preference panel.

Debugger Preferences

In this section we discuss preference panels that control debugger features. The debugger panels include:

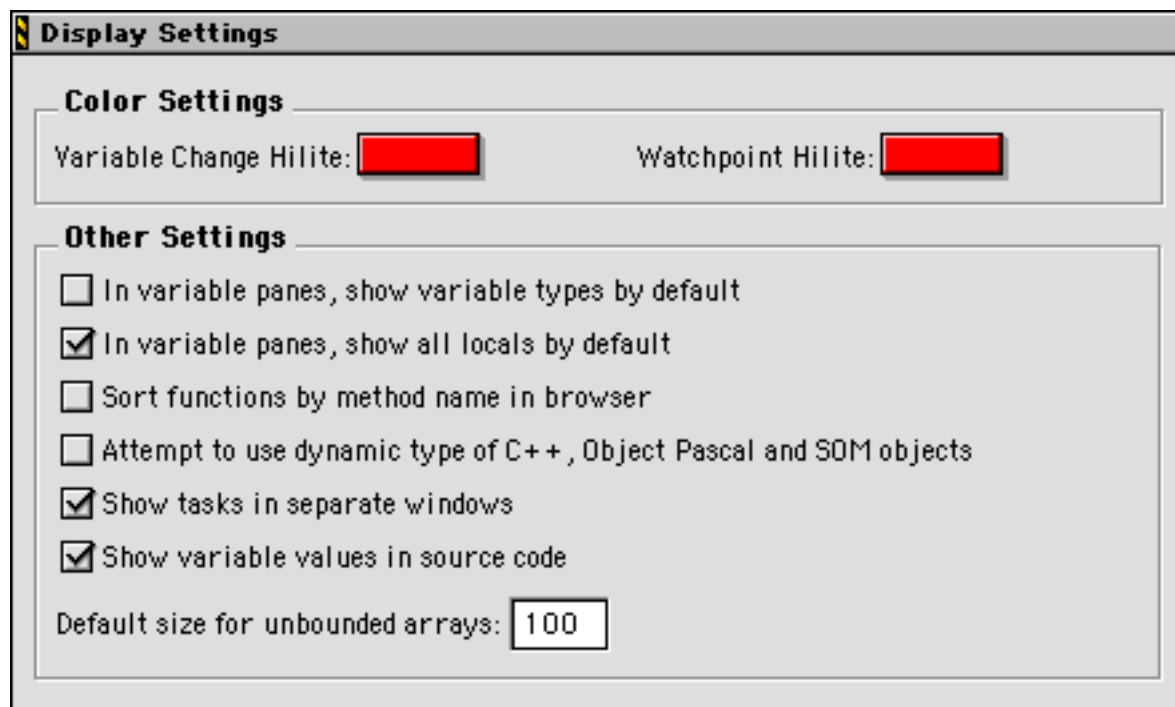
- [Display Settings](#)
- [Windowing](#)
- [Global Settings](#)
- [Java Debugging](#)
- [Java Settings \(Windows\)](#)
- [MetroNub Settings \(Mac OS\)](#)
- [x86 Settings \(Mac OS\)](#)

Display Settings

You can configure the CodeWarrior debugger to accommodate the way you work. The Display Settings panel is shown in [Figure 8.23](#).

To learn how to open the IDE Preferences window and select the Display Settings panel, see [“Preferences Guided Tour” on page 253](#).

Figure 8.23 Debugger Display Settings preference panel



Variable Change Hilite

This option lets you set the color that the debugger uses to identify a changed variable. To change the current color, click the color sample beside the name. The CodeWarrior IDE displays a standard dialog box for you to select a new color. The next time you view debugging windows, the CodeWarrior IDE uses the new color.

Watchpoint Hilite

This option lets you set the color that the debugger uses to identify a watchpoint. To change the current color, click the color sample beside the name. The CodeWarrior IDE displays a standard dialog box for you to select a new color. The next time you view debugging windows, the CodeWarrior IDE uses the new color.

In variable panes, show variable types by default

Enable this checkbox to show variable types when you open a new Variable window.

In variable panes, show all locals by default

Enable this checkbox to show all local variables when you open a new Variable window. When this option is enabled, the Variables pane listing is always **Variables:All**. When this option is disabled, the Variables pane listing can be either **Variables:All** or **Variables:Auto**, depending on the previous state. See [“Variables pane” on page 404](#) for more information about these states.

Sort functions by method name in browser

Changes the way C++, Object Pascal, and Java functions are sorted in the Browser window. When this checkbox is disabled, function names of the form `className : : methodName` are sorted alphabetically by class name first, then by method name. Enabling this checkbox causes the functions to be alphabetized directly by method name. Since most C++, Object Pascal, and Java source-code files tend to contain methods all of the same class, this preference makes it easier to select methods by typing from the keyboard.

Attempt to use dynamic type of C++, Object Pascal and SOM objects

Enabling this checkbox displays the runtime type of C++ or Object Pascal objects. Disabling this checkbox displays an object’s static type only.

Show tasks in separate windows

This checkbox lets you toggle between two ways of displaying tasks. When the checkbox is enabled, double-clicking a task in the [Processes Window](#) brings up a separate [Stack Crawl Window](#) to display the code. When the checkbox is disabled, the **Task** pop-up menu is displayed at the top of the Stack Crawl Window. Use this menu to choose a task to display in the Stack Crawl Window.

NOTE The effect of this option does not occur immediately. The setting that is active for the start of a debugging session stays active for the duration of that session. If you change this setting in the middle of a debugging session, you must stop debugging and then restart debugging to make the new preference take effect.

Show variable values in source code

Enable this option to automatically display variable values in your source code. When disabled, the variable values are not displayed.

Default size for unbounded arrays

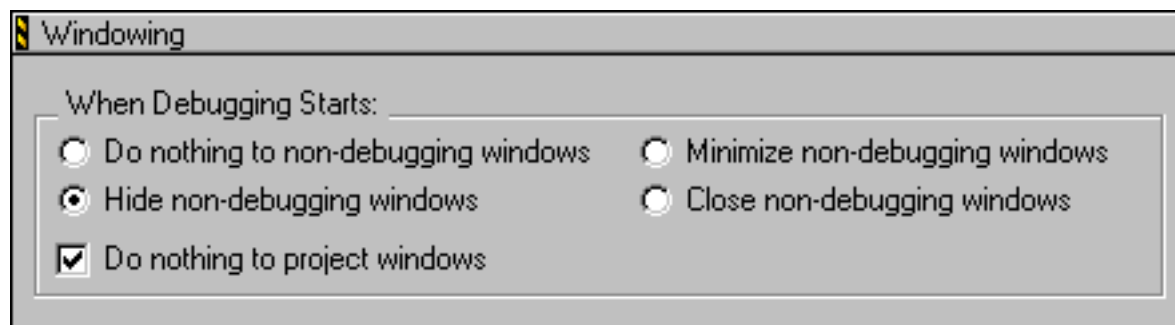
Use this field to specifies the array size to use when no size information is available.

Windowing

The Windowing preference panel, shown in [Figure 8.24](#) (Windows) and [Figure 8.25](#) (Mac OS), allows you to configure window behavior when you start a debugging session.

To learn how to open the IDE Preferences window and select the Windowing panel, see [“Preferences Guided Tour” on page 253](#).

Figure 8.24 Windowing preference panel (Windows)



When Debugging Starts

The following settings determine what the IDE does with non-debugging windows when a debugging session begins.

Do nothing to non-debugging windows Enable this option so that non-debugging windows are not closed or hidden when debugging begins. Since projects are used to initiate debugging, this setting is useful in situations where multiple build targets or multiple projects are being debugged.

Hide non-debugging windows Enable this option to hide, but not close, non-debugging windows. Use the [Window Menu](#) to access the non-debugging windows when they have been hidden.

Double-clicking files in the Project window and performing symbol lookups also shows the hidden windows. At the end of the debugging session, windows that were hidden are made visible.

Minimize non-debugging windows (Windows) Enable this option to minimize non-debugging windows. Use the [Window Menu](#) to access the non-debugging windows when they have been minimized. Double-clicking files in the Project window and performing symbol lookups also expands the minimized windows. At the end of the debugging session, windows that were minimized are expanded. Note that this option is available only when CodeWarrior is in MDI mode. For more information, see [“IDE Extras” on page 259](#) and [“Use Multiple Document Interface \(Windows\)” on page 260](#).

Collapse non-debugging windows (Mac OS) Enable this option to collapse all non-debugging windows. Use the [Window Menu](#) to access the non-debugging windows when they have been collapsed. Double-clicking files in the Project window and performing symbol lookups also expands the collapsed windows. At the end of the debugging session, windows that were collapsed are expanded.

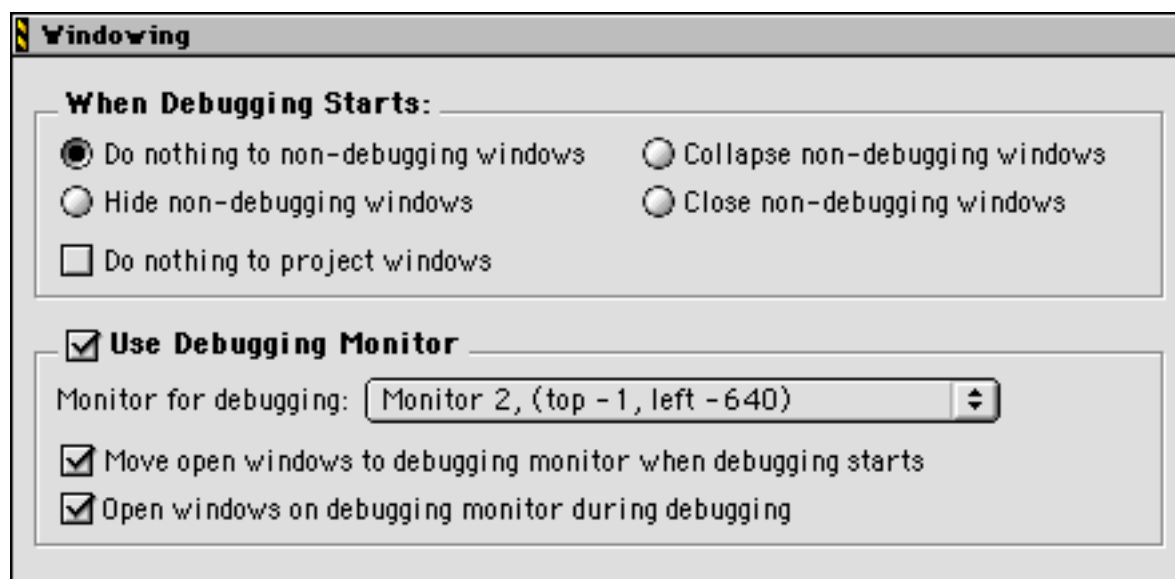
Close non-debugging windows Enable this option to close all non-debugging windows, except for the Project window of the project being debugged. At the end of the debugging session, windows that were closed are re-opened.

Multi-Monitor Debugging (Mac OS)

The following options, shown in [Figure 8.25](#), appear only when you have more than one monitor connected to your computer. These options help you manage debugging windows across your multiple-monitor setup. Enable the **Use Debugging Monitor** checkbox to use CodeWarrior’s multiple-monitor debugging features.

Monitor for debugging Use this pop-up menu to select the monitor you wish to use during debugging sessions. The coordinates in parentheses identify the selected monitor in QuickDraw space.

Figure 8.25 Windowing preference panel (Mac OS)



Move open windows to debugging monitor when debugging starts When you enable this option, all open windows are moved to the selected debugging monitor when you begin a debugging session. At the end of the debugging session, windows that were moved are restored to their original positions.

Open windows on debugging monitor during debugging

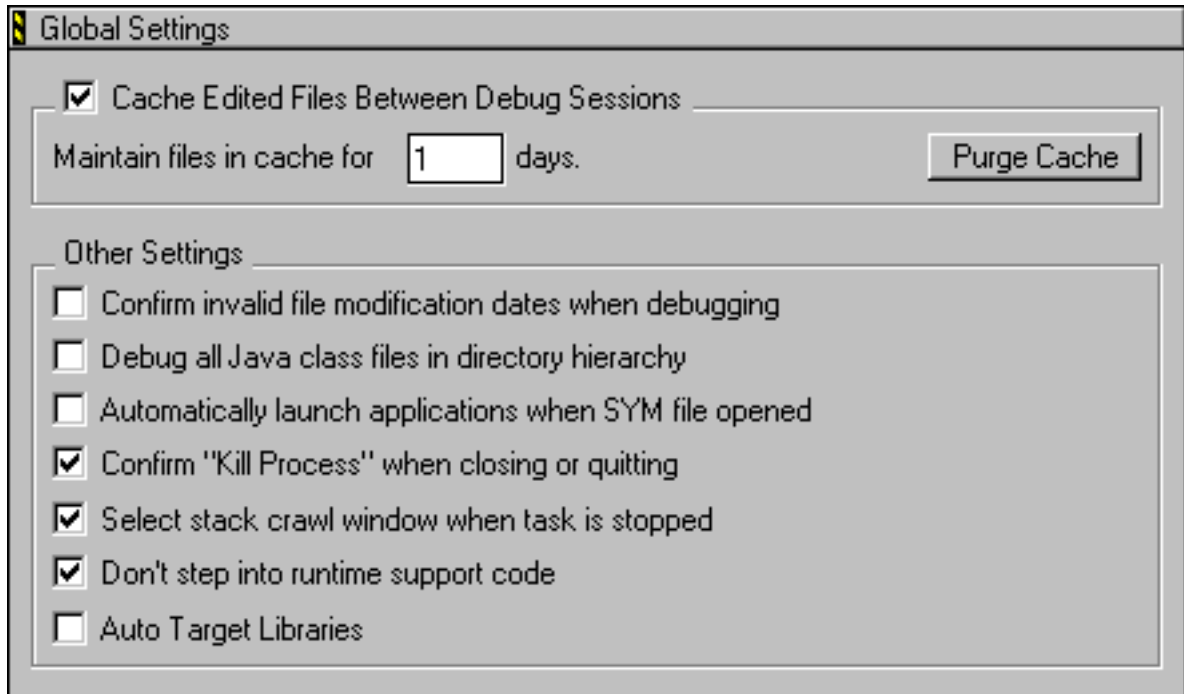
When you enable this option, any window that opens during a debugging session is displayed on the selected debugging monitor. The IDE does not save a window's position on the debugging monitor if that window is closed during the debugging session. This behavior prevents window positions from gravitating to the debugging monitor.

Global Settings

This section describes how to configure the debugger's behavior to make your debugging chores easier. The Global Settings preference panel is shown in [Figure 8.26](#).

To learn how to open the IDE Preferences window and select the Global Settings panel, see [“Preferences Guided Tour” on page 253](#).

Figure 8.26 Global Settings preference panel



Cache Edited Files Between Debug Sessions

Enable this checkbox to maintain a cache of edited files between debugging sessions. When enabled, you can specify the number of days to store the cache in the **Maintain files in cache for** field. You can also reset the file caches by clicking the **Purge Cache** button. When the checkbox is disabled, the IDE does not maintain a cache.

Confirm invalid file modification dates when debugging

The debugger keeps track of the modification dates of source files from which a symbolics file is created. When the modification dates do not match, the debugger can display an alert box warning you of possible discrepancies between the object code and the source code. When this checkbox is enabled, the debugger displays the warning.

Debug all Java class files in directory hierarchy

This checkbox applies when you want to debug a Java program, but you do not have a project file open.

When this checkbox is enabled and you begin a Java debugging session, the debugger searches for additional symbolics files to open. This search occurs in the same folder as the opened class file and includes all of the subfolders. When the checkbox is disabled, only the opened class file is debugged.

Automatically launch applications when SYM file opened

Enable this checkbox to automatically launch a target program when its symbolics file is opened and set an implicit breakpoint at the program's main entry point. Disabling the checkbox lets you open a symbolics file without launching the program, so that you can examine object code that executes before the main routine, such as C++ static constructors. You can also avoid launching the target program by holding down the Alt/Option key when opening a symbolics file.

Confirm “Kill Process” when closing or quitting

Enabling this checkbox causes the IDE to prompt you for confirmation before aborting a process when a target program is killed.

Select stack crawl window when task is stopped

Enabling this checkbox automatically brings the Stack Crawl window to the front when a task is stopped. If the checkbox is disabled, the Stack Crawl window remains in its previous position. This option is useful if you have several Variable windows open and want to see the variables change as you step through your code. You can control the Stack Crawl window even if it is not the currently active window.

Don't step into runtime support code

Enabling this checkbox causes the IDE to execute constructor code for C++ static objects normally, without displaying it in the Stack Crawl window.

Auto Target Libraries

This checkbox applies when you use the debugger to debug a file that is not a project file, such as when you attach a running process. For more information about attaching processes, see [“Processes window toolbar” on page 418](#)

Enable the **Auto Target Libraries** checkbox to let the IDE attempt to debug dynamically linked libraries (DLLs) that are loaded by the target application. The IDE attempts to automatically debug the loaded DLLs for which symbolics information is available.

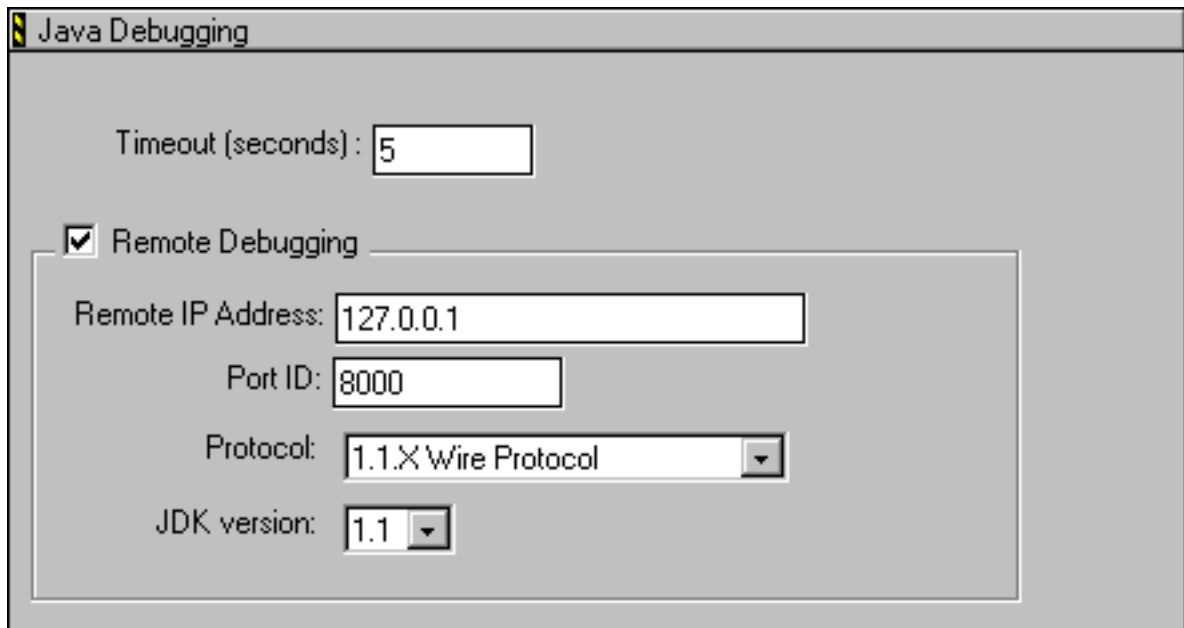
NOTE You might notice slower performance when you enable the **Auto Target Libraries** checkbox. If you encounter this problem, try disabling the checkbox to improve the IDE's performance.

Java Debugging

The Java Debugging preference panel, shown in [Figure 8.27](#), lets you set up your computer to connect to a virtual machine for Java debugging sessions. For more information about Java debugging, refer to *Targeting Java*.

To learn how to open the IDE Preferences window and select the Java Debugging panel, see [“Preferences Guided Tour” on page 253](#).

Figure 8.27 Java Debugging preference panel



Timeout

Enter in this field the number of seconds the CodeWarrior debugger should wait for a virtual machine to attach to it.

Remote Debugging

Enable this checkbox if you want to enable remote debugging of Java projects from your computer.

Remote IP Address

Use this field if you have a network connection to the virtual machine you plan to use for debugging. Enter in this field the Internet Protocol (IP) address of the computer on which the virtual machine is running. This field is enabled only when the [Remote Debugging](#) checkbox is enabled.

Port ID

Use this field if you have a network connection to the virtual machine you plan to use for debugging. Enter in this field the port ID number to be used for the connection between the virtual machine and the CodeWarrior debugger. This field is enabled only when the [Remote Debugging](#) checkbox is enabled.

Protocol

Choose from this pop-up menu the desired protocol for transmissions between your computer and the remote virtual machine during the debugging session. This pop-up menu is enabled only when the [Remote Debugging](#) checkbox is enabled.

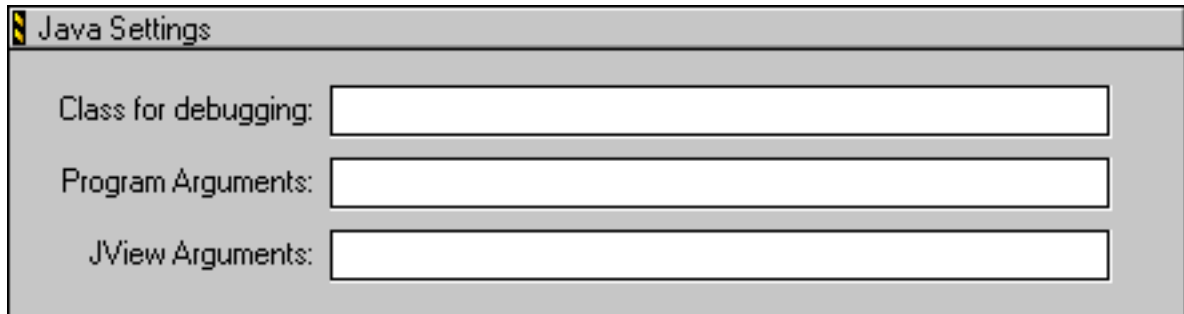
JDK version

Choose from this pop-up menu the Java Development Kit (JDK™) version you are using during the remote debugging session. This pop-up menu is enabled only when the [Remote Debugging](#) checkbox is enabled and the **1.1X Wire Protocol** is selected from the [Protocol](#) pop-up menu.

Java Settings (Windows)

The Java Settings preference panel, shown in [Figure 8.28](#), lets you configure general Java debugger settings. For more information about debugging Java programs and applets, see *Targeting Java*.

Figure 8.28 Java Settings preference panel



To learn how to open the IDE Preferences window and select the Java Settings panel, see [“Preferences Guided Tour” on page 253](#).

Class for debugging

Edit this field to specify the class file you want to debug.

Program Arguments

Edit this field to specify command-line arguments to be used by your project when debugging a Java application.

JView Arguments

Edit this field to specify any arguments JView might require while debugging your project.

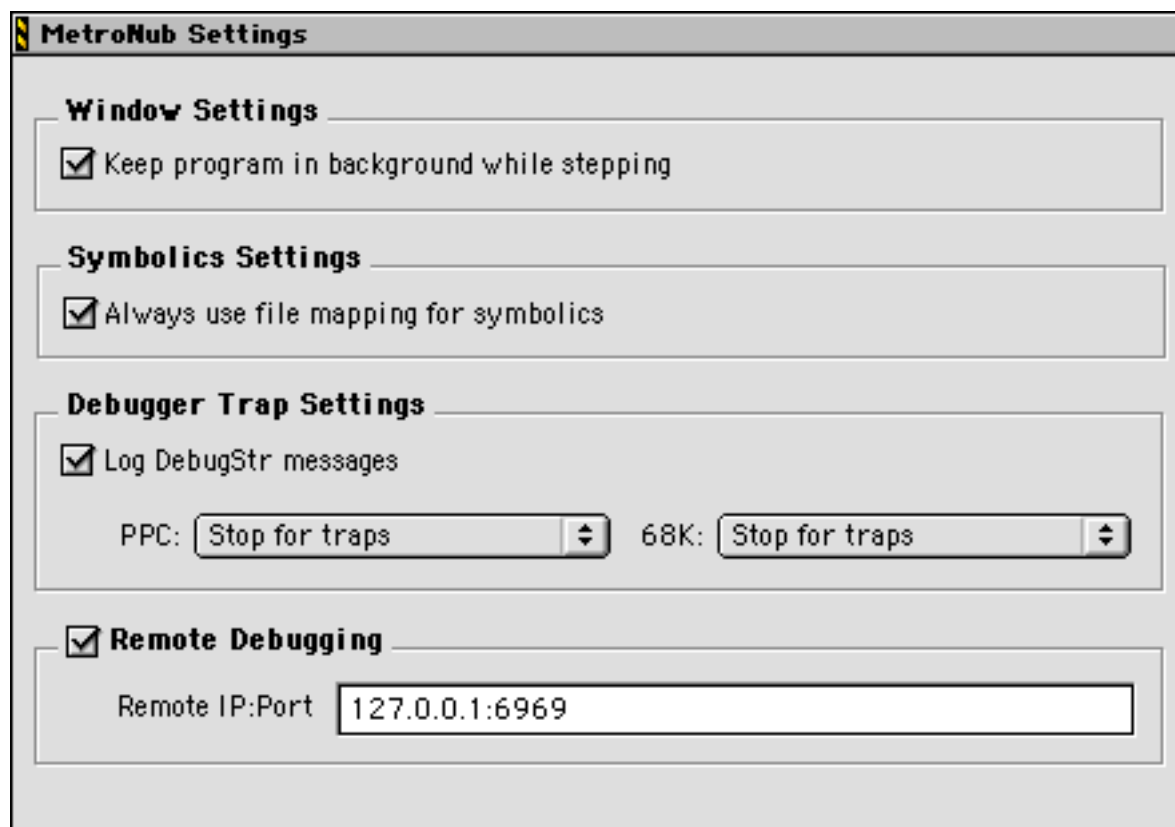
MetroNub Settings (Mac OS)

The MetroNub Settings preference panel is shown in [Figure 8.29](#). To learn how to open the IDE Preferences window and select the MetroNub Settings panel, see [“Preferences Guided Tour” on page 253](#).

Keep program in background while stepping

Enable this checkbox to step through a program’s code while the program is running in the background. This option accelerates stepping and avoids screen flicker. If you disable the checkbox, the program’s windows briefly come to the foreground and then return to the background. With rapid stepping, the constant movement of the program’s windows results in screen flicker.

Figure 8.29 MetroNub Settings preference panel (Mac OS)



WARNING! The **Keep program in background while stepping** option might affect the execution of your program if the program behaves differently in the background than in the foreground.

Always use file mapping for symbolics

This checkbox determines whether MetroNub or the IDE handles symbolics files. Enable this checkbox to let MetroNub handle the files. Disable this checkbox to let the IDE handle the files. This checkbox is enabled by default.

Debugger Trap Settings

These settings determine how the debugger handles `Debugger ()` and `DebugStr ()` traps when they are encountered during a program's execution.

Log DebugStr messages Enable this checkbox to log `DebugStr ()` messages when the debugger encounters them during a program's execution. If you do not want to log these messages, disable the checkbox.

PPC Use this pop-up menu to determine how the debugger handles PowerPC traps. You can choose to let MacsBug handle the traps (**MacsBug handles traps**), halt the program's execution after executing the traps (**Stop for traps**), or disregard the traps and continue the program's execution (**Ignore traps**).

68K Use this pop-up menu to determine how the debugger handles 68K traps. You can choose to let MacsBug handle the traps (**MacsBug handles traps**), halt the program's execution after executing the traps (**Stop for traps**), or disregard the traps and continue the program's execution (**Ignore traps**).

Remote Debugging

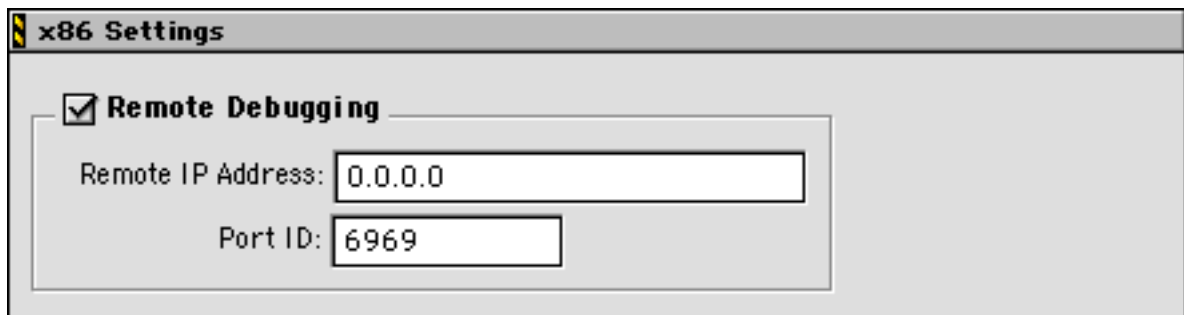
Enable this checkbox if you want to enable remote debugging of Mac OS 8 or Mac OS X projects from your computer.

Remote IP:Port Enter in this field the Internet Protocol (IP) address of the remote computer, followed by a colon, followed by the port ID number of the remote computer. The field shows sample values by default.

x86 Settings (Mac OS)

The x86 Settings preference panel, shown in [Figure 8.30](#), lets you set up your Mac OS computer to connect to a remote Windows computer for debugging sessions. To learn more about the x86 Settings panel, see *Targeting Mac OS*.

Figure 8.30 x86 Settings preference panel (Mac OS)



To learn how to open the IDE Preferences window and select the x86 Settings panel, see [“Preferences Guided Tour” on page 253](#).

Remote debugging

Enable this checkbox if you want to enable remote debugging of Windows projects from your computer.

Remote IP Address

Enter in this field the Internet Protocol (IP) address of the remote computer.

Port ID

Enter in this field the port ID number of the remote computer.

NOTE You should leave the port ID number at its default setting. Changing the default number might cause CodeWarrior to fail to connect to the remote computer.

Customizing the IDE

The IDE lets you customize menus, key bindings, and toolbars to better suit your programming needs. You customize these items in the Customize IDE Commands window. To display this window, choose **Commands & Key Bindings** from the Edit menu.

This section discusses the following topics:

- [Dialog Box Buttons](#)
- [Customizing Commands](#)
- [Customizing Key Bindings](#)
- [Customizing Toolbars](#)

Dialog Box Buttons

There are several dialog box buttons in the Customize IDE Commands window that control how settings are used and applied.

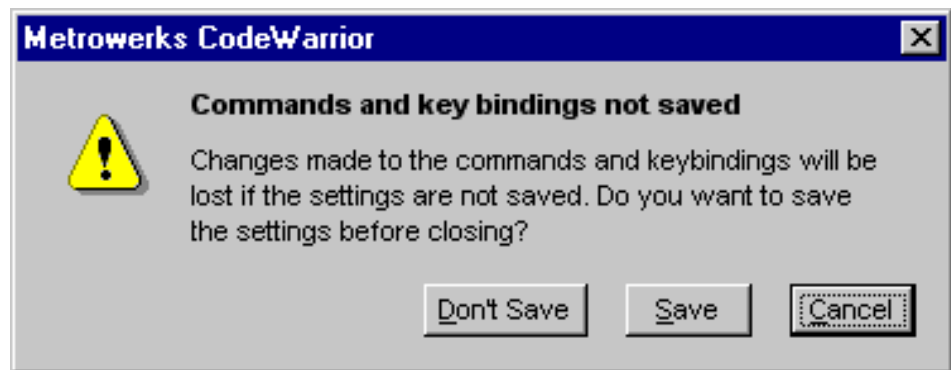
The topics in this section are:

- [Discarding changes](#)
- [Factory Settings button](#)
- [Revert Panel button](#)
- [Save button](#)

Discarding changes

If you change the settings in the Customize IDE Commands window and then try to close the window, CodeWarrior displays the dialog box shown in [Figure 8.31](#). To discard your changes, click **Don't Save**. To keep your changes, click the **Save** button. To discard your changes and return to the Customize IDE Commands window so that you can continue making changes, click **Cancel**.

Figure 8.31 Commands and Key Bindings Confirmation Dialog Box



Factory Settings button

The **Factory Settings** button causes the Customize IDE Commands window to revert to the settings that CodeWarrior uses as defaults.

WARNING! All settings are reset to their defaults when you click the **Factory Settings** button. If you do not export your customized command groups or menu commands before restoring the default settings, your customized items will be lost. See [“Exporting commands and key bindings” on page 310](#) for more information about exporting command groups and menu commands.

Revert Panel button

The **Revert Panel** button lets you undo changes to the Customize IDE Commands window. Clicking this button causes the window to reset to its last saved state. This is useful when you decide against the changes you made.

Save button

The **Save** button commits all of the current settings in the Customize IDE Commands window. After closing the window, CodeWarrior behaves according to the settings that you saved.

Customizing Commands

You can customize the menu commands that appear in the IDE menu bar. You can control the display of specific menu commands, create new command groups in which to place menu commands, and create new commands associated with a command line (Windows) or a script (Mac OS). The customized commands you create have access to IDE information, such as the current editor selection, the frontmost window, the current project, and the output file of the current project.

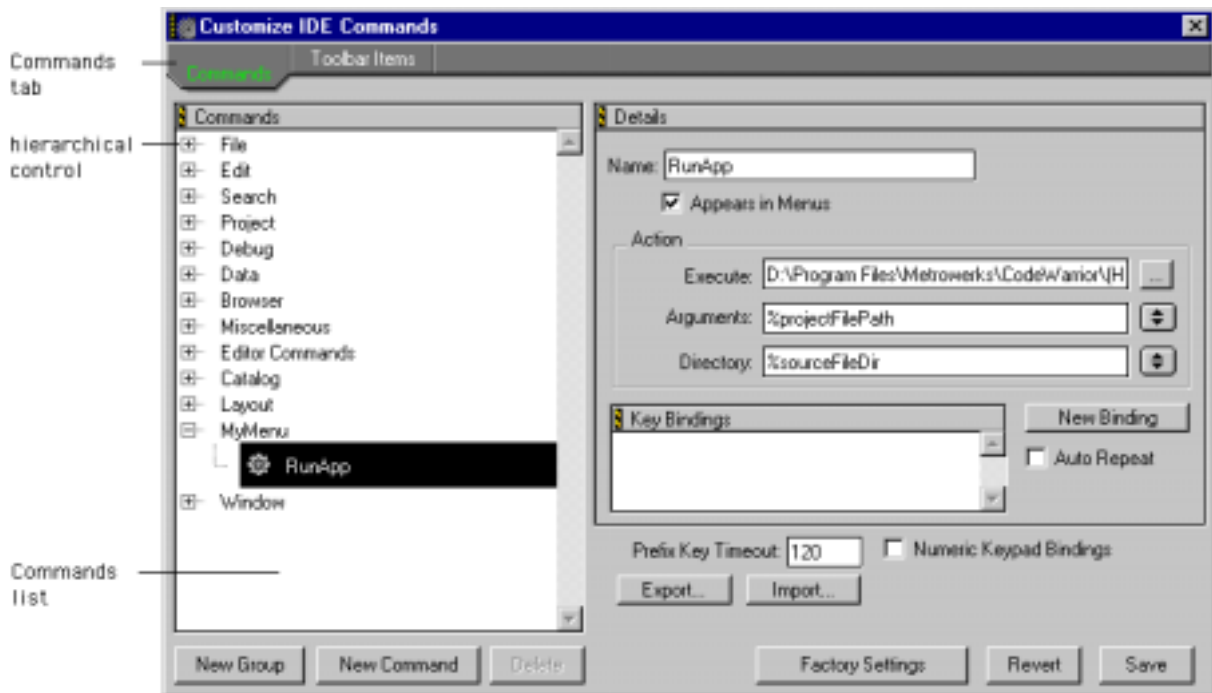
For example, you might wish to create a command group called **MyMenu** and place in it a command called **RunApp**, which launches an external application. The Customize IDE Commands window helps you define the new command group and command.

Click the **Commands** tab at the top of the Customize IDE Commands window to display the Commands view, shown in [Figure 8.32 on page 299](#) (Windows) and [Figure 8.33 on page 303](#) (Mac OS). You use this view to create and modify new command groups and commands for use with the IDE.

The topics discussed in the section are.

- [Modifying existing command groups and menu commands](#)
- [Adding customized command groups and menu commands](#)
- [Deleting customized command groups and menu commands](#)

Figure 8.32 Customize IDE Commands - Custom Commands (Windows)



Modifying existing command groups and menu commands

You can use the Commands view of the Customize IDE Commands window to examine and modify existing command groups and menu commands. The Commands view includes a Commands list, shown in [Figure 8.32](#). This hierarchical list organizes individual menu commands into command groups. Click the hierarchical control next to a command group to expand that group and view its contents.

To examine a particular item, select it in the Commands list. The information for the selected item appears on the right-hand side of the Customize IDE Commands window. The window provides the following information for each item:

- **Name**—This field shows the name of the selected item. If the IDE does not permit you to change the name of the selected item, this field is grayed out.
- **Appears in Menus**—Enable this checkbox to display the selected item in the specified position in the CodeWarrior menu bar. For example, enabling this checkbox for the **RunApp**

command in [Figure 8.32](#) allows the command to appear under the **MyMenu** command group in the menu bar. Disabling the checkbox prevents the **RunApp** command from appearing in the menu bar under the **MyMenu** command group.

- **Action**—This section shows various information about the action performed by the selected command. For default commands, this section describes the command type, such as **Command** or **Hierarchical Menu**. For customized commands that you create, this section lets you specify a command line (Windows) or a script (Mac OS) that runs when you choose the customized command.
- **Key Bindings**—For more information about the Key Bindings list, **New Binding** button, and **Auto Repeat** checkbox, see [“Customizing Key Bindings” on page 304](#).

Adding customized command groups and menu commands

The Customize IDE Commands window lets you add new menu commands to the CodeWarrior menu bar and arrange those new commands into command groups. The menu commands you create can run a command line (Windows) or an application or script (Mac OS).

Click the **Commands** tab at the top of the Customize IDE Commands window to display the Commands view, shown in [Figure 8.33 on page 303](#) (Windows) and [Figure 8.32 on page 299](#) (Mac OS). You use this view to create new menu commands and arrange them into command groups. The top-most level of the Commands list shows existing command groups. The contents of these groups show existing menu commands.

This section discusses the following topics:

- [Creating command groups](#)
- [Creating menu commands](#)
- [Defining menu command actions \(Windows\)](#)
- [Defining menu command actions \(Mac OS\)](#)

Creating command groups

To create a new command group in which to place menu commands, follow these steps:

1. Click the New Group button.

The IDE creates a new command group named **New Group**, adds it to the Commands list, and displays its information in the Customize IDE Commands window. The new command group appears in alphabetical order with existing command groups in the Commands list.

2. Enter a name for the new command group.

You can change the default name of **New Group**. Type the new name in the **Name** field of the Customize IDE Commands window.

3. Enable or disable the display of the new command group.

The **Appears in Menus** checkbox controls the display of the new command group in the CodeWarrior menu bar. When the checkbox is enabled, the command group appears in the menu bar. When the checkbox is disabled, the command group does not appear in the menu bar. Set this preference as desired.

Creating menu commands

To create a new menu command, follow these steps:

1. Select a command group in which to place the new menu command.

You must select an existing command group in the Commands list. To create a new command group in which to place the menu command, follow the steps described in [“Creating command groups” on page 300](#).

2. Click the New Command button.

The IDE creates a new menu command named **New Command** and places it in alphabetical order within the selected command group. The information for the new menu command appears in the Customize IDE Commands window.

3. Enter a name for the new menu command.

You can change the default name of the newly created menu command. Type the new name in the **Name** field of the Customize IDE Commands window.

4. Enable or disable the display of the new menu command.

The **Appears in Menus** checkbox controls the display of the new menu command in the selected command group. When the

checkbox is enabled, the menu command appears in the CodeWarrior menu bar under the command group. When the checkbox is disabled, the menu command does not appear under the command group. Set this preference as desired.

5. Define the desired Action for the new menu command.

See [Defining menu command actions \(Windows\)](#) or [Defining menu command actions \(Mac OS\)](#) for more information.

Defining menu command actions (Windows)

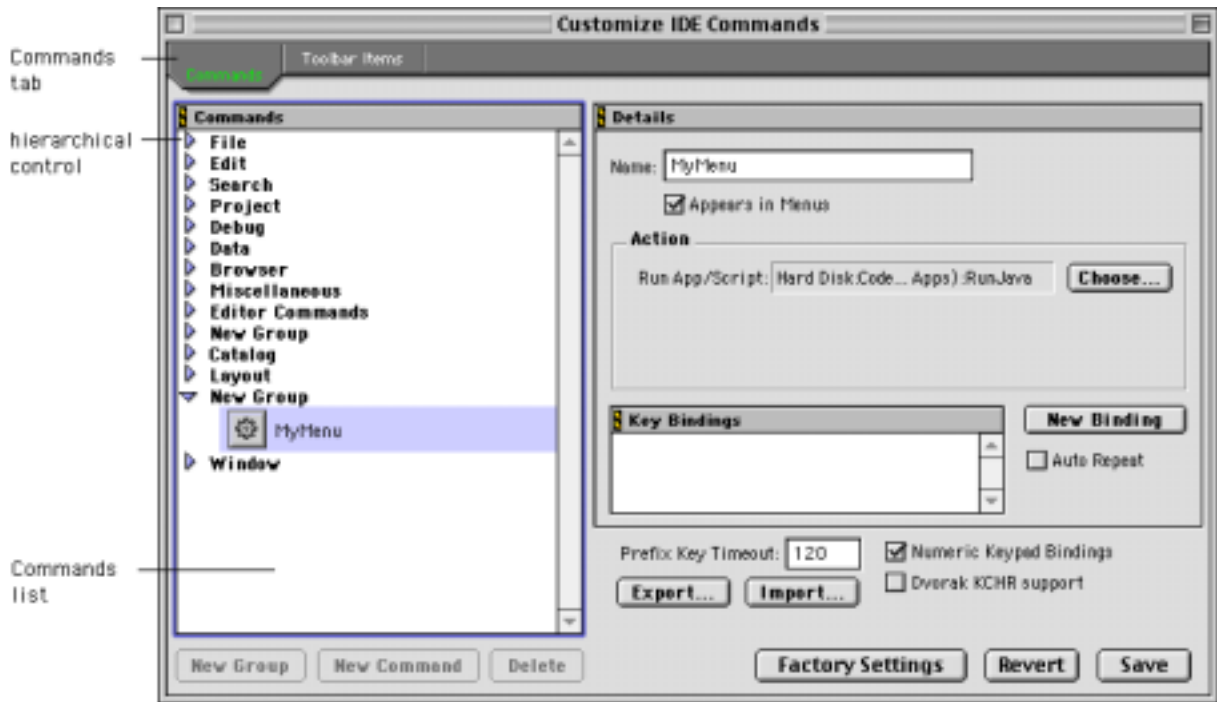
After creating a new menu command as described in [“Creating menu commands” on page 301](#), the Customize IDE Commands window displays three fields in the **Action** section. These fields help you define the action to be performed by the new menu command:

- **Execute**—Enter in this field a command line to run an application. Alternatively, click the button next to the field, shown in [Figure 8.32 on page 299](#), to select the application using a standard dialog box.
- **Arguments**—Enter in this field the argument that you want CodeWarrior to pass to the application specified in the **Execute** field. Alternatively, choose the desired argument from the pop-up menu next to the field, shown in [Figure 8.32 on page 299](#).
- **Directory**—Enter in this field the directory that you want CodeWarrior to pass to the application specified in the **Execute** field. Alternatively, choose the desired directory from the pop-up menu next to the field, shown in [Figure 8.32 on page 299](#).

Defining menu command actions (Mac OS)

After creating a new menu command as described in [“Creating menu commands” on page 301](#), the Customize IDE Commands window displays the **Run App/Script** field. This field, shown in [Figure 8.33](#), appears in the **Action** section of the window. Click the **Choose** button next to the field to display a standard Open dialog box. Use the dialog box controls to select an application or script. The IDE launches this application or script when you use the menu command. The path to the selected application or script appears in the **Run App/Script** field.

Figure 8.33 Customize IDE Commands - Custom Commands (Mac OS)



Deleting customized command groups and menu commands

You can delete the command groups and menu commands that you create for use with the IDE. Once removed, the command groups no longer appear and the menu commands no longer activate their associated command lines (Windows) or applications or scripts (Mac OS).

TIP If you only need to temporarily remove your customized command groups and menu commands, consider exporting your settings. Using this approach, you do not need to reconstruct your settings when you need to use them in the future. Instead, you can import your settings into the IDE and conveniently make them available for use once again. See [“Exporting commands and key bindings” on page 310](#).

To delete a command group or menu command, follow these steps:

1. **Select the command group or menu command you wish to delete.**

If necessary, click the hierarchical control next to a group to expand and view its contents.

2. **Click the Delete button.**

After clicking the **Delete** button, shown in [Figure 8.33](#), the selected command group or menu command disappears from the Commands list.

Customizing Key Bindings

You can customize the keyboard shortcuts used for menu, keyboard, and editor commands in the CodeWarrior IDE. You can attach, or “bind,” a set of keystrokes to virtually any command. When you want to activate the command, you can type its associated keystroke set, or “key binding.” The Customize IDE Commands window lets you change the default IDE key bindings.

For example, you might wish to change the key binding for [“Move Line Up”](#) from the Up-Arrow key to the keystroke Shift-Up Arrow. You could change this keyboard shortcut by changing the key binding for the command in the Customize IDE Commands window.

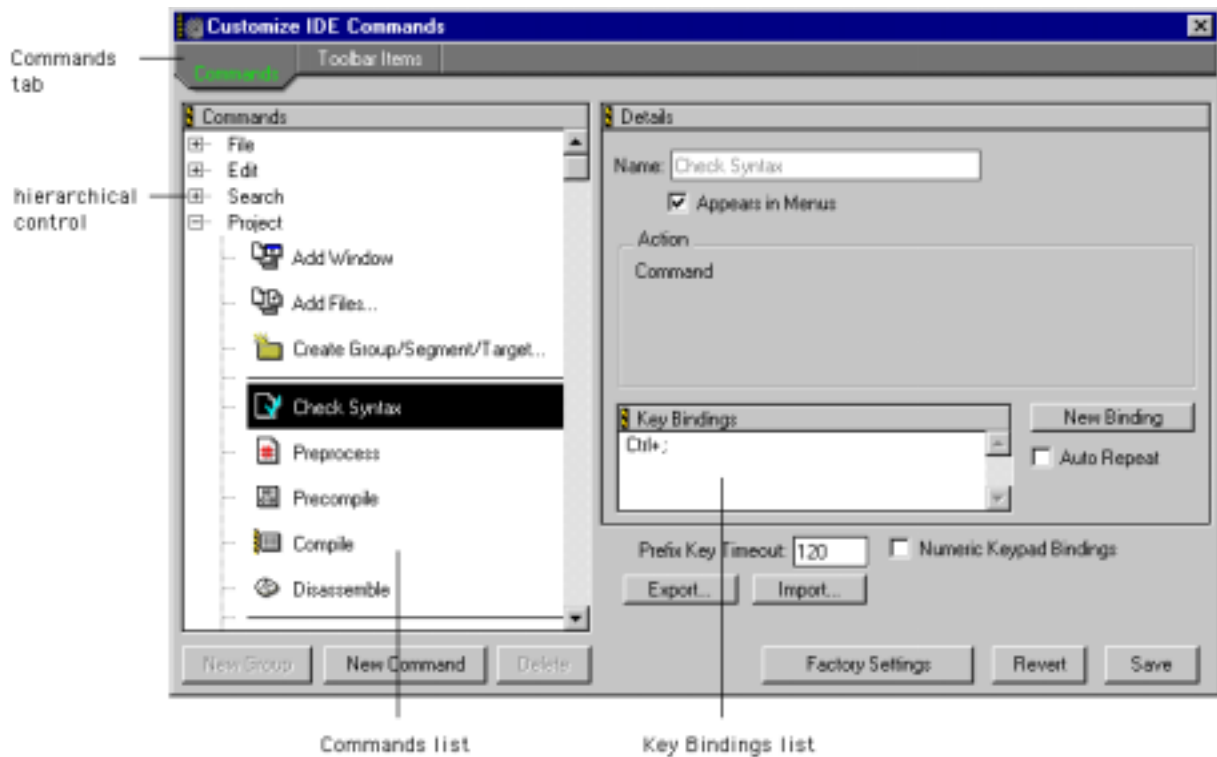
Click the **Commands** tab at the top of the Customize IDE Commands window to display the Commands view, shown in [Figure 8.34](#). You use this view to set the key bindings for menu commands, editor actions, and other miscellaneous actions. You can also specify special prefix keys. For a detailed list of default key bindings, see [“IDE Default Key Bindings” on page 660](#).

The topics discussed in this section are:

- [Restrictions on choosing key bindings](#)
- [Modifying key bindings](#)
- [Adding key bindings](#)
- [Deleting key bindings](#)
- [Setting Auto Repeat for key bindings](#)
- [Exporting commands and key bindings](#)

- [Importing commands and key bindings](#)
- [Prefix keys](#)
- [Quote Key prefix](#)
- [Setting the Prefix Key Timeout](#)

Figure 8.34 Customize IDE Commands - Key Bindings



NOTE (Solaris and Linux) By default, the Solaris- and Linux-hosted CodeWarrior IDE uses the same modifier key names as used for Mac OS (Shift, Command, Option, and Control). Likewise, the Customize IDE Commands window uses Mac OS symbols to represent modifier keys. [Table 17.1 on page 662](#) shows the default modifier key mappings and the symbols used to represent them.

On Solaris and Linux computers, modifier keys can be mapped to any key on the keyboard via the Keyboard Preferences dialog box in the Info menu. When reading this manual, you will need to keep the Keyboard Preferences settings in mind. See [“Keyboard Preferences](#)

[Dialog Box” on page 687](#) for more information on changing the default modifier key mappings.

Restrictions on choosing key bindings

When you customize key bindings, you need to be aware of some restrictions on keys that can and cannot be used. These restrictions include the following:

- The Escape and Space keys are always invalid for key bindings.
- Function keys and the Clear key are valid for creating key bindings.
- (Windows) The Return and Tab keys require at least the Control or Shift key. This restriction does not apply for the second key of a two-key sequence.
- (Mac OS) The Return and Tab keys require at least one of the following: Control, Command, or Shift. This restriction does not apply for the second key of a two-key sequence.
- (Mac OS) The Command-Period (Command-.) key combination is invalid for key bindings.

Modifying key bindings

The Commands view of the Customize IDE Commands window includes a Commands list, shown in [Figure 8.34 on page 305](#). This hierarchical list organizes the IDE’s commands into categories. Click the hierarchical control next to a category to expand that category and view its contents.

To modify a key binding, follow these steps:

1. **Select the command you wish to modify from the Commands list.**

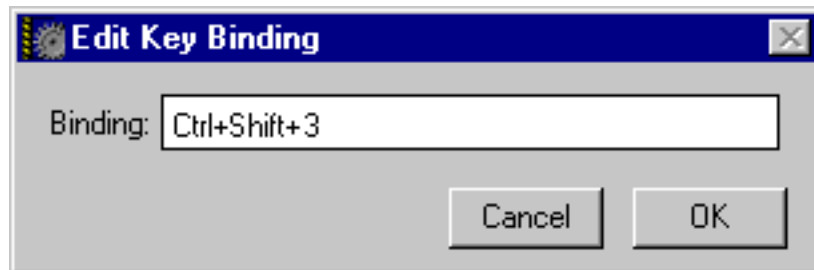
You might need to click the hierarchical control next to the appropriate category in order to view the individual commands.

NOTE If you want to use your keyboard’s numeric keypad as part of the new key binding, enable the **Numeric Keypad Bindings** checkbox in the Customize IDE Commands window.

2. **Double-click the key binding you wish to change in the Key Bindings list.**

The Key Bindings list is shown in [Figure 8.34 on page 305](#). After double-clicking the desired command in this list, the IDE displays the Edit Key Binding dialog box, shown in [Figure 8.35](#).

Figure 8.35 Edit Key Binding dialog box



3. **Press the key combination you wish to use for the selected command.**

For example, if you want to change the key combination to Control-8, you would press the Control key and the 8 key at the same time. If you decide against the key combination you just entered, or if you make a mistake, click the Cancel button in the Edit Key Binding dialog box. The IDE returns you to the Customize IDE Commands window.

NOTE (Mac OS) If you are using a Dvorak keyboard and you are experiencing problems with the Command key being recognized, enable the **Dvorak KCHR Support** checkbox in the Customize IDE Commands window.

4. **Click OK in the Edit Key Binding dialog box.**

The new key binding is displayed in the Key Bindings list of the Customize IDE Commands window.

5. **Click the Save button in the Customize IDE Commands window to save your changes.**

The new key binding is now available for use in the IDE.

Adding key bindings

In addition to modifying key bindings, the Customize IDE Commands window also lets you specify additional key bindings for a particular command.

To add a key binding, follow these steps:

1. **Select the command from the Commands list to which you want to add a new key binding.**

You might need to click the hierarchical control next to the appropriate category in order to view the individual commands.

NOTE If you want to use your keyboard's numeric keypad as part of the new key binding, enable the **Numeric Keypad Bindings** checkbox in the Customize IDE Commands window.

2. **Click the New Binding button.**

The **New Binding** button is shown in [Figure 8.34 on page 305](#). After clicking this button, the IDE displays the Edit Key Binding dialog box, shown in [Figure 8.35 on page 307](#).

3. **Press the key combination you wish to use for the selected command.**

For example, if you want to add the key combination Control-8, you would press the Control key and the 8 key at the same time.

If you decide against the key combination you just entered, or if you make a mistake, click the Cancel button in the Edit Key Binding dialog box. The IDE returns you to the Customize IDE Commands window.

NOTE (Mac OS) If you are using a Dvorak keyboard and you are experiencing problems with the Command key being recognized, enable the **Dvorak KCHR Support** checkbox in the Customize IDE Commands window.

4. **Click OK in the Edit Key Binding dialog box.**

The new key binding is displayed in the Key Bindings list of the Customize IDE Commands window.

5. **Click the Save button in the Customize IDE Commands window to save your changes.**

The new key binding is now available for use in the IDE.

Deleting key bindings

You can delete key bindings that you do not plan to use with the CodeWarrior IDE. Once removed, the key bindings no longer activate the selected command.

To delete a key binding, follow these steps:

1. **Select the command from the Commands list that contains the key binding you wish to delete.**

You might need to click the hierarchical control next to the appropriate category in order to view the individual commands.

2. **Select the key binding you wish to delete from the Key Bindings list.**

The Key Bindings list is shown in [Figure 8.34 on page 305](#).

3. **Press Backspace/Delete.**

The IDE removes the selected key binding from the Key Bindings list.

4. **Click the Save button in the Customize IDE Commands window to save your changes.**

The removed key binding no longer activates the selected command.

Setting Auto Repeat for key bindings

The **Auto Repeat** checkbox, shown in [Figure 8.34 on page 305](#), changes the automatic behavior of the items shown in the Key Bindings list.

Enable the **Auto Repeat** checkbox to automatically repeat the assigned command while you continue to press the key combination. Disable the checkbox to require the key bindings before activating the assigned command.

A useful example is the [Find Next](#) command. If you enable **Auto Repeat** for the [Find Next](#) key binding, then you can just hold down

the key combination while you watch the search engine find all the text matching your search criteria in the open file. This is a quick way to jump through a file, finding all the patterns you are looking for and showing them quickly. If you disable **Auto Repeat**, then you would have to release the keys and press them again every time.

Exporting commands and key bindings

If you wish to save your commands and key bindings in a file so that you can import them into the IDE at another time, you use the **Export** button in the Customize IDE Commands window. When you click this button, a standard Save dialog box displays. Navigate to the place on your hard disk where you want to save the commands and key bindings file and click **Save**.

TIP We suggest naming your commands and key bindings file with a .mkb file name extension, like this: MyCmsKbs.mkb. This naming convention helps you quickly identify the file on your hard disk. In addition, the Windows-hosted version of the CodeWarrior IDE uses this extension to identify the commands and key bindings file.

Importing commands and key bindings

If you wish to import saved commands and key bindings from a previously-exported file, click the **Import** button in the Customize IDE Commands window. When you click this button, a standard Open dialog box displays. Use the dialog box controls to locate and open the commands and key bindings file.

Prefix keys

Prefix keys let you create multiple-keystroke command keys, such as those used in the Emacs text editor available on many different computer platforms. For example, the Emacs key sequence to save a file is Control-X followed by Control-S.

To emulate this Emacs key binding in the CodeWarrior IDE, you can set a prefix key to Control-X, and then set the command key for the [Save](#) menu command to Control-X Control-S.

You can also adjust the maximum time to wait for a keypress after typing a prefix key. To learn how to do this, refer to [“Setting the Prefix Key Timeout” on page 312](#).

Quote Key prefix

The Quote Key is a special prefix key with a very simple function. It lets you use a simple printing character as a key equivalent (without any modifier key), and still retain the ability to use that character in regular type in the editor window.

In typical use, a key equivalent involves two keys: a modifier key (such as the Control key) combined with an actual printing key. However, you are not required to have a modifier key in CodeWarrior.

For example, you can assign the key for the number 1 (with no modifier) to a command. However, if you make this assignment, you can no longer simply type a 1 into your code in the editor. The reason for this problem is that the IDE interprets the 1 as a command. The Quote Key prefix is the way around such conflicts.

You can assign any key to be recognized as the Quote Key prefix. For more information, see [“Assigning the Quote Key prefix” on page 312](#). Despite its name, the Quote Key prefix does not have to be the key that creates quote symbols in text.

After typing an assigned Quote Key prefix, CodeWarrior interprets the next keypress as a keystroke, not as a command.

Returning to the earlier example, assume you assign the 1 key to a command. Assume also that you assign the tilde key (~) to be your Quote Key prefix. To execute the command, you would type the 1 key. To type a 1 into the editor, you would type the tilde key first, then the 1 key. (To type a tilde, you would press the tilde key twice.)

WARNING!

The Quote Key only affects the next key or combination of keys that you type. You must use the Quote Key once for each bound key or combination of keys for which you want to type the equivalent character on-screen.

Assigning the Quote Key prefix

To assign the Quote Key prefix, follow these steps:

1. **Click the hierarchical control next to the Miscellaneous item.**

The Miscellaneous item, shown in [Figure 8.36](#), is part of the Commands list in the Customize IDE Commands window.

2. **Select the Quote Key item.**

The Quote Key item is part of the Miscellaneous command group.

NOTE If you want to use your keyboard's numeric keypad as part of the new key binding, enable the **Numeric Keypad Bindings** checkbox in the Customize IDE Commands window.

3. **Click the New Binding button next to the Key Bindings list.**

The Key Bindings list is shown in [Figure 8.36](#). After clicking the **New Binding** button, the Edit Key Binding dialog box appears.

4. **Type the desired Quote Key prefix in the Edit Key Binding dialog box.**

The keys you type are displayed in the dialog box. If you make a mistake or decide against what you have typed, click **Cancel** to return to the Customize IDE Commands window.

NOTE (Mac OS) If you are using a Dvorak keyboard and you are experiencing problems with the Command key being recognized, enable the **Dvorak KCHR Support** checkbox in the Customize IDE Commands window.

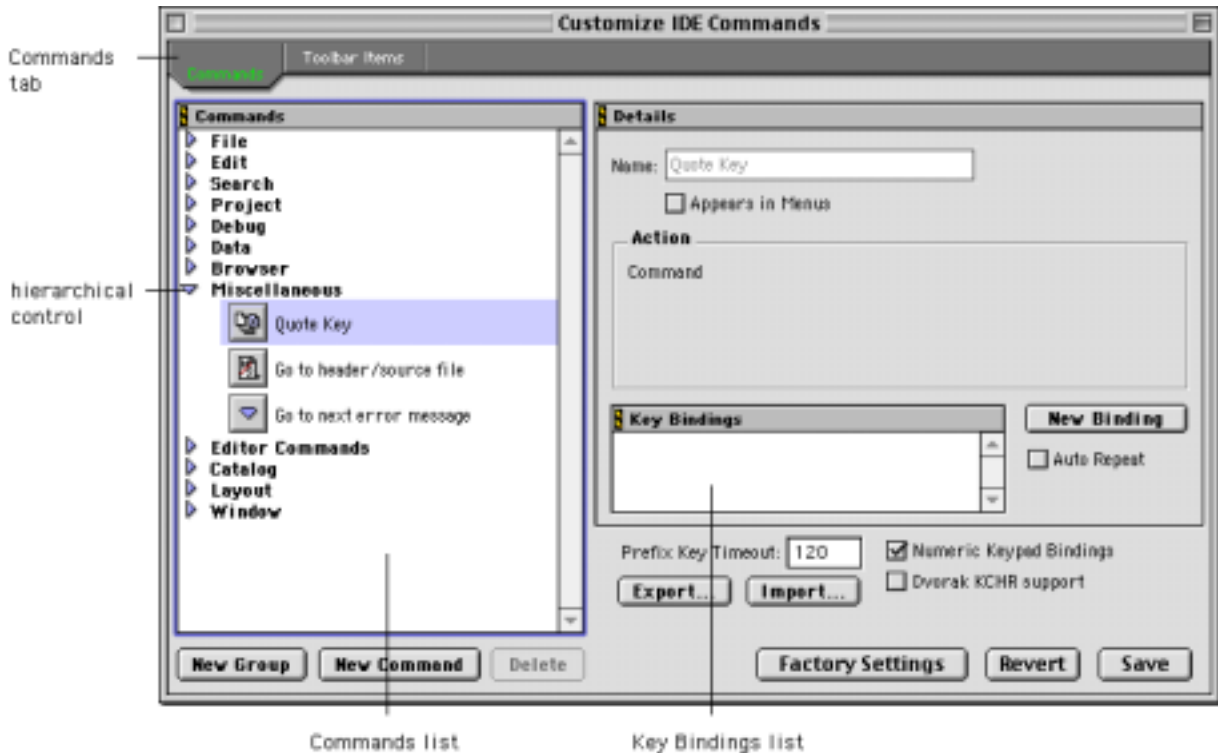
5. **Click OK in the Edit Key Binding dialog box.**

The new Quote Key prefix appears in the Key Bindings list.

Setting the Prefix Key Timeout

The **Prefix Key Timeout** field sets the length of time that the IDE waits for the second key after you press a prefix key. The larger the value in this field, the longer the IDE waits for you to press the second key.

Figure 8.36 Customize IDE Commands - Quote Key prefix



The timeout value is measured in “ticks,” where each tick is 1/60th of a second (16.67 milliseconds). Legal values range from 1 to 999. The default value is 120 ticks.

Customizing Toolbars

A toolbar contains a series of elements, represented by icons, that act as buttons. Each element typically represents a corresponding menu command. When you click the element, the IDE executes the associated command. The toolbar can also contain elements that carry out actions other than menu commands. [Figure 8.37](#) (Windows) and [Figure 8.38](#) (Mac OS) show the toolbar in the CodeWarrior Project window.

TIP (Mac OS) To display balloon help that describes a toolbar icon, place the cursor over the toolbar element, then press and hold the Control key.

Figure 8.37 The Project window toolbar (Windows)



This section discusses toolbars in detail. The topics are:

- [Kinds of toolbars](#)
- [Toolbar elements](#)
- [Showing and hiding a toolbar](#)
- [Modifying a toolbar](#)
- [Restoring a toolbar to default settings](#)
- [Anchoring the floating toolbar \(Mac OS, Solaris, and Linux\)](#)

Figure 8.38 The Project window toolbar (Mac OS, Solaris, and Linux)



Kinds of toolbars

There are two types of toolbars in CodeWarrior:

- the floating toolbar (sometimes referred to as the global toolbar or the main toolbar) that you can use at all times
- toolbars that appear in particular windows, such as the Project window toolbar and the Browser window toolbar

This distinction is important, because you can show, hide, clear, and reset the toolbar types using menu commands in the CodeWarrior IDE. These commands distinguish between the floating toolbar and the other window toolbars.

Your changes to one of these toolbar types apply to every instance of that toolbar type subsequently created. For example, if you modify the toolbar in an editor window, your changes appear in all editor windows opened thereafter. For more information, see [“Modifying a toolbar” on page 318](#).

Each of the toolbars has a default configuration of elements that you can restore at any time. For more information, see [“Restoring a toolbar to default settings” on page 320](#).

When you choose a menu command related to a window toolbar, the command affects the toolbar in the active window. For more information, see [“Toolbar Submenu” on page 653](#).

Toolbar elements

A toolbar can contain the following types of elements.:

- *Commands*—buttons that execute IDE menu commands when clicked
- *Controls*—the IDE pop-up menu buttons (Document Settings, Functions, Header Files, Markers, and Version Control), plus the Change Current Target button
- *Miscellaneous* —other elements (such as the File Dirty and File Path indicators)
- *Scripts* (Mac OS)—buttons that execute one of the scripts available through the Scripts menu in the IDE.

Click the **Toolbar Items** tab at the top of the Customize IDE Commands window to display the Toolbar view, shown in [Figure 8.39](#). You use this view when adding new elements to a toolbar. For more information on adding elements to a toolbar, see [“Adding a toolbar element” on page 318](#).

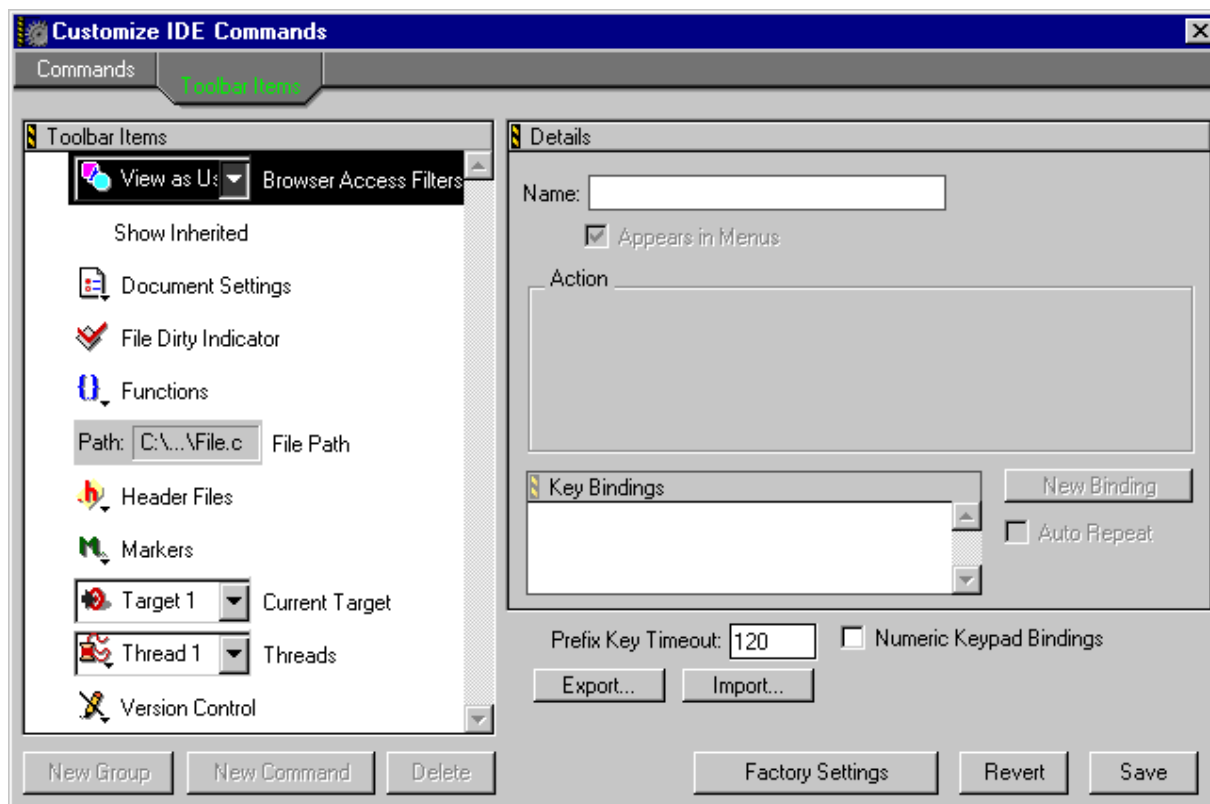
Showing and hiding a toolbar

Use the commands in the [Toolbar Submenu](#) to show or hide a toolbar. Hiding a toolbar does not change the toolbar’s configuration of elements.

The following types of toolbars are discussed:

- [Main toolbar \(Windows\)](#)
- [Floating toolbar \(Mac OS, Solaris, and Linux\)](#)
- [Window toolbar](#)

Figure 8.39 Customize IDE Commands - Toolbars



Main toolbar (Windows)

To display the main toolbar (sometimes called the floating toolbar), choose **Window > Toolbar > Show Main Toolbar**. To hide the main toolbar, choose **Window > Toolbar > Hide Main Toolbar**. For more information about restoring default toolbar settings, see [“Restoring a toolbar to default settings” on page 320](#).

Floating toolbar (Mac OS, Solaris, and Linux)

To display the floating toolbar, choose **Window > Toolbar > Show Floating Toolbar**. To hide the floating toolbar, choose **Window > Toolbar > Hide Floating Toolbar**. For more information about restoring default toolbar settings, see [“Restoring a toolbar to default settings” on page 320](#). For information about anchoring the floating toolbar, refer to [“Anchoring the floating toolbar \(Mac OS, Solaris, and Linux\)” on page 320](#).

NOTE (Mac OS, Solaris, and Linux) Hiding the floating toolbar does not change its anchored state. For example, if the floating toolbar is unanchored when hidden, it will remain unanchored when displayed again. For more information, see [“Anchoring the floating toolbar \(Mac OS, Solaris, and Linux\)” on page 320.](#)

Window toolbar

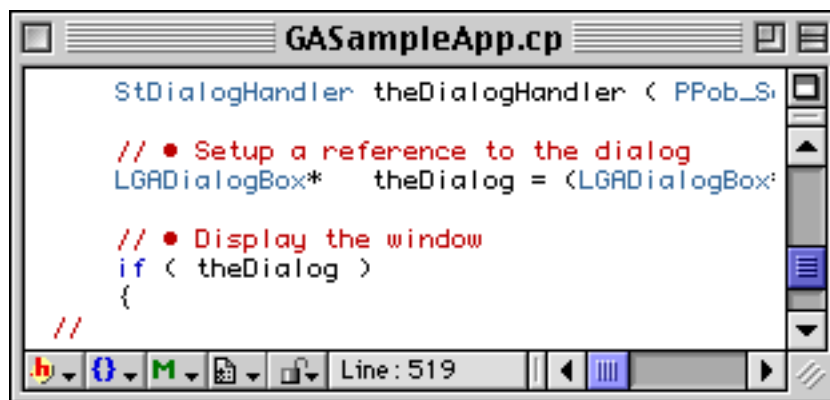
To hide an active window’s toolbar, choose **Window > Toolbar > Hide Window Toolbar**. The other components of the window shift to fill the space previously occupied by the window’s toolbar. All subsequently opened windows of that type will have hidden toolbars.

To display an active window’s hidden toolbar, choose **Window > Toolbar > Show Window Toolbar**. The other components of the window shift to allow room for the window’s toolbar. All subsequently opened windows of that type will display toolbars.

When you hide the editor window toolbar, the default tools appear along the bottom of the editor window, as shown in [Figure 8.40](#). When you show the editor window toolbar, the tools reappear along the top of the editor window, as shown in [Figure 5.1 on page 138](#).

You can also show or hide the editor window toolbar with the Toolbar Disclosure button. For more information, refer to [“Seeing Window Controls” on page 146](#).

Figure 8.40 Editor window with hidden toolbar



Modifying a toolbar

You can modify a toolbar by:

- [Adding a toolbar element](#)
- [Removing a toolbar element](#)
- [Removing all toolbar elements](#)

In certain circumstances, there are restrictions on which elements you can add or remove from a toolbar. These limitations are fully explained in [“Adding a toolbar element” on page 318](#) and [“Removing a toolbar element” on page 319](#).

If you modify a toolbar, the changes apply to every instance of that toolbar subsequently created. For example, if you customize the Project window toolbar, those changes will affect every Project window you open, not just the toolbar in the active Project window. Windows that are already open are not affected.

Adding a toolbar element

You add an element to a toolbar by dragging and dropping it from the Toolbar Items list onto a toolbar. This list is part of the **Toolbar Items** view in the Customize IDE Commands window.

To add an element from the Toolbar Items list to a toolbar, follow these steps:

1. **Select the element you want to add to a toolbar.**
Make sure the destination toolbar is visible as well.
2. **Drag the element from the Toolbar Items list to the destination toolbar.**

If the destination toolbar accepts the element, framing corners appear in the toolbar. These framing corners show you where the new element will appear when you release the cursor. If the destination toolbar does not accept the element, framing corners do not appear.

3. **Release the element at the desired position.**

After release, the element is inserted in the destination toolbar at the point indicated by the framing corners.

There are several reasons why a toolbar does not accept an element:

- the toolbar is full
- the element already exists on the toolbar
- commands can be added to a window's toolbar only for menu commands that are available when the current window is active
- the Document Settings, Functions, Header Files, Markers, and Version Control pop-up menus, as well as the File Dirty and File Path indicators, can only be added to an editor window toolbar
- the Current Target element can only be added to the Project window toolbar

Removing a toolbar element

You can remove toolbar elements to better suit your needs. To remove an element from a toolbar, Ctrl-right click (Windows) or Control-Command-click (Mac OS) on the element. Choose **Remove Toolbar Item** from the pop-up menu that appears. The element is then removed from the toolbar.

Removing all toolbar elements

You can clear all elements from a toolbar using commands in the Window menu. Clearing a toolbar is handy if you want to build your own toolbar from scratch.

Windows To clear the main toolbar, choose **Window > Toolbar > Clear Main Toolbar**.

Mac OS, Solaris, and Linux To clear the floating toolbar, choose **Window > Toolbar > Clear Floating Toolbar**.

To clear the toolbar from the current window, choose **Window > Toolbar > Clear Window Toolbar**.

In some cases, certain elements might not be removed by the **Clear Window Toolbar** command because they are critical to the window's basic purpose.

NOTE

If the floating toolbar is currently hidden, you cannot clear it. You must display the floating toolbar before clearing it. See [“Showing and hiding a toolbar” on page 315](#) for more information.

Restoring a toolbar to default settings

You can reset a toolbar to its original (default) factory settings by using commands in the Window menu.

Windows To reset the main toolbar, choose **Window > Toolbar > Reset Main Toolbar**.

Mac OS, Solaris, and Linux To reset the floating toolbar, choose **Window > Toolbar > Reset Floating Toolbar**.

To reset a the toolbar in the current window, choose **Window > Toolbar > Reset Window Toolbar**.

NOTE If the floating toolbar is currently hidden, you cannot reset it. You must display the floating toolbar before resetting it. See [“Showing and hiding a toolbar” on page 315](#).

Anchoring the floating toolbar (Mac OS, Solaris, and Linux)

You can anchor the floating toolbar to the top-left corner of the screen, just below the menu bar. In its anchored state, the floating toolbar is joined to the IDE menu bar, cannot be closed with a mouse-click, and cannot be moved. Unanchored, the floating toolbar can be positioned anywhere on your screen.

To anchor the floating toolbar when it is currently free-floating, choose **Window > Toolbar > Anchor Floating Toolbar**.

To release the floating toolbar when it is currently anchored, choose **Window > Toolbar > Unanchor Floating Toolbar**.

Configuring Target Options



This chapter discusses the [Target Settings](#) window. This window handles settings that affect a particular build target in a project.

The options in the Target Settings window are organized into a series of panels, and each panel is devoted to a particular topic. For example, one panel contains settings that specify folders to search for files listed in the Files view of a build target.

After customizing these options for a particular build target, you can create project stationery that incorporates your settings. To learn more about this topic, refer to [“Creating Your Own Project Stationery” on page 61](#).

The topics in this chapter include:

- [Target Settings Guided Tour](#)
- [Choosing Target Settings](#)

Target Settings Guided Tour

To open the Target Settings window, choose the [Target Settings](#) command from the **Edit** menu. When a project file is open, the actual name of the Target Settings command includes the name of the current build target. For example, if the name of your current build target is **ANSI Console Win32**, then the **Target Settings** command becomes **ANSI Console Win32 Settings**.

The topics in this section are:

- [Settings Panels](#)
- [Dialog Box Buttons](#)

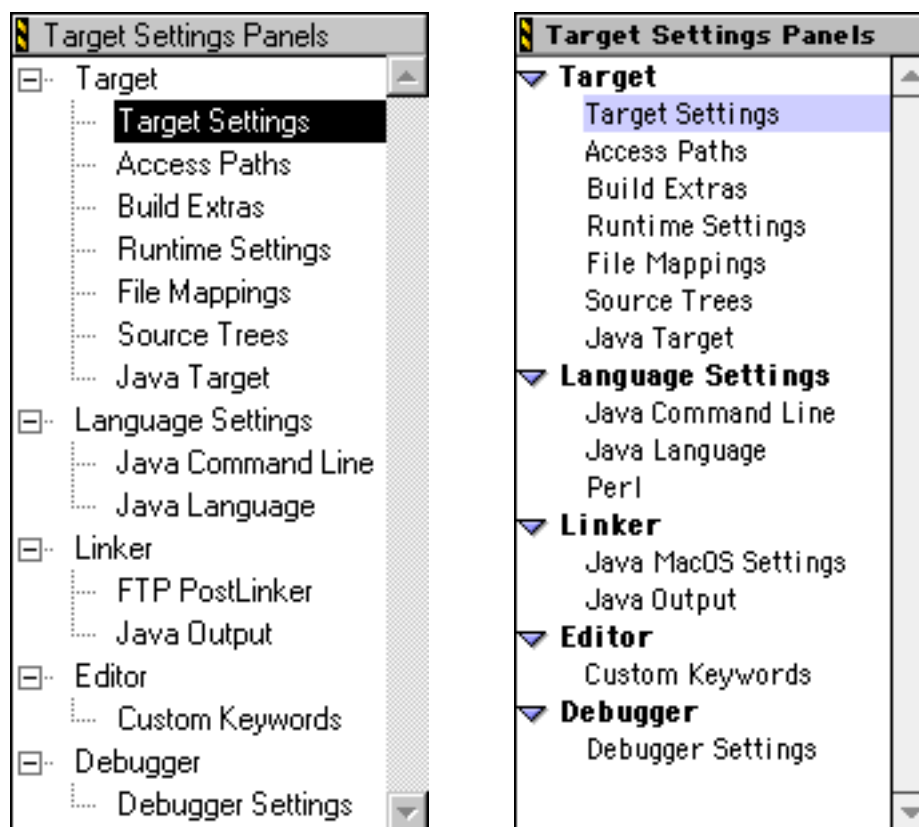
Settings Panels

The left side of the Target Settings window has a hierarchical list of available settings panels. The panel selected in this list appears on the right side of the window. The collection of panels varies, depending on the current build target and the CodeWarrior product you use.

To view a settings panel, select it in the list. You can use the arrow keys or click the name of the panel. [Figure 9.1](#) shows a selected panel in the Target Settings window.

Each settings panel consists of a series of related options. These options apply to the current build target in the active project. See the following section, [“Dialog Box Buttons”](#), for more information about applying or discarding changes to the settings panels.

Figure 9.1 **Selecting a settings panel**



After you set all the settings for a particular build target, you can create Project stationery that incorporates your settings when creating a new project. To learn more about this topic, refer to [“Creating Your Own Project Stationery” on page 61.](#)

Dialog Box Buttons

There are several dialog box buttons in the Target Settings window that control how a panel’s settings are used and applied.

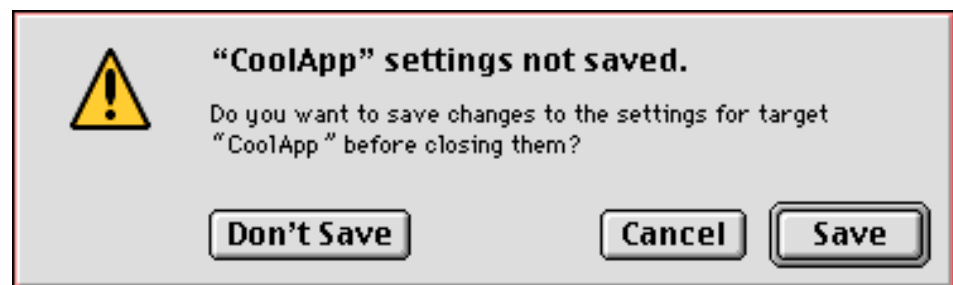
The topics in this section are:

- [Discarding changes](#)
- [Factory Settings button](#)
- [Revert Panel button](#)
- [Save button](#)

Discarding changes

If you make changes in the Target Settings window and attempt to close it, a dialog box similar to that shown in [Figure 9.2](#) might appear. To save your changes and close the dialog box, click **Save**. To discard your changes and close the dialog box, click the **Don’t Save** button. To continue using the Target Settings window without saving changes, click **Cancel**.

Figure 9.2 Settings Confirmation dialog box



Factory Settings button

The **Factory Settings** button causes the panel to revert to the settings that the CodeWarrior IDE uses as defaults. Settings in other

panels are not affected by this button. Only the settings for the current panel are reset.

Revert Panel button

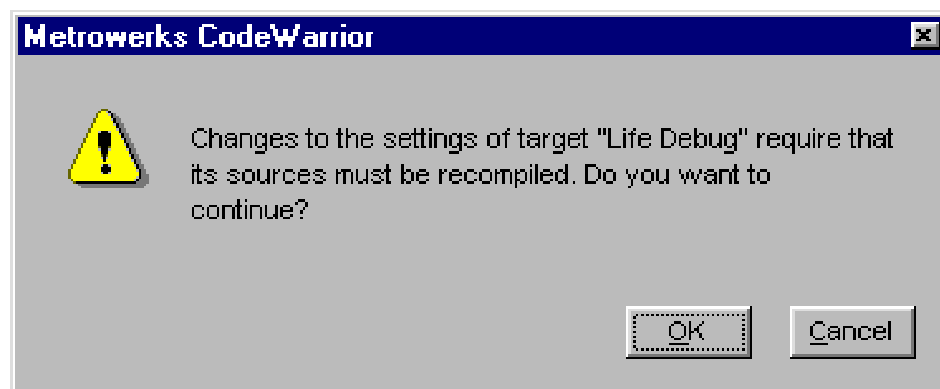
The **Revert Panel** button lets you reset the state of the current panel to its previously-saved settings. This is useful if you start making changes to a panel and then decide against your changes.

Save button

The **Save** button commits the current settings in all of the panels. If you change an option that requires CodeWarrior to recompile your project, the IDE displays a dialog box similar to that shown in [Figure 9.3](#). Click **OK** or **Cancel** depending on whether you want to keep your changes.

After closing the Target Settings window, CodeWarrior handles target builds according to the settings that you saved.

Figure 9.3 Save Changes dialog box



Choosing Target Settings

This section discusses setting options for individual build targets. These options configure the IDE to suit your needs when building the targets in your project.

To learn how to open the Target Settings window and select a particular settings panel, see ["Target Settings Guided Tour" on page 321](#).

The setting panels that appear in the Target Settings window depend on the operating system or chip family of your build target and the programming language that you use.

For example, if you are working with an x86 project, you do not see any settings panels for Motorola 68K configurations. If the C++ language is not available for a particular build target, panels related to that language do not appear.

In addition, the IDE ensures that only the files affected by a settings change are marked for recompilation. For example, changing a setting in the resource compiler causes the IDE to mark only resource compiler sources as dirty. This specific marking reduces the amount of recompilation required during the next [Make](#) operation and increases linking speed.

This manual does not discuss settings panels that are specific to a particular operating system, microprocessor family, or programming language. [Table 9.1](#) lists the various manuals where you can learn the details of OS-, processor-, or language-specific settings panels. This list includes the most common settings panels supported in various CodeWarrior products.

While some settings panels apply to a small number of build targets, other panels are pertinent to all build targets. This section discusses these general settings panels. They are organized into the following groups:

- [Target Configurations](#)
- [Code Generation Configurations](#)
- [Editor Configurations](#)
- [Debugger Configurations](#)

Table 9.1 **Where to learn about specific settings panels**

For these panels	See this manual
x86 Target Windows RC x86 Processor x86 Linker x86 Exceptions	<i>Targeting Win32</i>
Java Project Java VM Java Linker	<i>Targeting Java</i>
68K Target Rez 68K Processor 68K Disassembler 68K Linker CFM68K Linker	<i>Targeting Mac OS</i>
PPC Target PPCAsm Rez PPC Processor PPC Disassembler PPC Linker PPC PEF	<i>Targeting Mac OS</i>
C/C++ Compiler C/C++ Warnings	<i>C Compiler Guide</i>
Pascal Compiler Pascal Warnings	<i>Pascal Compiler Guide</i>

Target Configurations

The following settings panels apply to the CodeWarrior build targets:

- [Target Settings](#)
- [Access Paths](#)
- [Build Extras](#)
- [Runtime Settings](#)
- [File Mappings](#)

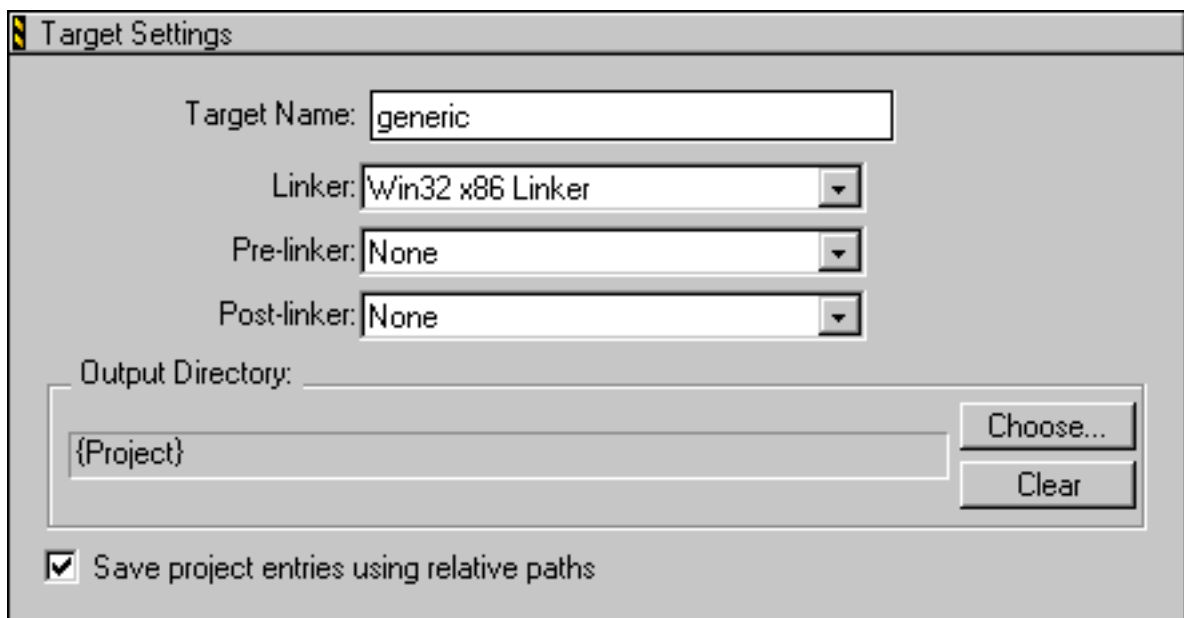
- [Source Trees](#)

Target Settings

The Target Settings panel, shown in [Figure 9.4](#), is the single most critical panel in CodeWarrior. This is the panel where you choose a target operating system and/or microprocessor for your project.

To learn how to open the Target Settings window and select the Target Settings panel, see [“Target Settings Guided Tour” on page 321](#).

Figure 9.4 Target Settings panel



The Target Settings panel lets you set the name of the current build target and the linker plug-ins for the build target. When you select a linker, you are specifying the target operating system and/or chip. The other panels available in the Target Settings window change to reflect your choice.

Because the linker choice affects the visibility of other related settings panels, you must set your build target first before you can specify other options like compiler and linker settings.

Configuring Target Options

Choosing Target Settings

It is possible to completely change build targets in this panel. When you change the build target, you must change the libraries contained in the project file for that build target. Choosing a new value for a build target does not change these related files for you. For this reason, you should remember to remove inappropriate libraries and add required libraries when changing the build target.

If you create a new project from stationery, the necessary library and support files are already included. You can then add source files from the old project. For more information on creating a project based on stationery, see [“Creating a New Project” on page 52](#).

Target Name

Use the **Target Name** field to set or change the name of the current build target. This name appears in the Targets view in the Project window.

The name you enter in this field is not the name of your final output file. Rather, this field shows the name you assign to the build target for your personal use. The name of the final output file is typically set in the settings panel for the linker you choose for your build target.

Linker

Choose a linker from the items listed in the **Linker** pop-up menu.

To learn more about the choices available to you, refer to the *Targeting* manual for your selected build target. The choices depend on the plug-in linkers you have for your CodeWarrior product.

For example, CodeWarrior Professional provides linkers for Mac OS on 68K and PowerPC, Win32 on x86, and the Java Virtual Machine. CodeWarrior for PlayStation OS provides a linker for the PlayStation OS running on MIPS, as well as a post-linker.

NOTE See [“File Mappings” on page 341](#) for more information about the file mappings associated with the linker you choose. These file mappings determine whether the IDE recognizes files in your project.

Pre-linker

Some build targets have pre-linkers that perform additional work on the object code in your project. This work takes place before the object code is linked into the final executable file. For more information about the pre-linkers available for your build target, refer to the information in the appropriate *Targeting* manual.

Post-linker

Some build targets have post-linkers that perform additional work (such as format conversion) on the final executable file. For more information about the post-linkers available for your build target, refer to information in the appropriate *Targeting* manual.

Output Directory

This field shows the location in which the IDE places your final linked output file. The default location is the directory that contains your project file. Click the **Choose** button to specify a different location. Click the **Clear** button to remove the current location.

Save project entries using relative paths

When this checkbox is enabled, the IDE remembers the location of a project entry as a relative path from one of the access paths. This extra location information lets the IDE distinguish different source files with the same name. The IDE remembers this location even if it needs to re-search for files in the access paths.

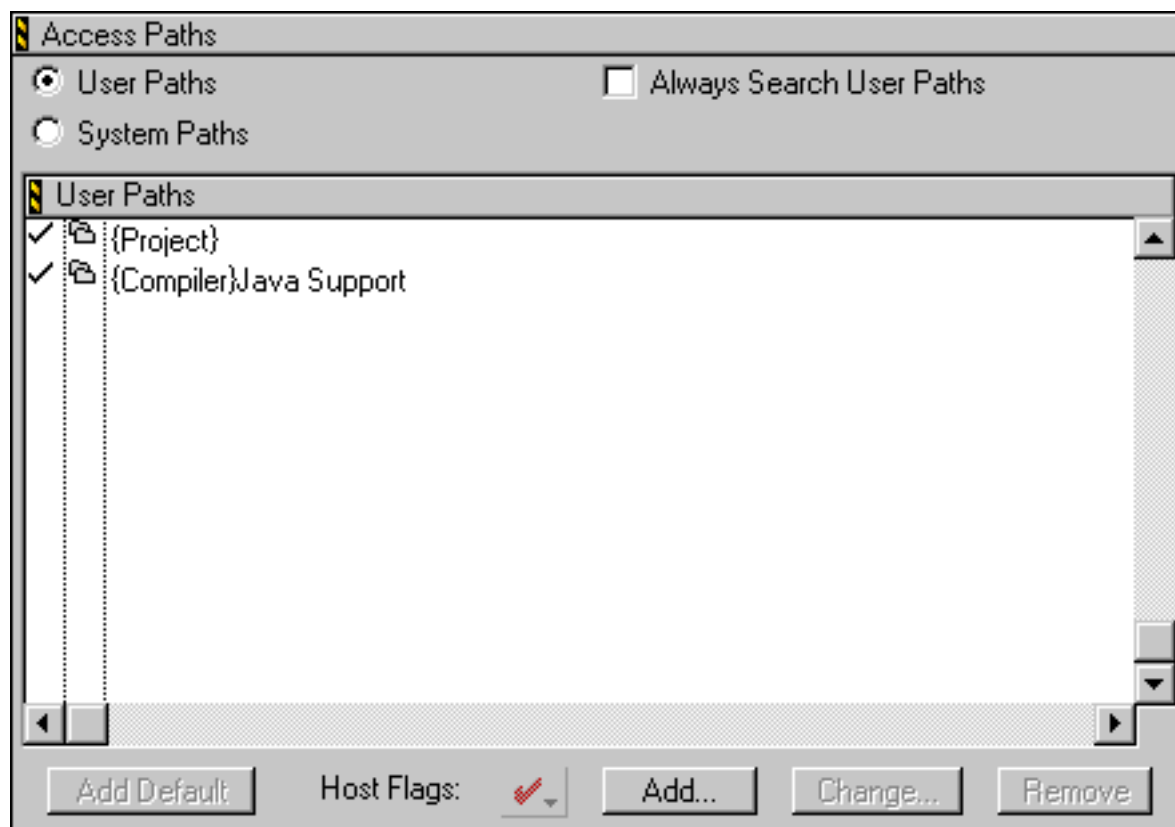
If the checkbox is disabled, the IDE only remembers project entries by name. Re-searching for files could cause the IDE to find the project entry in a different access path.

See [“Re-search for Files” on page 635](#) and [“Reset Project Entry Paths” on page 636](#) for more information.

Access Paths

If you need to define additional access paths for the CodeWarrior IDE to search while compiling and linking your project, use the Access Paths settings panel, shown in [Figure 9.5](#) (Windows) and [Figure 9.6](#) (Mac OS).

Figure 9.5 Access Paths settings panel (Windows)

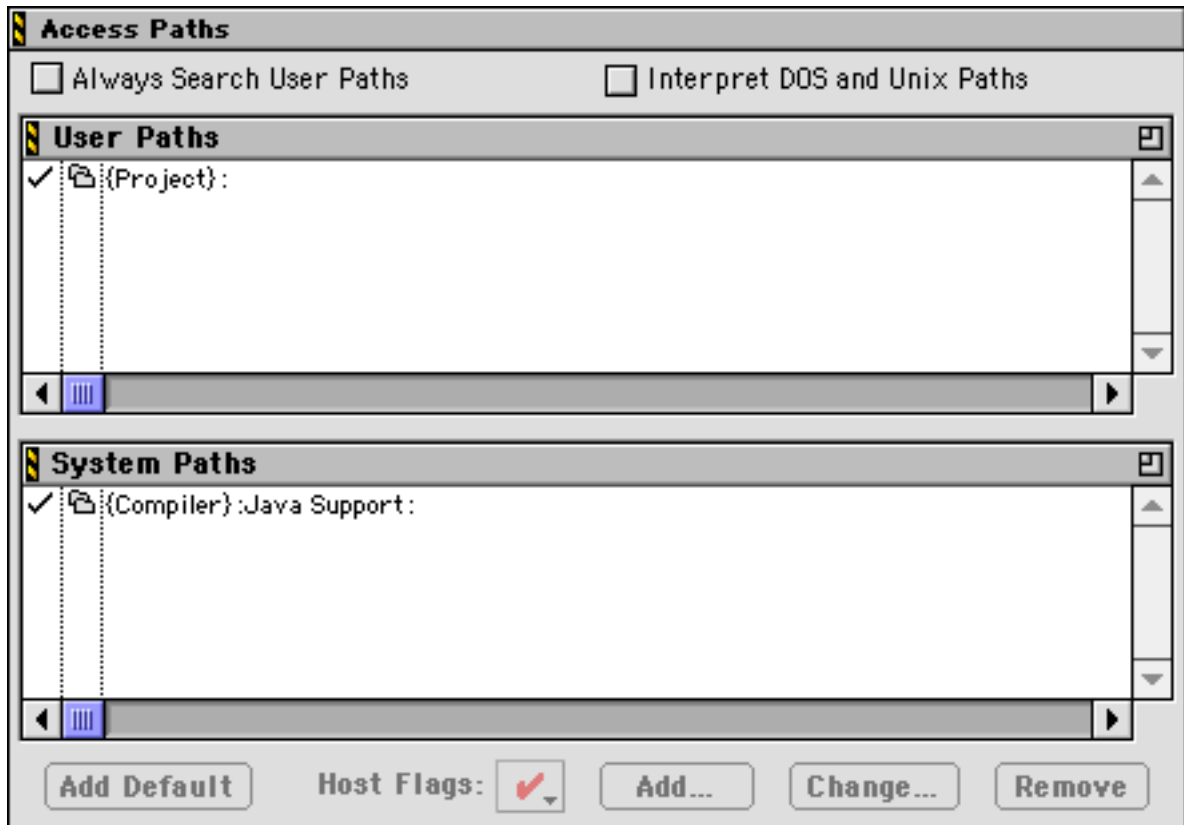


To learn how to open the Target Settings window and select the Access Paths panel, see [“Target Settings Guided Tour” on page 321](#).

If a folder icon appears beside the name of a folder in either the [User Paths pane](#) or the [System Paths pane](#), the IDE performs a recursive search on the path. That is, the CodeWarrior IDE searches the folder and all of its subfolders.

Clicking the folder icon to the left of any path in the [User Paths pane](#) or the [System Paths pane](#) toggles recursive searching of the subdirectories in that path. If the folder icon is visible, recursive searching is enabled. If the folder icon is not visible, recursive searching is disabled.

Figure 9.6 Access Paths settings panel (Mac OS)



TIP If you disable recursive searching of paths, and add each specific path of every directory that contains your files to either the User Paths pane or the System Paths pane, you can speed compilation of your project.

Clicking the checkmark icon to the left of any path in the User Paths pane or the System Paths pane toggles searching that path in the current CodeWarrior host computer. If the access path is checked, the current host computer searches that path. If the access path is not checked, the current host computer ignores that path.

If the files or libraries in your project are not located in the default access paths, the IDE will not find them when compiling, linking, or running your project. You must add their access paths to tell the IDE where to look.

Windows `Resource.frk` files are also automatically excluded from the search list.

TIP You can prevent the IDE from searching a folder and all its subfolders in an access path by renaming the folder with enclosing parentheses. For example, changing the name of a folder from `GameImages` to `(GameImages)` excludes it from all subsequent searches. To add the `(GameImages)` folder to the search list, you must explicitly add it as an access path.

For more information about access paths, see [“Interfaces pop-up menu” on page 114](#).

User Paths (Windows)

Click this radio button to display the [User Paths pane](#) in the Access Paths preference panel.

System Paths (Windows)

Click this radio button to display the [System Paths pane](#) in the Access Paths preference panel.

Always Search User Paths

To search for system header files or interface files in the same way as user header files, enable this checkbox. If this checkbox is disabled, the IDE treats `#include <...>` directives differently from `#include "..."` directives.

Interpret DOS and Unix Paths (Mac OS)

This checkbox determines how the IDE treats file names for interface files. If this checkbox is disabled, the IDE treats the backslash (`\`) and forward slash (`/`) characters as part of the file name in an interface file. If this checkbox is enabled, the IDE treats these characters as subfolder separator characters.

For example, consider the following directive:

```
#include "sys/socks.h"
```

If the **Interpret DOS and Unix Paths** checkbox is disabled, the IDE attempts to search for a file named `"sys/socks.h"` for use with

the project. If the checkbox is enabled, the IDE looks for a subfolder named “sys” that has a file named “socks.h”.

User Paths pane

This pane shows the access paths specific to your project. In Pascal, the IDE first searches these access paths. In C, an `#include "..."` directive causes the IDE to search these access paths. By default, the User Paths pane contains {Project}, which is the folder that contains the current project.

System Paths pane

This pane shows the access paths for system headers and similar files. In Pascal, the IDE searches these access paths after those in the [User Paths pane](#). In C, an `#include <...>` directive causes the IDE to search these access paths. By default, the System Paths pane contains {Compiler}, which is the folder that contains the CodeWarrior IDE.

Add Default

The CodeWarrior IDE lets you restore the default access paths in the [User Paths pane](#) or [System Paths pane](#) after you delete them. To add the default path to the active pane, click the **Add Default** button. The IDE adds the default path back into the active pane. Note that this button is not available if the default access path is currently present in the active pane.

Host Flags

This pop-up menu specifies the host platform that can use an access path. To allow a host platform to use a selected access path in the [User Paths pane](#) or [System Paths pane](#), choose the host platform from the **Host Flags** pop-up menu.

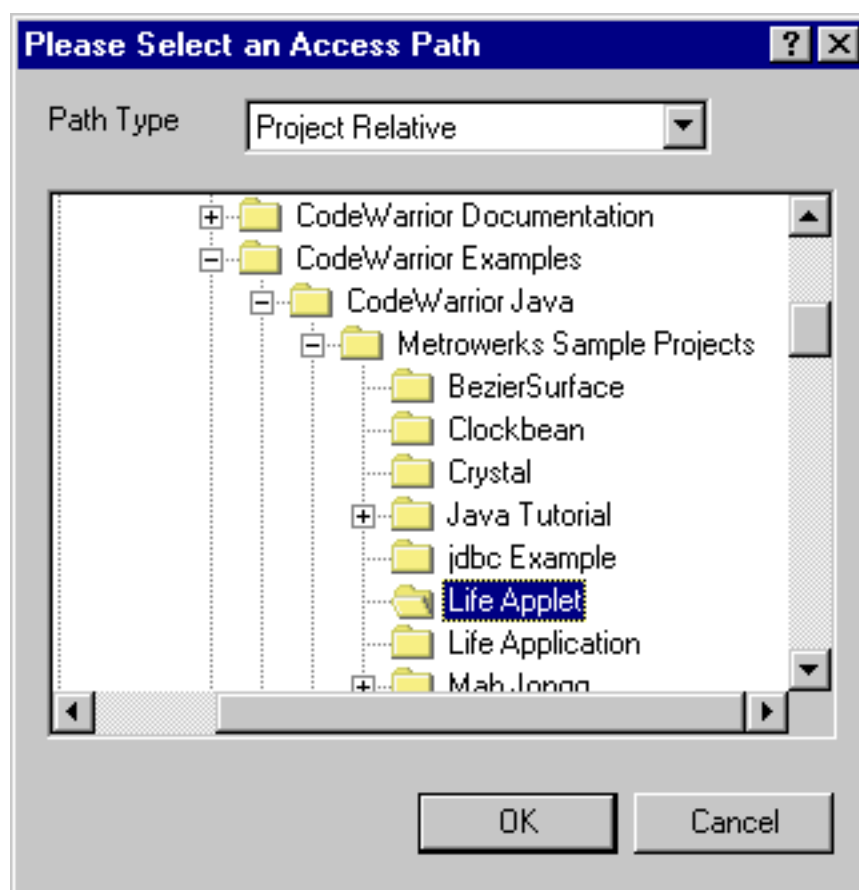
To allow all host platforms to use the access path, choose **All** from the pop-up menu. To prevent any host from using the access path, choose **None** from the pop-up menu.

For example, selecting a path and choosing **Mac OS** in the **Host Flags** pop-up menu specifies that the IDE must search the access path on a Mac OS computer and ignore the access path when running on any other host platform.

Add

To add a new access path, first select the [User Paths pane](#) or the [System Paths pane](#), then click the **Add** button. The dialog box shown in [Figure 9.7](#) (Windows) or [Figure 9.8](#) (Mac OS) appears. Use the dialog box controls to select the item for which you want to add an access path. You can also drag-and-drop folders onto the appropriate pane in order to add access paths.

Figure 9.7 Access Path Selection Dialog Box (Windows)



You can specify how CodeWarrior stores an access path by choosing one of the following options:

- **Absolute Path**—means that the IDE stores the access path from the root level of the startup hard drive to the folder whose access path you want to add, including all folders in between. You need to update absolute access paths if you move the project to

another system, rename the hard drive, or rename any of the folders along the access path.

- **Project Relative**—means that the IDE stores the access path from the folder that contains the project to the folder whose access path you want to add. You do not need to update project relative access paths if you move a project, as long as the hierarchy of the relative path is the same. You cannot create a relative path to a folder on a different hard drive than where your project file resides.

Figure 9.8 Access Path Selection Dialog Box (Mac OS)



- **Compiler Relative**—means that the IDE stores the access path from the folder that contains the CodeWarrior IDE to the folder whose access path you want to add. You do not need to update compiler relative access paths if you move a project, as long as the hierarchy of the relative path is the same. You cannot create a relative path to a folder on a different hard drive than where

your CodeWarrior IDE resides. On the Mac OS-hosted IDE, the **Compiler Relative** option is named **CodeWarrior Relative**.

- **System Relative**—means that the IDE stores the access path from the operating system's base folder to the folder whose access path you want to add. You do not need to update system relative access paths if you move a project, as long as the hierarchy of the relative path is the same. You cannot create a relative path to a folder on a different hard drive than where your active operating system's base folder resides.
- **Source Tree Relative**—means that the IDE stores the access path from the folder specified in the source tree. The actual name of this option includes the name of the source tree. For example, if you create a source tree named **Sample**, you can choose the **Sample** option when specifying how the IDE stores the access path. For more information about defining global source trees, see ["Source Trees" on page 266](#). For information about project-specific source trees, see ["Source Trees" on page 345](#).

NOTE Relative paths allow projects to contain two or more files with identical names. However, for large projects, you might notice slower performance when adding relative paths to a project.

Select an access path type from the available types, and then select the folder whose access path you want to add. Click **OK** to add the selected item's access path to the [User Paths pane](#) or the [System Paths pane](#), or click **Cancel** to discard your changes and leave the panes unchanged.

Change

To change an access path, first select the path in the [User Paths pane](#) or the [System Paths pane](#), then click the **Change** button. The dialog box shown in [Figure 9.7 on page 334](#) (Windows) or [Figure 9.8](#) (Mac OS) appears. Use this dialog box to select a new access path to replace the original path. To learn more about the options in the dialog box, refer to ["Add" on page 334](#).

Remove

To remove an access path, first select the path to be removed from the [User Paths pane](#) or the [System Paths pane](#). Then, click the **Remove** button to delete the path from the pane.

Windows You can also remove access paths by dragging them to the Recycle Bin on the Desktop.

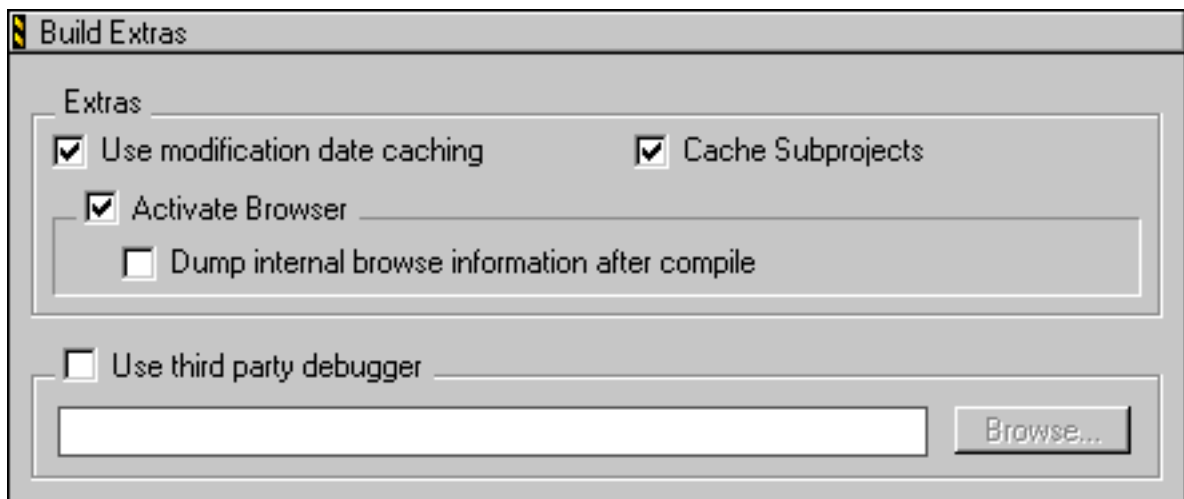
Mac OS You can also remove access paths by dragging them to the Trash on the Desktop.

Build Extras

The Build Extras panel contains various options that affect the way the IDE builds a project. These options are shown in [Figure 9.9](#) (Windows) and [Figure 9.10 on page 338](#) (Mac OS).

To learn how to open the Target Settings window and select the Build Extras panel, see [“Target Settings Guided Tour” on page 321](#).

Figure 9.9 Build Extras settings panel (Windows)



Use modification date caching

This checkbox determines whether the IDE checks the modification date of each project file prior to making the project. Enabling the checkbox causes the IDE to cache the modification dates of the files in a project. At compilation time, the IDE refers to this cache to determine whether a specific file should be recompiled. Disabling the checkbox causes the IDE to check every file each time you recompile the project.

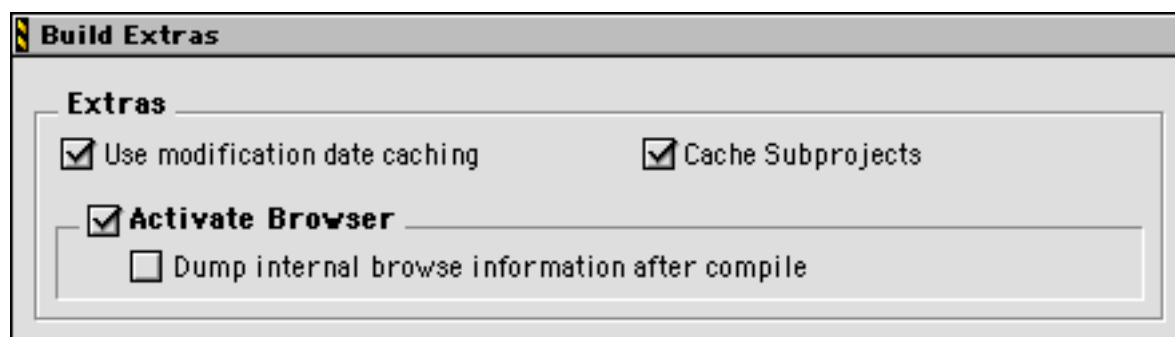
If you exclusively use the CodeWarrior editor to create and modify the text files in your project, enable the checkbox to shorten

compilation time. If you use other text editors in addition to the CodeWarrior editor, disable the checkbox to ensure that the IDE checks every file at compilation time.

Cache Subprojects

Enabling this checkbox improves multiproject updating and linking. Enabling the checkbox also allows the browser to generate symbolics information for both the build targets and the subprojects within each build target. Disabling the checkbox reduces the amount of memory required by the CodeWarrior IDE.

Figure 9.10 Build Extras settings panel (Mac OS)



Activate Browser

Enable this checkbox to let the IDE generate symbolics information required by the browser the next time your project is built. Without this information, you cannot open browser windows for your project.

See [“Making a Project” on page 366](#) for more information about rebuilding your project. To learn more about browser settings and options, see [“Browsing Source Code” on page 207](#).

Dump Internal Browse Information After Compile

Enable this checkbox to view the raw browser information that a plug-in compiler or linker provides for the IDE. This option is useful if you develop plug-ins for use with the IDE.

NOTE When the **Dump internal browse information after compile** checkbox is enabled, you should only compile single files or small

files. The internal browser information that the IDE displays can be huge when compiling an entire project.

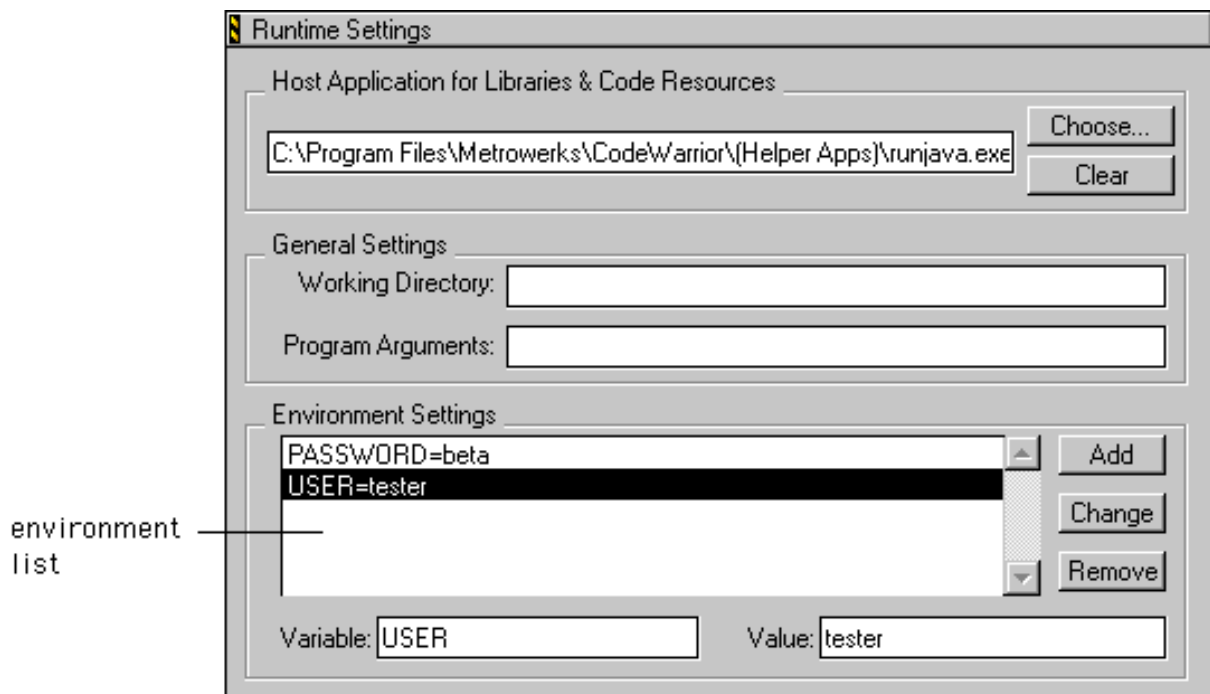
Use third party debugger (Windows)

Enable this checkbox to use a third-party debugger in place of the CodeWarrior debugger. Enter the path to the third-party debugger application in the field provided, or click the **Browse** button to locate the debugger application using a standard Open dialog box.

Runtime Settings

The Runtime Settings panel, shown in [Figure 9.11](#), includes options for specifying a debugging application for non-executable files, defining a working directory, listing program arguments, and creating environment variables.

Figure 9.11 Runtime Settings panel



Host Application for Libraries & Code Resources

This field lets you specify a host application to use when debugging a non-executable file, such as a shared library, dynamic link library

(DLL), or code resource. The application you specify in this field is not a debugger application, but rather the application with which the non-executable file interacts.

For example, if you write a Photoshop® plug-in, you would use the select Photoshop as the host application. When you use the **Debug** command, the IDE builds the plug-in, loads the symbolics information for the plug-in, and then launches Photoshop to let you debug the plug-in.

To specify a host application, type the path to that application directly into the field. Alternatively, click **Choose** to display a standard dialog box. Use the dialog box controls to select the application. Click **Clear** to remove the current path from the field.

General Settings (Windows)

This section has the following fields:

- **Working Directory**—Use this field to specify the default directory to which the current project has access. This location is where debugging occurs. If no directory is specified, debugging occurs in the same directory as the executable file.
- **Program Arguments**—Use this field to enter command-line arguments to pass to the project at the beginning of a debugging session. Your program receives these arguments when started with the **Run** command from the Project menu.

Environment Settings (Windows)

This section lets you specify environment variables that are passed to your program as part of the environment parameter in your program's `main()` function, or as part of environment calls. These variables are only available to the target program. When your program terminates, the settings are no longer available. For example, if you write a program that logs into a server, you could use variables for `PASSWORD` and `USER` as shown in [Figure 9.11](#).

To create a new environment variable, follow these steps:

1. **Type a name for the environment variable in the Variable field.**
2. **Type a value for the environment variable in the Value field.**

3. Click the Add button.

The new environment variable appears in the environment list shown in [Figure 9.11](#).

To modify an existing environment variable, follow these steps:

- 1. Select from the environment list the environment variable you wish to modify.**
- 2. Change the Variable and Value fields as desired.**
- 3. Click the Change button.**

The modifications appear in the environment list shown in [Figure 9.11](#).

To delete an existing environment variable, follow these steps:

- 1. Select from the environment list the environment variable you wish to delete.**
- 2. Click the Remove button.**

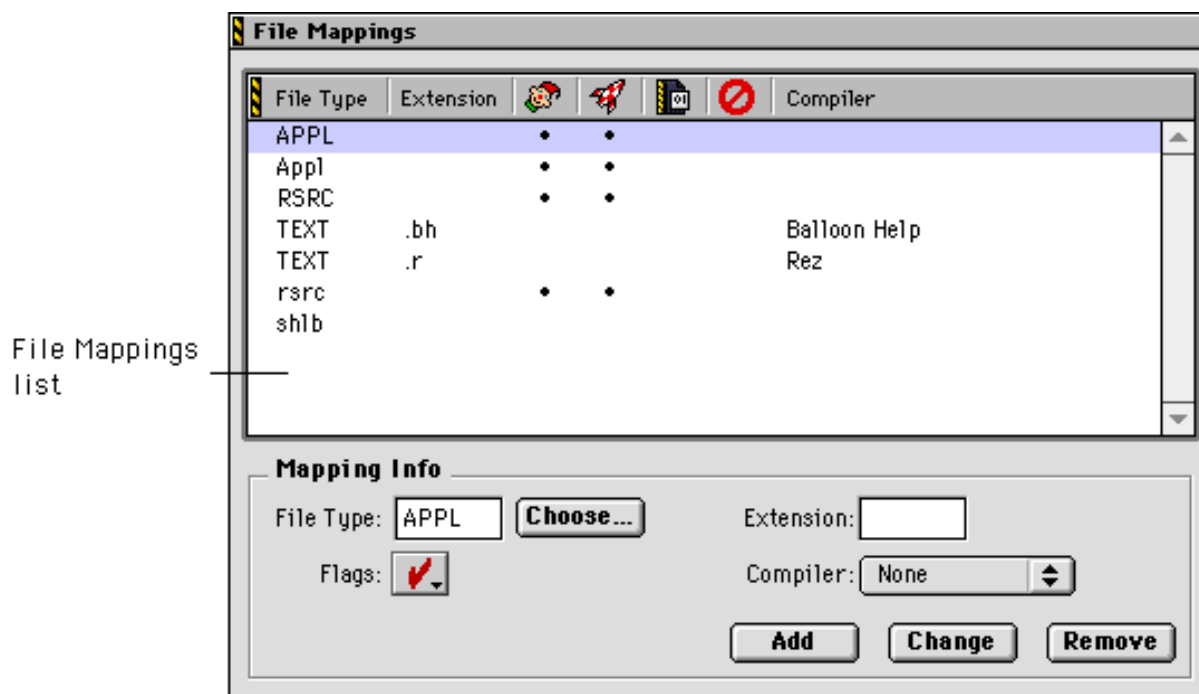
The selected environment variable is removed from the environment list shown in [Figure 9.11](#).

File Mappings

The File Mappings settings panel, shown in [Figure 9.12](#), is used to associate a file name extension such as “.c” or “.p” with a plug-in compiler. This tells the CodeWarrior IDE which compiler to use when a file with a certain name extension is encountered.

NOTE These file mappings determine whether the IDE recognizes files in the project. If you have trouble adding files to your project, or if the IDE refuses a folder or file that you drag and drop into the Project window, check the File Mappings settings panel. Because the file mappings are associated with the current linker, these mappings change as you select different linkers. For more information, see [“Linker” on page 328](#).

Figure 9.12 File Mappings settings panel



To learn how to open the Target Settings window and select the File Mappings Panel, see [“Target Settings Guided Tour” on page 321](#).

File Mappings list

The File Mappings list contains a **File Type**, associated **Extension**, and compiler choice for each file-name extension in the list. This list determines which compiler the IDE invokes when a given file name is encountered.

To add a new extension to this list, choose an existing entry in the list, edit the information in the **Mapping Info** section as desired and click the **Add** button.

For example, to add a documentation file mapping to the current project, you would follow these steps:

1. **Choose an existing file mapping.**
2. **Delete the text in the [File Type](#) field.**
3. **Change the text in the [Extension](#) field to “.doc” (without the quotes).**

4. Choose None from the [Compiler](#) pop-up menu.
5. Click the [Add](#) button.

File Type

Enter in this field the file type for the selected file mapping in the [File Mappings list](#).

Extension

This flag lets you enter a file name extension, such as the “.c” or “.h” extensions, for a selected [File Type](#) in the [File Mappings list](#). [Table 9.2](#) shows the default file name extensions used by the CodeWarrior IDE.

Table 9.2 Default file name extensions

Type	Extension	Description
Minimum CodeWarrior Installation	.iSYM	CodeWarrior Intel Symbols
	.mch	CodeWarrior Precompiled Header
	.mcp	CodeWarrior Project File
	.SYM	CodeWarrior Mac OS 68K Debug Symbols
	.xSYM	CodeWarrior Mac OS PPC Debug Symbols
Default CodeWarrior Installation	.dbg	CodeWarrior Debug Preferences
	.exp	Exported Symbol File
	.iMAP	CodeWarrior Link Map
	.MAP	CodeWarrior Link Map
	.xMAP	CodeWarrior Link Map

Configuring Target Options

Choosing Target Settings

Type	Extension	Description
Library	.lib	Library File
	.o	Object File
	.obj	Object File
	.pch	Precompiled Header Source File
	.pch++	Precompiled Header Source File
Default C and C++	.c	C Source File
	.cp	C++ Source File
	.cpp	C++ Source File
	.h	C and C++ Header File
Default Java	.class	Java Class File
	.jar	Java Archive File
	.jav	Java Source File
	.java	Java Source File
Default Pascal	.p	Pascal Source File
	.pas	Pascal Source File
Assembly	.a	Assembly Source File
	.asm	Assembly Source File
	.dump	CodeWarrior Disassembled File
C and C++	.c++	C++ Source File
	.cc	C++ Source File
	.hh	C++ Header File
	.hpp	C++ Header File
	.i	C Inline Source File
	.icc	C++ Inline Source File
	.m	Object C Source File
	.mm	Object C++ Source File

Type	Extension	Description
Java	.JMAP	Java Import Mapping Dump
	.jpob	Java Constructor File
	.mf	Java Manifest File
Pascal	.ppu	Pascal Precompiled Unit

Compiler

This pop-up menu lets you choose a compiler for the selected [File Type](#) in the [File Mappings list](#).

Flags

This pop-up menu lets you enable and disable four options:

Resource File Enabling this flag causes the IDE to include in your finished product the resources from files with the selected file mapping.

Launchable Enabling this flag causes the IDE to open the source-code file with the application that created it when you double-click the file in the Project window.

Precompiled Enabling this flag causes the IDE to compile files with the selected file mapping before compiling other files. This flag is useful if the original files create documents that other source files or compilers use. For example, this option lets you create a compiler that translates a file into a C source-code file and then compiles the C file. YACC (Yet Another Compiler Compiler) files are treated as precompiled files because YACC generates C source code to be compiled by a C compiler.

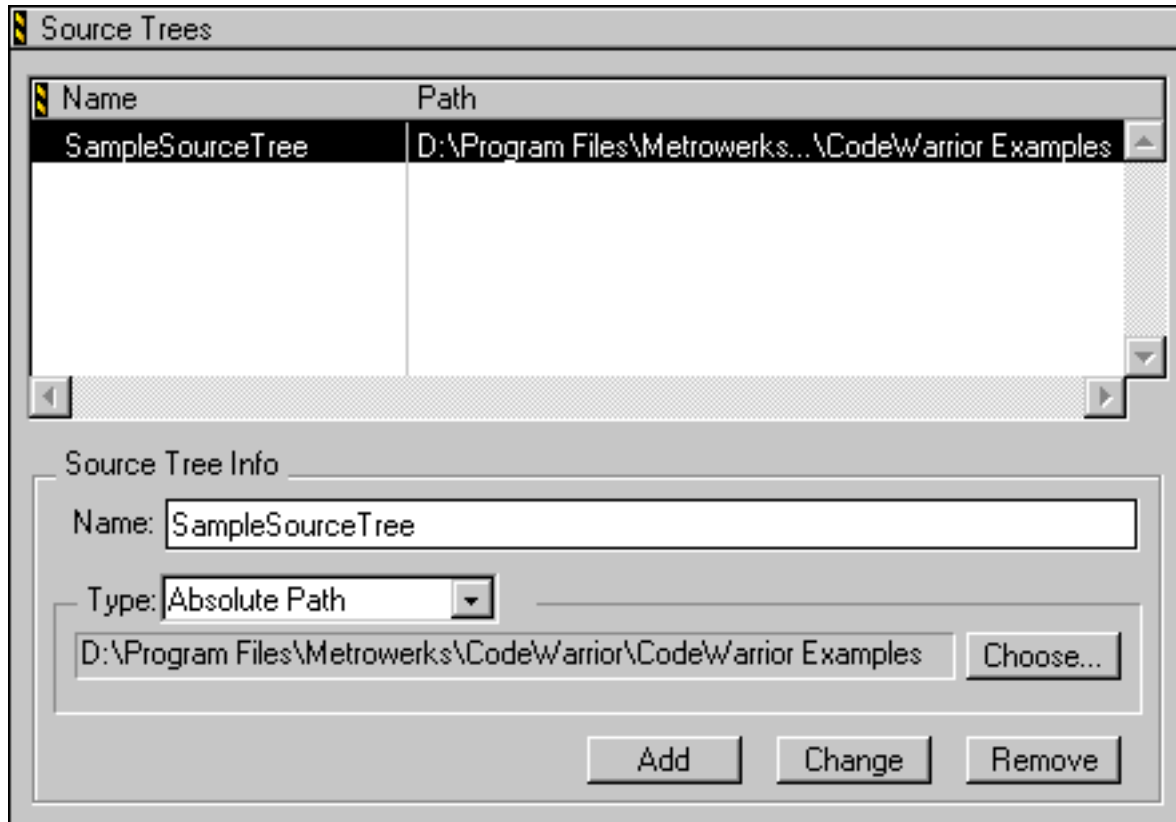
Ignored by Make Enabling this flag causes the IDE to ignore files with the selected file mapping when compiling or linking the project. This flag is useful if the files contain comments or documentation that you want to include with your project.

Source Trees

The Source Trees settings panel, shown in [Figure 9.13](#), lets you define project-specific source trees (root paths) for use in your projects. You can define your project's access paths and build-target

output in terms of source trees. Using this approach, you can easily share projects across various hosts. You need only make minor changes to the source trees' paths to maintain your project's functionality.

Figure 9.13 Source Trees settings panel



There is also a Source Trees preference panel that applies globally to the IDE. For more information about this settings panel, see [“Source Trees” on page 266](#). The source trees defined in the Target Settings window can only be used with the current build target in the active project. The source trees you define in the IDE Preferences window can be used for all projects. If you define the same source trees in both panels, the target-specific source trees take precedence over the global source trees.

Source Trees list

This list shows all of the project's source trees. The list consists of two columns:

- **Name**—This column shows the name of each source tree. When you define access paths in terms of source trees, you use this name in your access path's definition. For more information, see [“Add” on page 334](#).
- **Path**—This column shows the path to each source tree. You might need to modify the source trees' paths when you transfer your project to a new host. Refer to [“Change” on page 350](#) for more information about modifying the current path.

Name

This field is part of the **Source Tree Info** section shown in [Figure 9.13](#). Use the **Name** field to enter a name for a new source tree or to change the name of an existing source tree.

Type

This field is part of the **Source Tree Info** section shown in [Figure 9.13](#). Use the **Type** field to choose one of the following types of source trees:

- **Absolute Path**—This type of source tree is based on a file path.
- **Environment Variable**—This type of source tree is based on an existing definition of an environment variable. You cannot create or modify this type of source tree on the Mac OS-hosted IDE.
- **Registry Key**— (Windows) This type of source tree is based on an existing key entry in the registry.

Add

To add a new source tree, first select the appropriate type of source tree from the **Type** pop-up menu. Next, enter a name for the new source tree in the **Name** field.

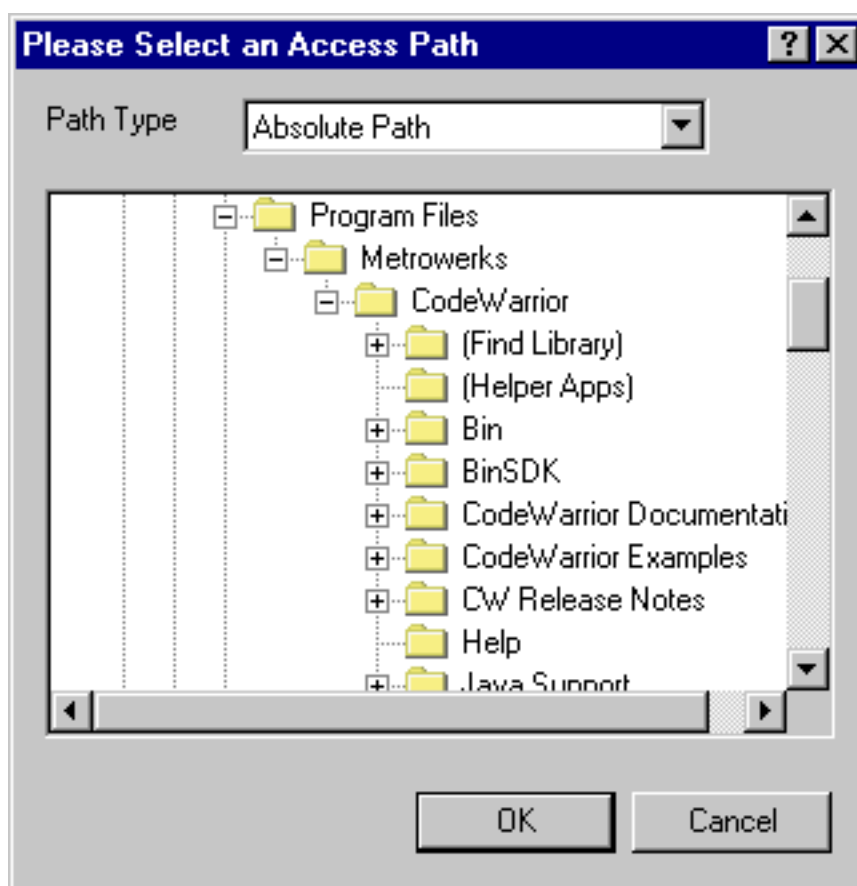
When you create an **Absolute Path** source tree, the **Choose** button is available. Click this button to select a path using a standard dialog box, as shown in [Figure 9.14](#) (Windows) and [Figure 9.15](#) (Mac OS).

On the Windows-hosted IDE, you can specify how CodeWarrior stores a path by choosing one of these path type options:

- **Absolute Path**—means that the IDE stores the path from the root level of the startup hard drive to the folder whose path you want to add, including all folders in between. You need to

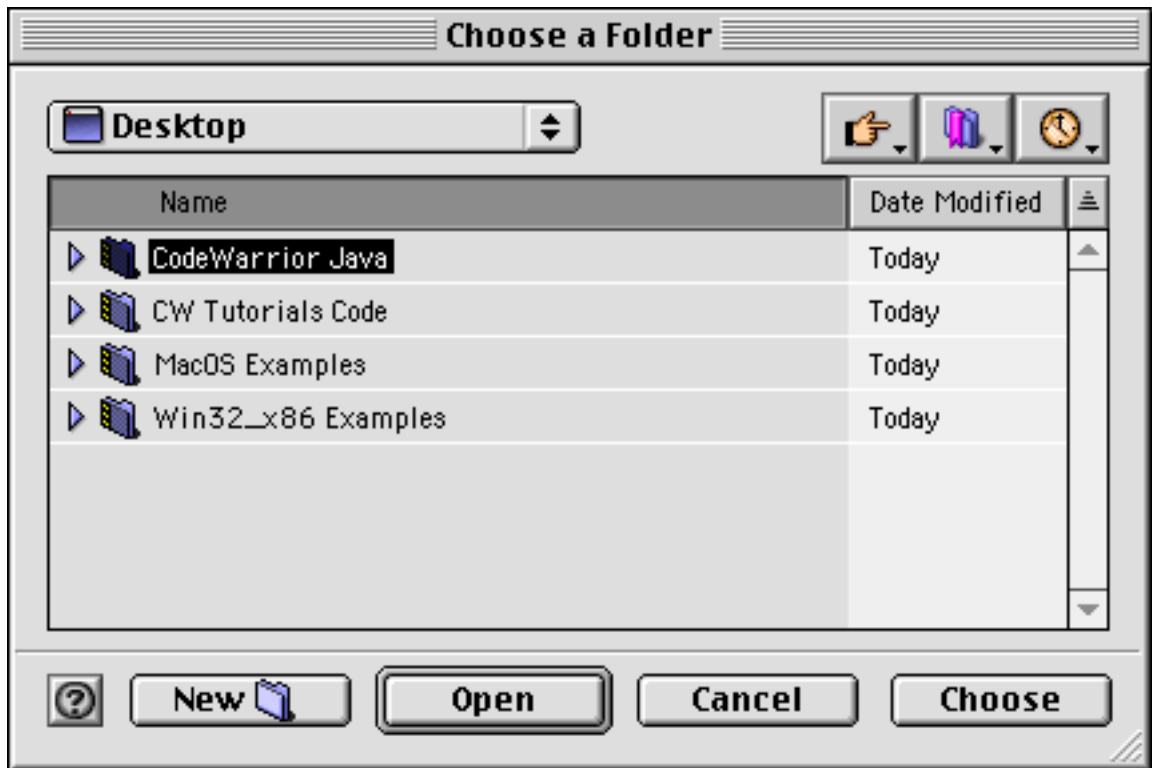
update absolute paths if you move the project to another system, rename the hard drive, or rename any of the folders along the path.

Figure 9.14 Access Path Selection dialog box (Windows)



- **Compiler Relative**—means that the IDE stores the path from the folder that contains the CodeWarrior IDE to the folder whose path you want to add. You do not need to update compiler-relative paths if you move a project, as long as the hierarchy of the relative path is the same. You cannot create a relative path to a folder on a different hard drive than where your CodeWarrior IDE resides.

Figure 9.15 Access Path Selection dialog box (Mac OS)

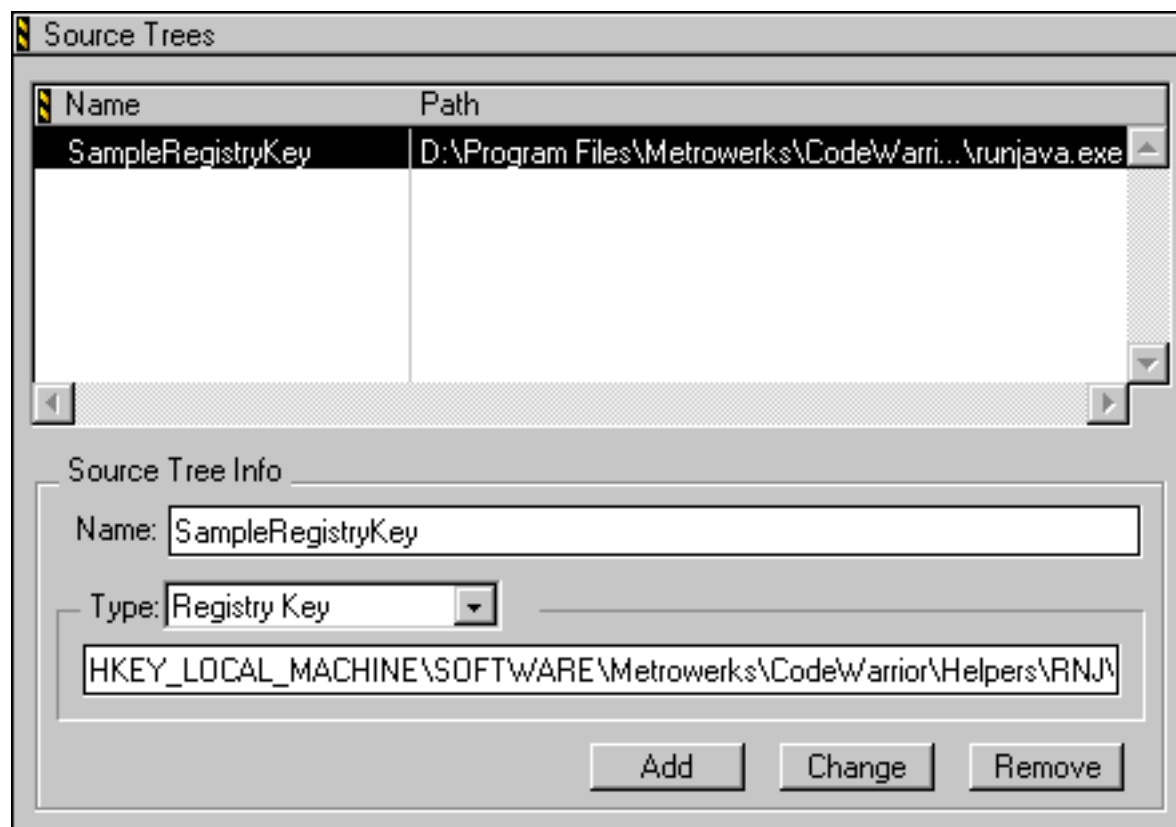


- **System Relative**—means that the IDE stores the path from the operating system's base folder to the folder whose path you want to add. You do not need to update system relative paths if you move a project, as long as the hierarchy of the relative path is the same. You cannot create a relative path to a folder on a different hard drive than where your active operating system's base folder resides.

NOTE Relative paths allow projects to contain two or more files with identical names. However, for large projects, you might notice slower performance when adding relative paths to a project.

When you create an **Environment Variable** or **Registry Key** source tree, the **Type** field changes to the field shown in [Figure 9.16](#). Enter the path to the environment variable or registry key in this field.

Figure 9.16 Creating a registry key (Windows)



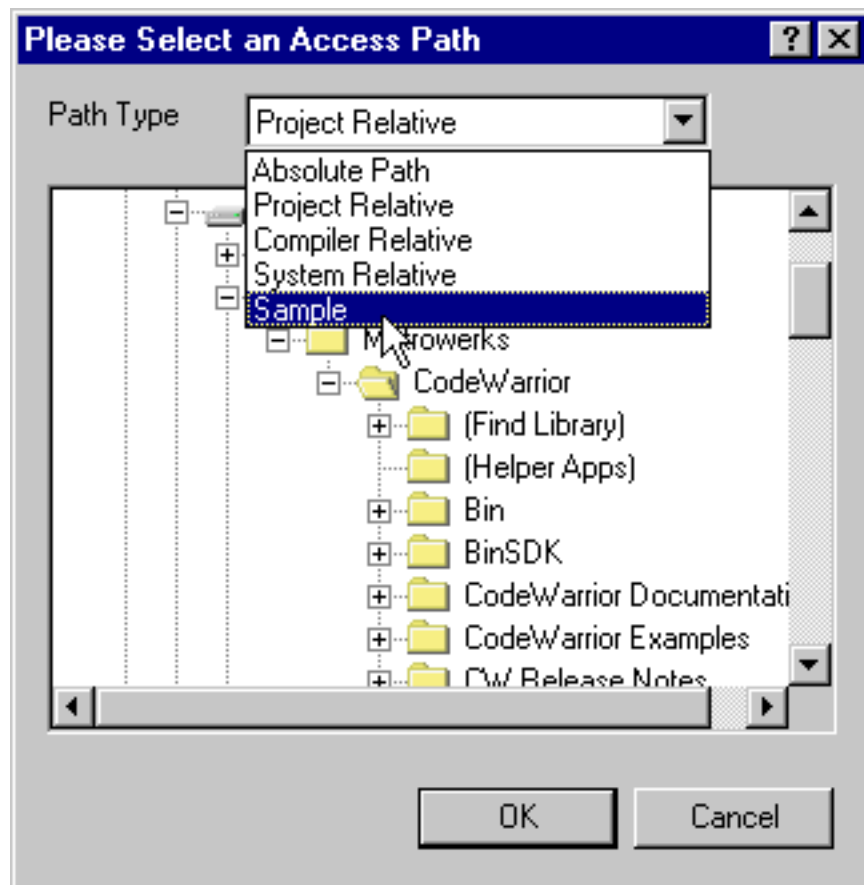
When you finish adding source trees, click the **Save** button in the IDE Preferences window to save your changes. The source trees are then displayed in the path pop-up menus for your project. For example, if you create a source tree named **Sample**, you can use it to create an access path, as shown in [Figure 9.17](#). You can also define your project's target output in terms of the source trees by using the [Output Directory](#) field in the [Target Settings](#) panel. For more information about this settings panel, refer to "[Target Settings](#)" on [page 327](#).

Change

To change a source tree, first select it in the [Source Trees list](#). Next, modify the [Name](#) and [Type](#) fields as desired. Then click the **Change** button. When you finish modifying the source trees, click the **Save** button in the IDE Preferences window to save your changes.

NOTE After changing a source tree, you might need to modify your project so that you do not refer to the source tree in its original form. Therefore, the IDE displays a message reminding you to update your project.

Figure 9.17 Choosing source trees in access path pop-up menus



If your project's files or libraries are not in the source trees, the IDE will not find them when compiling, linking, or running your project. Therefore, after you change an existing source tree, the IDE displays a message reminding you to update your project.

Remove

To remove a source tree, first select it in the Source Trees panel. Then, click the **Remove** button, and the source tree is removed.

When you finish removing source trees, click the **Save** button in the IDE Preferences window to save your changes.

NOTE After removing a source tree, you might need to modify your project so that you do not refer to the removed source tree. Therefore, the IDE displays a message reminding you to update your project.

Code Generation Configurations

The following panels customize the way the IDE generates code:

- [Global Optimizations](#)

Global Optimizations

The Global Optimizations panel, shown in [Figure 9.18](#), is available when for many build targets. Use this panel to configure how the compiler rearranges its object code to produce smaller and faster-executing object code. Some optimizations remove redundant operations in a program, while other optimizations analyze an item's use in a program. The goal of these optimizations is to improve your program's performance.

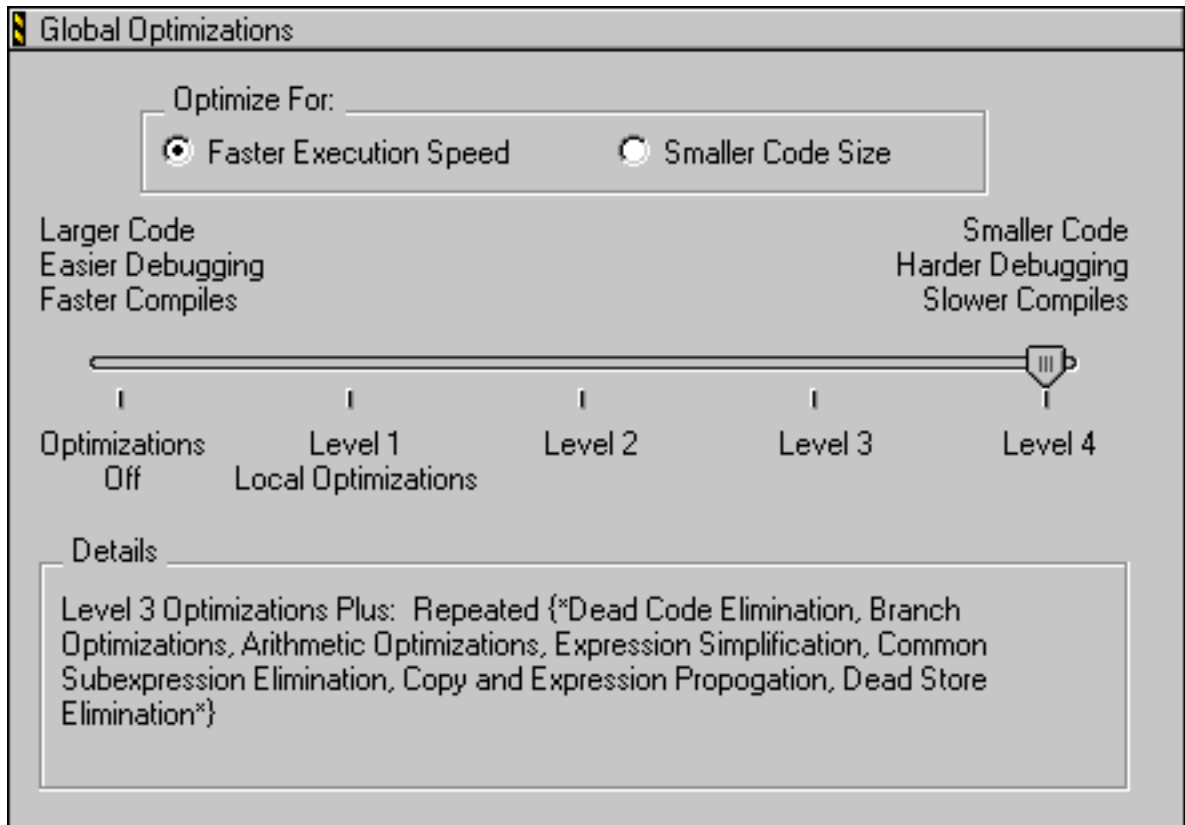
To learn how to open the Target Settings window and select the Global Optimizations panel, see [“Target Settings Guided Tour” on page 321](#).

All optimizations rearrange object code without affecting the object code's logical sequence of execution. In other words, an unoptimized program and its optimized counterpart produce the same results.

NOTE Use compiler optimizations only after you finish debugging your program. Using a debugger on a program that has been optimized might affect the debugger's ability to view source code.

To find specific information on how these settings apply to a build target, see [“Targeting Documentation” on page 27](#).

Figure 9.18 Global Optimizations settings panel



Optimize For

Use these options to configure how the CodeWarrior IDE optimizes your code:

Faster Execution Speed This option improves the execution speed of object code. Object code is faster, but can be larger.

Smaller Code Size This option reduces the size of object code produced by the compiler. Object code is smaller, but can be slower.

Optimization Level Slider

Use the slider to determine the optimizations that are applied to your code. You can choose to disable code optimizations entirely, or you can choose to apply one of four levels of optimizations. The higher the level you select, the more optimizations the IDE applies to your code.

Details

The **Details** field, below the [Optimization Level Slider](#), lists the optimizations applied to the project. The following list describes each of the optimizations:

- **Global Register Allocation**—Working values of heavily used variables are stored in registers instead of memory. This option also appears as **Global Register Allocation Only for Temporary Values** when you set the [Optimization Level Slider](#) to “Optimizations Off.”
- **Dead Code Elimination**—Removes statements that, logically, can never be executed or are never referred to by other statements. With this option enabled, object code is smaller.
- **Branch Optimizations**—The optimizer merges and restructures portions of the intermediate code translation. These optimizations eliminate some branch instructions. With this option enabled, the object-code size is reduced and the execution speed improves.
- **Arithmetic Optimizations**—Intensive computational instructions are replaced with faster equivalent instructions that produce the same result. For example, a multiply instruction can replace several add instructions. Also, several multiply instructions can replace a library call for raising a variable to a specified power. These optimizations result in faster object code.
- **Expression Simplification**—These optimizations find equivalent and simplified arithmetic expressions. For example, a replaces $1 * a$, $a + b$ replaces $a - (-b)$, $x \& y$ replaces $x \& (x \& y)$, and x replaces $x \& (x | y)$.
- **Common Subexpression Elimination**—Replaces similar redundant expressions with a single expression. For example, if two consecutive statements both use the expression $a * b * c + 10$, the compiler generates object code that computes the expression only once and applies the resulting value to both statements. With this option enabled, object code is smaller and faster.
- **Copy Propagation**—Replaces multiple occurrences of one variable with a single occurrence. With this option enabled, object code is smaller and faster. This option also appears as **Copy and Expression Propagation** for some build targets.
- **Peephole Optimization**—Applies local optimizations to small sections of your code. With this option enabled, the optimized sections of code are faster.

- **Dead Store Elimination**—Removes assignments to a variable if the variable is not used before being reassigned again. With this option enabled, object code is smaller and faster.
- **Live Range Splitting**—This optimization handles a local variables that is used in two completely separate contexts in a program. For example, note the two disjoint uses of the variable `i` in [Listing 9.1](#). The second use of the variable `i` can be replaced by a different compiler temporary variable. Live range splitting reduces the lifetimes of variables so that they can be optimally allocated. Long live ranges cause unnecessary spilling in registers.

Listing 9.1 Live Range Splitting example

```
int a[100], b[100], c[100], d[100], e[100], f[100];
foo(int n)
{
    int i;

    for(i=0; i<n; i++)
    {
        a[i] = b[i] + c[i];
    }

    for(i=0; i<n; i++)
    {
        d[i] = e[i] + f[i]
    }
}
```

- **Loop-Invariant Code Motion**—Moves computations that do not change on the inside of a loop to the outside of the loop to improve the loop's speed. With this option enabled, object code is faster.
- **Strength Reduction**—Replaces multiplication instructions that are inside loops with addition instructions to speed up the loop. With this option enabled, object code is larger but executes faster.
- **Loop Transformations**—Reorganizes the translated code for a loop in order to reduce the code overhead of setting up the loop

or testing for loop completion. With this option enabled, object code executes faster.

- **Loop Unrolling**—This optimization duplicates several times the code inside the looping structure. This duplication spreads over more operations the overhead of testing for loop completion and branching back to the beginning of the loop. This option improves execution speed, but the object code is larger.
- **Vectorization**—For processors that support vector optimizations, computations with arrays using code loops are translated into the appropriate vector instructions. This option accelerates object code. Note that some code loops cannot be translated into vector instructions.
- **Lifetime Based Register Allocation**—Uses the same processor register for different variables in the same routine if the variables are not used in the same statement. With this option enabled, object code executes faster.
- **Instruction Scheduling**—Rearranges a program's sequence of instructions to reduce conflicts with the use of registers and processor resources. This option improves the execution speed of object code.
- **Repeated**—The optimizations listed between the { * and * } braces in the [Details](#) field are repeated. With iteration, later phases of optimizations can provide more opportunities for optimization.

Editor Configurations

The following panels apply to the CodeWarrior Editor:

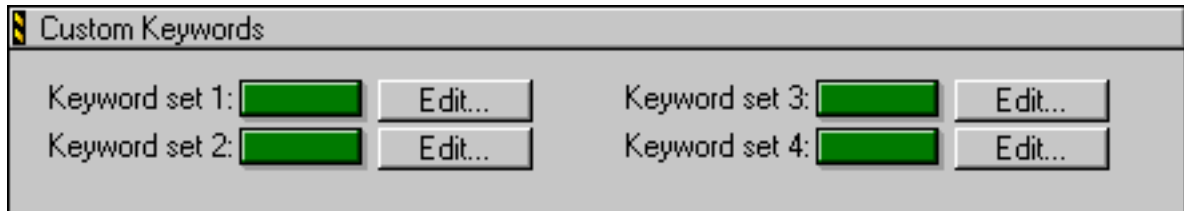
- [Custom Keywords](#)

Custom Keywords

The Custom Keywords settings panel, shown in [Figure 9.19](#), lets you define your own keyword sets to associate with certain colors when they appear in your editor files. These keywords are project-specific, not global to the CodeWarrior IDE.

To learn how to open the Target Settings window and select the Custom Keywords panel, see [“Target Settings Guided Tour” on page 321](#).

Figure 9.19 Custom Keywords settings panel



For information on defining keyword sets, see [“Syntax Coloring” on page 279](#). That topic also discusses the details of setting colors, specifying keywords, and importing and exporting keywords.

To change the color for a keyword set, click its associated color sample. To change the contents of a keyword set, click the **Edit** button to display a dialog box. Change the entries in the dialog box as desired.

Debugger Configurations

The following panels apply to the CodeWarrior debugger:

- [Debugger Settings](#)

Debugger Settings

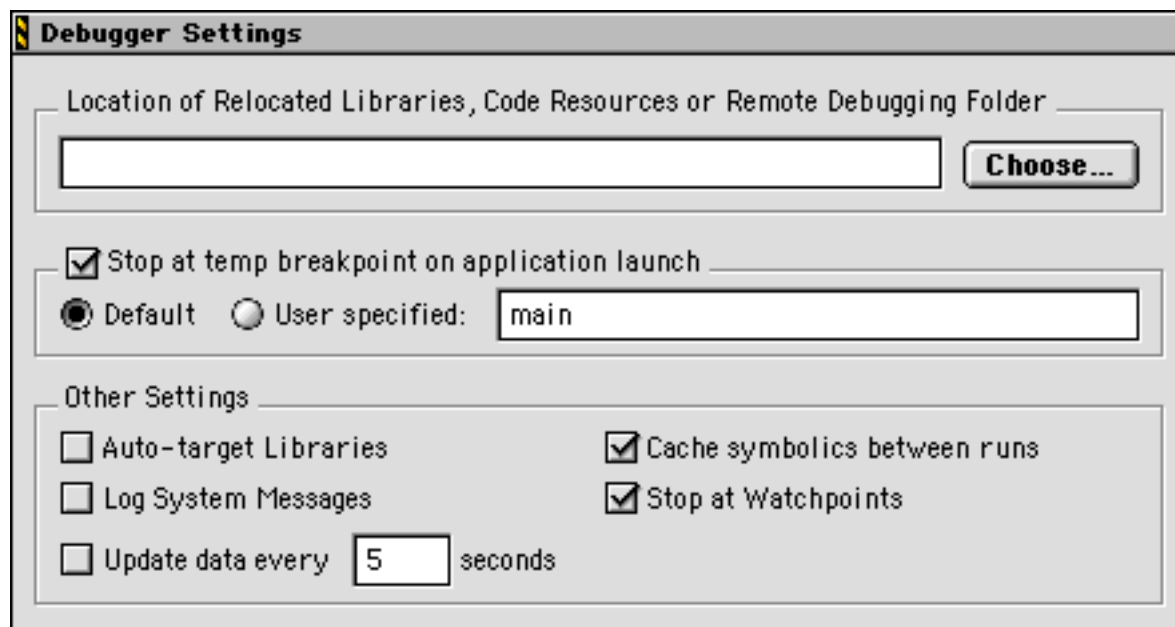
The Debugger Settings panel, shown in [Figure 9.20](#), includes options to log activities, change data-update intervals, and set other related options.

To learn how to open the Target Settings window and select the Debugger Settings panel, see [“Target Settings Guided Tour” on page 321](#).

Location of Relocated Libraries, Code Resources or Remote Debugging Folder

Use this field to specify the location of the relocated libraries, code resources, or other non-executable files required by the current project. You can also use the field to specify the remote debugging folder used by the project. Enter the path directly into the field. Alternatively, click **Choose** to display a standard dialog box. Use the dialog box controls to select the required files.

Figure 9.20 Debugger Settings panel



Stop at temp breakpoint on application launch

Enabling this checkbox causes the debugger to stop program execution at a specified temporary breakpoint at the beginning of the debugging session. Click the **Default** radio button to always stop at the `main()` function, or click the **User specified** radio button and enter in the field to the right the name of the function at which you want to stop.

Auto-target Libraries

This checkbox applies to the current project when you use the debugger to debug a file that is not a project file. For example, this situation can occur when you attach a running process. For more information about attaching processes, see [“Processes window toolbar” on page 418](#)

Enable the **Auto Target Libraries** checkbox to let the IDE attempt to debug dynamically linked libraries (DLLs) that are loaded by the target application. The IDE attempts to automatically debug the loaded DLLs for which symbolics information is available.

NOTE You might notice slower performance when you enable the **Auto Target Libraries** checkbox. If you encounter this problem, try disabling the checkbox to improve the IDE's performance.

Log System Messages

Enable this checkbox to log all system messages to a file. If you do not wish to create a log file, disable the checkbox. For more information about viewing log files, see [“Log Window” on page 434](#).

NOTE In Windows, log information includes messages about the loading and unloading of DLLs, as well as debug `printf()` messages. In the Mac OS, log information refers to messages about the loading of PowerPC code fragments, as well as `DebugStr()` messages.

Cache symbolics between runs

Enable this checkbox to let the debugger cache the symbolics information for a project and refer to the cached information during subsequent debugging sessions. Disable the checkbox to force the debugger to discard the symbolics information for the project after killing each debugging session. Enabling the symbolics cache is most useful for improving the performance of successive debugging sessions.

Stop at Watchpoints

Enable this checkbox to halt a program's execution when the debugger encounters a watchpoint, regardless of whether the watched value changes. Disable this checkbox to halt execution only when the watched value changes.

Update Data

Enable this checkbox to update the information in debugging windows according to a specified time interval. Enter an interval in the **Update data every x seconds** field, where *x* represents the number of seconds you wish to elapse before the next update. If you do not wish to update the debugging information, disable the checkbox. In this case, the information in debugging windows stays the same throughout the debugging session.

Compiling and Linking



This chapter discusses how to compile and link a project to produce a final executable file, and how to correct common compiler and linker errors using the Message window. The information in this chapter assumes you have already created a project, added the necessary files, grouped these files, and set the project's options. To learn more about these tasks, refer to other chapters in this book, including [“Working with Projects” on page 41](#), [“Working with Files” on page 109](#), [“Editing Source Code” on page 137](#), and [“Configuring IDE Options” on page 253](#).

You should also be familiar with the columns and pop-up menus in the Project window. To learn more about these features, refer to [“Working with Projects” on page 41](#).

This chapter does not describe in detail the various types of programs the CodeWarrior IDE can create. For that information, please see the *Targeting* manual appropriate for your platform. A table describing some of these manuals is shown in [“Targeting Manuals for various platform targets” on page 28](#).

The CodeWarrior IDE can only compile and link files belonging to an open project. In other words, you should have a project open before trying to compile any files.

The topics in this chapter are:

- [Choosing a Compiler](#)
- [Compiling and Linking a Project](#)
- [Using Precompiled or Preprocessed Headers](#)
- [Preprocessing Source Code](#)
- [Disassembling Source Code](#)
- [Guided Tour of the Message Window](#)
- [Using the Message Window](#)

- [Special Library Options \(Mac OS\)](#)

Choosing a Compiler

When you create source-code files, you use a certain programming language, such as C, C++, or Pascal. These languages have naming conventions for the extensions you add to file names. For example, in the C language, a source code file ends with a “.c” extension and a header file ends with a “.h” extension.

This section describes how to correctly associate a file extension with a compiler for a particular language in the CodeWarrior IDE.

Understanding Plug-in Compilers

The CodeWarrior IDE allows compilation of many different programming languages. The IDE takes a modular approach to compiling code by using various plug-in compilers.

Plug-ins are basically small, loadable code modules that allow the IDE to have many different compilers at its disposal. For example, there are plug-in compilers for C, C++, Pascal, Java, and assembly language.

Plug-in compilers usually have default build-target settings to help the CodeWarrior IDE decide which project files a plug-in handles. During regular compile and link operations, the IDE assigns files to the proper plug-in automatically.

Setting a File Extension

To associate a plug-in compiler with a given file, you must configure the options in the [File Mappings](#) settings panel. For a description of how to configure these options to assign a compiler to your source code files, refer to [“File Mappings” on page 341](#).

Compiling and Linking a Project

The CodeWarrior IDE provides many different ways to build a project. When you build a project, you compile it into object code and then link the object code into a final executable file.

All compiling and linking commands are available from the [Project Menu](#). Depending on your project type, some of these commands might be disabled or have different names. For example, you cannot [Run](#) a shared library, but you can [Make](#) it. Also, a command for compiling or linking might be dimmed because CodeWarrior is busy executing another command, or the project is being debugged.

The CodeWarrior IDE can only compile and link files belonging to an open project. That is, you should have a project open before trying to compile any files.

If you have multiple projects open at the same time, you might want to learn about selecting a default project before compiling and linking. For more information, refer to [“Choosing a Default Project” on page 74](#).

The topics in this section are:

- [Compiling Files](#)
- [Setting Link Order](#)
- [Updating a Project](#)
- [Making a Project](#)
- [Enabling Debugging](#)
- [Running a Project](#)
- [Debugging a Project](#)
- [Generating a Link Map](#)
- [Synchronizing Modification Dates](#)
- [Removing Object Code](#)
- [Advanced Compile Options](#)

Compiling Files

You can instruct CodeWarrior to compile a single file or to compile only certain files in your project. Of course, CodeWarrior can also compile all of the files in your project.

You might want to switch between multiple build targets in a project when compiling files. To learn more, refer to [“Setting the Current Build Target” on page 96](#).

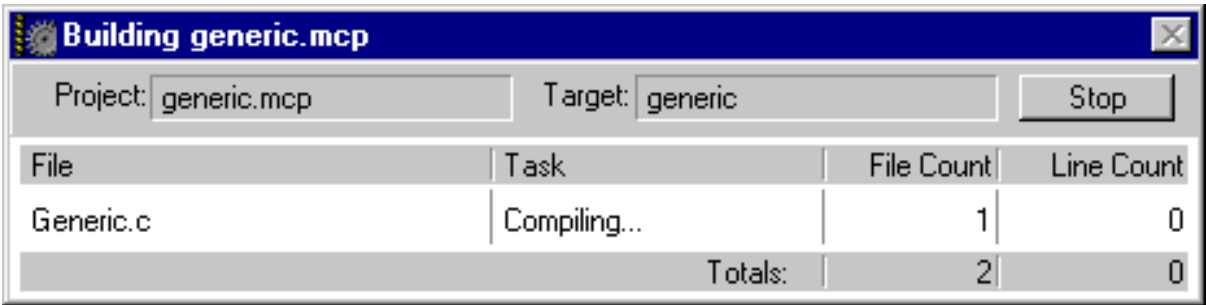
The **Compile** command in the [Project Menu](#) is grayed out in the following situations:

- There is no open project.
- The active editor window does not have a source-code file name extension.
- The active window’s source code file is not included in your project.
- (Windows) The binary file, such as the application you created from the project, is running under the debugger.
- (Windows) An application created from the project is running.

As CodeWarrior compiles source-code files and libraries in the open project, it highlights them in the Project window. The Build Progress window ([Figure 10.1](#)) displays a line count and the name of the file being compiled.

A status line displaying the total number of files to be compiled and the number of files being compiled is also provided at the bottom of the Project window.

Figure 10.1 Build Progress window



Compiling One File

To compile a single file in your project, select that file in the Project window and choose **Compile** from the [Project Menu](#).

Alternatively, you can open the file in the CodeWarrior editor, make the editor window the active window, and choose **Compile** from the [Project Menu](#).

Compiling Selected Files

You can compile several files in your project by selecting those files in the Project window and then choosing **Compile** from the [Project Menu](#). To learn how to select several files in your project, refer to [“Selecting Files and Groups” on page 77](#).

Recompiling Files

CodeWarrior does not always recognize file changes and might not automatically recompile the file. You can force CodeWarrior to recompile a changed file by “touching” that file. To learn how to touch a file, refer to [“Touching and Untouching Files” on page 89](#).

After touching the file or files that you want to recompile, choose **Bring Up To Date** or **Make** from the [Project Menu](#).

Setting Link Order

You can specify the order in which the IDE compiles files by using the [Link Order View](#) of the Project window. By re-arranging the order of the files, you can prevent linkage errors caused by file dependencies. In other words, if file Alpha depends on the completed compilation of file Zeta, you can use change the link order and position file Zeta to compile before file Alpha, thus preventing an error message.

To set the link order in the current project:

1. **Click the Link Order tab in the Project window.**

The Link Order view appears in the Project window.

2. **Sort the files into the desired link order.**

Drag and drop each file into the desired build order.

The next time you [Bring Up To Date](#), [Make](#), [Run](#), or [Debug](#) the project, the IDE uses the new build order.

See [“Link Order View” on page 51](#) for additional information.

NOTE (Mac OS) If you are building a 68K project, the Link Order tab is named the Segments tab.

Updating a Project

When you have many newly-added, modified, or touched files in your project, you can use the [Bring Up To Date](#) command in the [Project Menu](#) to compile all of the files.

When using this command, the linker is not invoked, so the compilation does not produce an output binary of your project. The [Bring Up To Date](#) command only runs the compiler.

You might want to switch between multiple build targets when updating a project. To learn how to do this, refer to [“Setting the Current Build Target” on page 96](#).

Making a Project

When you are ready to produce a binary file, such as an application, library, or shared library, use the [Make](#) command in the [Project Menu](#). This command builds the selected project type by updating the newly-added, modified, and touched files, and then linking the project into a final executable file.

The results of a successful build depend on the selected project type. For example, if the project type is an application, the [Make](#) command builds an application and saves that application in the same folder as your project. [Table 10.1](#) lists some sample project types and the resulting files after issuing a [Make](#) command. To find a full list of the types of software products you can produce, refer to the *Targeting* manual for your build target. Refer to [Table 1.1 on page 28](#) to determine which manual applies to various build targets.

Table 10.1 **Sample files produced for certain project types**

Project Type	Build Target	Make Command Creates...
Application	Win32	Win32 Application
Library	68K/PowerPC	Mac OS A4 or A5 Library
Dynamic link library	Solaris	Solaris Shared Library (.o)

Once all the modified files and touched files have been compiled successfully, CodeWarrior links all the files in the project to produce your output binary. If the project has already been compiled using [Bring Up To Date](#) or another command, then the **Make** command only links the compiled source code files together.

You might want to switch between multiple build targets in a project when making that project. Refer to [“Setting the Current Build Target” on page 96](#) for more information.

Enabling Debugging

When you choose [Enable Debugger](#) command from the [Project Menu](#), the **Debug** command runs the CodeWarrior debugger in order to debug your project. When you choose [Disable Debugger](#) from the [Project Menu](#), the **Run** command runs your project normally.

To learn how to generate debugging information for all of the files in the project, refer to [“Controlling Debugging in a Project” on page 104](#).

To learn more about running your project, refer to the following section, entitled [“Running a Project.”](#)

Running a Project

When you choose the **Run** command from the [Project Menu](#), CodeWarrior compiles and links the project (if necessary), creates a stand-alone application, and then launches that application.

If the current project is not an executable file, the **Run** command is not available. Non-executable projects include libraries, shared libraries, dynamic linked libraries (DLLs), code resources, and tools.

When compiling and linking is successful, CodeWarrior saves the new application to your hard disk. The new application is named according to options you set in the Target Settings window. If you would like to change these options, refer to [“Choosing Target Settings” on page 324.](#)

If the current project is designed to operate on another platform, you must connect your host computer to the target computer or device before choosing the **Run** command. For additional information, see the appropriate *Targeting* manual for the target platform.

Debugging a Project

To debug your project, there are basically two steps you need to do. Of course, you must already have your project compiled and linked with debugging information generated. To learn how to enable debugging for your project, refer to [“Enabling Debugging” on page 367.](#)

The second step is for you to launch the debugger with your compiled application as the debug target. You can do this by choosing the **Debug** command from the [Project Menu](#). If the **Debug** command is not visible in the menu, your project is not currently configured for debugging. Refer to [“Enabling Debugging” on page 367](#) to learn how to remedy this problem.

After you choose the **Debug** command, CodeWarrior compiles and links your project, creates a debugging information file, and then opens that information file with the debugger.

If the **Debug** command is grayed out, make sure to configure the proper options for debugging, as detailed in [“Enabling Debugging” on page 367.](#) If the **Debug** command is still grayed out, you are probably attempting to run a project whose project type cannot be run (such as a shared library or dynamic link library), or the application is already running.

NOTE The [Debug](#) command does not open any application that you might need to debug your project. For example, if you are debugging a Photoshop plug-in or any other project that requires an application, you must launch the application before you can debug the project.

If the current project is designed to operate on another platform, you must connect your host computer to the target computer or device before choosing Run. For additional information, see the appropriate *Targeting* manual for the target platform.

Generating a Link Map

The CodeWarrior C/C++ compilers let you create a link-map file that contains function and class-section information about the generated object code.

CodeWarrior's Pascal compilers let you create a make-map file that contains a list of dependencies and the compilation order.

The settings to control this option are in the build target's Linker panels. After configuring these panels as desired, you must [Make](#) your project. If the compile and link operations are successful, the IDE generates a link-map file, names the file after your project, adds a .MAP extension, and saves the link-map file in your project folder.

To learn more about a build target's Linker panels, see the discussion in the appropriate *Targeting* manual.

Synchronizing Modification Dates

If you want to update the modification dates of all the files stored in the project, choose the [Synchronize Modification Dates](#) command from the [Project Menu](#).

To learn more about this topic, see [“Synchronizing modification dates” on page 90](#).

Removing Object Code

When you compile your project, the CodeWarrior IDE adds the object code from each source file to the project. This binary object

code increases the size of the project file. You can choose commands that affect this object code if you want the project file to consume less memory on your hard disk, or if you want to remove all object code and start compilation over again.

Removing Object Code

In some cases, you might wish to remove all of the object code from the project and restart the compiling and linking process. To remove a project's object code, select the [Remove Object Code](#) command from the [Project Menu](#). The dialog box shown in [Figure 10.2](#) appears.

Figure 10.2 Remove Objects dialog box



Clicking **All Targets** removes all object code data for all build targets in the project, and resets the Code and Data size of each file in the Project window to zero. Clicking **Current Target** removes the object code for the current build target only, and leaves the object code in place for all other build targets. Clicking **Cancel** aborts the operation, so none of your object code is removed.

To learn how to change the current build target of your project, refer to ["Setting the Current Build Target" on page 96](#).

Remove Objects & Compact command

This command removes all binaries from the project and compacts the project to consume the minimum amount of space on your hard disk. The procedure is similar to that discussed in ["Removing Object Code" on page 370](#), but in addition the project file is compressed.

Compacting the project removes all object code and debugging information stored in the project file, and retains only the information about project-specific option settings and the files that belong to the project.

For more information about this command, refer to [“Remove Object Code & Compact” on page 635](#).

Advanced Compile Options

This section describes two options that speed up your project-build times or alert you when a build is completed.

Alerting Yourself After a Build (Mac OS)

You can start a project compile/link cycle in CodeWarrior, then switch to another application running on your machine. To learn how to receive notification when CodeWarrior completes the build process, refer to [“Play sound after ‘Bring Up To Date’ & ‘Make’ \(Mac OS\)” on page 258](#).

Speeding Up a Build by Avoiding Date Checks

To learn how to optimize the speed of your builds in CodeWarrior, refer to [“Use modification date caching” on page 337](#).

Using Precompiled or Preprocessed Headers

Source code files in a project typically use many header files (“`.h`” or “`.hpp`” files). Often, several different source-code files include the same header file, forcing the compiler to (inefficiently) read the same header file many times during the compilation process. Many programming languages support precompiled headers, including C and C++.

To shorten the time spent compiling and recompiling a header file, use the [Precompile](#) command in the [Project Menu](#). A precompiled header file takes the compiler significantly less time to process than an ordinary, uncompiled header file.

For instance, a header file that contains the most frequently used headers in your project could be made into one precompiled header

file. Instead of having to compile the same thousands of lines of header files for each source file in your project, the compiler only has to load one precompiled header file.

NOTE You can only include one precompiled header in a source file. Including more than one precompiled header results in an error.

TIP The precompiled header format frequently changes in order to accommodate new features in CodeWarrior updates. Therefore, precompiled header formats are often incompatible between CodeWarrior updates. After installing a new CodeWarrior update, you usually need to update your precompiled headers to use the new format. However, you can also configure your precompiled headers to automatically update to the new format. See [“Automatic updating” on page 373](#) for more information.

The topics in this section are:

- [Creating Precompiled Headers](#)
- [Defining Symbols For C/C++](#)
- [Defining Symbols For Pascal](#)

Creating Precompiled Headers

To create a precompiled header file, you must first open a project. The settings from this project are used when precompiling. A file to be precompiled does not have to be a header file (“.h” or “.hpp”), but it must meet these requirements:

- The file must be a text file. You cannot precompile libraries or other files.
- The file must be saved with a recognized extension, such as “.pch” or “.pch++,” at the end of its file name.
- The file does not have to be in a project, although a project must be open to precompile.
- The file must not contain any statements that generate data or code. However, C++ source code can contain inline functions and constant variable declarations (const).

- Precompiled header files for different build targets are not interchangeable. For example, to generate a precompiled header for use with Win32 compilers, you must use a Win32 compiler.
- A source file can include only one precompiled header file using the `#include` directive.

To create a precompiled header file, create a text file using [New Text File](#) in the [File Menu](#). In that text file, place your `#include` directives. For example, if you want to create a precompiled header file of the files `string.h` and `stdio.h`, place the following in your text file:

```
#include <stdio.h>
#include <string.h>
```

After you save the text file, the IDE can use that text file to create a precompiled header. You can specify the name of the precompiled header at the time of precompilation. You can also specify the name by including the following line at the beginning of your text file, where *name* is the name you wish to call the precompiled header:

```
#pragma precompile_target "name"
```

Windows Append the extension `".mch"` to *name*. This extension is required for precompiled headers.

Precompile command

To precompile a saved text file, choose [Precompile](#) from the [Project Menu](#). This command precompiles the text file in the active window, creating a precompiled header file. If compiler errors are detected, a Message window appears.

To learn more about the Message window and correcting compiler errors, consult ["Correcting Compiler Errors and Warnings" on page 385](#).

To learn more about automatic updating of precompiled headers, see the next section, ["Automatic updating."](#)

Automatic updating

If the source code has been modified, the CodeWarrior IDE automatically updates a precompiled header during a [Make](#) or [Bring Up To Date](#) operation.

If CodeWarrior encounters a “.pch” or “.pch++” file that was modified since it was last precompiled, the IDE precompiles it again to ensure that the resulting precompiled header is up-to-date.

To create a precompiled header file that is automatically updated, open the project that uses the precompiled header. Then, create a text file that will be used to create the precompiled header.

To read about the requirements for a precompiled header text file, refer to [“Creating Precompiled Headers” on page 372](#).

In the first line of the text file, add the line:

```
#pragma precompile_target "name"
```

This `pragma` tells the compiler to create a precompiled header with the file name of *name*.

Windows Append the extension “.mch” to *name*. This extension is required for precompiled headers.

When you finish the text file, save it with a valid extension in the same folder as your project.

Windows Save the text file with a “.pch” extension if it will be used to create a precompiled header in either C or C++.

Mac OS Save the text file with a “.pch” extension if it will be used to create a precompiled header in C. Use the “.pch++” extension if the text file will be used to create a precompiled header in C++.

Next, choose [Precompile](#) from the [Project Menu](#).

Now add the text file to the open project using the [Add Window](#) command in the [Project Menu](#).

Whenever the precompiled header file is modified, the CodeWarrior project manager automatically updates the header file by precompiling it.

To include the precompiled header in a project source-code file, add this line as the first `#include` directive in the source-code file:

```
#include "name"
```

Alternatively, you can specify a precompiled header as a prefix file, using the settings for your build target. To learn how to do this, refer to the discussion of the C/C++ Language settings panel in the *C Compiler Guide*.

NOTE Do not use the precompiled header text file (".pch" or ".pch++") in `#include` directives; use the name of the resulting precompiled header file instead (".mch"). Although using the precompiled header text file is legal and does not affect the final binary, you are not taking advantage of the precompiled header's speed.

Defining Symbols For C/C++

To automatically update and add predefined symbols and other preprocessor directives, you can create a precompiled header file, add it to your project, and include it in the prefix file specified in the C/C++ Language settings panel.

- 1. Create a new text file.**

Open your project and create a new text file with the [New Text File](#) command on the [File Menu](#).

This new text file will contain your preprocessor directives. You will use this file as a precompiled header file that you will add to your project.

- 2. Open the C/C++ Compiler settings panel.**

Choose [Target Settings](#) from the [Edit Menu](#). The actual name of the Target Settings command includes the name of the current build target. Select the **C/C++ Language** settings panel in the Target Settings window.

3. Get the Prefix File name if it exists.

If there is a file name in the **Prefix File** field, [Copy](#) it, then close the [Target Settings](#) window.

In the new text file window, paste the file name that you copied from the **Prefix File** field into an `#include` directive. Make sure this is the *first* directive in the text file.

For example, if the prefix file is `MyHeaders`, then the first directive in the text-file window is:

```
#include <MyHeaders>
```

4. Add the `#pragma` statement.

Add the `#pragma precompile_target` statement to the text file. This statement lets you name the precompiled file. For example, to create a file named `MyPrecomp`, use the following statement:

```
#pragma precompile_target "MyPrecomp"
```

Windows Append the extension `".mch"` to the desired file name. This extension is required for precompiled headers. In this example, `"MyPrecomp.mch"` is the correct file name.

5. Type in all your own preprocessor directives.

Type in all your own `#define`, `#include`, and other preprocessor directives corresponding to the needs of your source code. The text file cannot contain any source code that generates data or executable code. However, C++ source code can contain inline functions and constant variable declarations (`const`).

6. [Save](#) the text file.

Choose [Add Window](#) from the [Project Menu](#) to add the text file to your project. Save the text file with a valid extension in the same folder as your project.

Windows Save the text file with a `".pch"` extension.

Mac OS Save the text file with a `".pch"` extension if it will be used to create a precompiled header in C. Use the `".pch++"` extension if the text file will be used to create a precompiled header in C++.

7. [Precompile](#) the text file.

Choose [Precompile](#) from the [Project Menu](#). The IDE uses the text file to create a precompiled header file.

8. Open the C/C++ Compiler settings panel.

Choose [Target Settings](#) from the [Edit Menu](#) and select the C/C++ Language settings panel.

9. Set the new Prefix File.

In the **Prefix File** field, enter the name of your precompiled file, or “MyPrecomp” in this example. Click **Save** to save your changes.

Windows In this example, you would enter “MyPrecomp.mch” in the **Prefix File** field.

Whenever CodeWarrior builds your project, the project manager updates your precompiled header and automatically includes it in each source code file.

Defining Symbols For Pascal

Although the Pascal preprocessor is not as powerful as the C/C++ preprocessor, you can still create files that can automatically insert your own preprocessor symbols and compiler directives into your project. For more information on the Pascal compiler directives, see the *Pascal Language Manual* on the CodeWarrior Reference CD.

1. Create a new text file.

Open your project and create a new text file with the [New Text File](#) command.

This new text file will contain your compiler directives.

2. Open the Pascal Compiler settings panel.

Choose [Target Settings](#) from the [Edit Menu](#). The actual name of the Target Settings command includes the name of the current build target. Select the Pascal Language settings panel in the Target Settings window.

3. **Get the Prefix File name if it exists.**

If there is a file name in the **Prefix File** field, [Copy](#) it, then close the Target Settings window.

In the new text file window, paste the file name that you copied from the **Prefix File** field into an include directive, `{ $I }`. Make sure this is the *first* directive in the file.

For example, if the prefix file is `OtherDefs.p`, then the first directive in the editor window is:

```
{ $I OtherDefs.p }
```

4. **Type in all your own `{ $SETC }`, `{ $I }`, and other preprocessor directives.**

The text file cannot contain any source code that generates data or executable code.

5. [Save](#) the text file.

Save the file as an ordinary Pascal file in the same folder as your project. For example, save the file with “`MyPrecomp.pas`” as its name.

6. [Precompile](#) the text file.

Choose [Precompile](#) from the [Project Menu](#). The IDE uses the text file to create a precompiled header file.

7. **Open the Pascal Compiler settings panel.**

Choose [Target Settings](#) from the [Edit Menu](#), and select the Pascal Language settings panel.

8. **Set the new Prefix File.**

In the **Prefix File** field, enter the name of your file, in this example “`MyPrecomp`”, then click **Save** to save your changes.

Whenever CodeWarrior builds your project, the project manager automatically includes the precompiled header in each source code file.

Preprocessing Source Code

The preprocessor prepares source code for the compiler. This preprocessor interprets directives beginning with the “`#`” and “`$`”

symbols (such as `#define`, `$pragma` and `#ifdef`), removes extra spaces and blank lines, and removes comments (such as `/*...*/` and `//`). You might want to preprocess a file if you want to see what the code looks like just before compilation.

Open a file that you want to preprocess, or select a file in your currently-open Project window. To preprocess a file, select the [Preprocess](#) command from the [Project Menu](#). The results of the [Preprocess](#) command are stored in a new file. This new file is named after the preprocessed source-code file and begins with the “#” character.

To save the contents of the new window, choose one of the save commands in the [File Menu](#).

Disassembling Source Code

If you want to see the code that would be generated for your file, you can disassemble the file. Disassembling is useful if you want to view the machine-level code that is being executed when your source code is executed. In addition, the disassembled code can be a model for writing your own assembly routines. Library files can also be examined using this command.

The [Disassemble](#) command in the [Project Menu](#) disassembles the compiled source-code file selected in the Project window and displays its assembly-language code in a new window. The title of the new window consists of the name of the source code file with the extension “`.dump`.”

To save the contents of the “`.dump`” window, choose one of the save commands in the [File Menu](#). If the file being disassembled has not been compiled, choosing the [Disassemble](#) command causes the IDE to compile the file before disassembling it.

Guided Tour of the Message Window

The Message window, shown in [Figure 10.3](#), is used to display messages about events that have occurred when compiling, linking, or searching files. There are a number of elements in the window

that are useful for accomplishing certain tasks, such as navigating to error locations and scrolling to see all messages for a project.

There are some user interface items in the Message window that are not discussed here. To learn about the [Marker Pop-Up Menu](#), [Options Pop-Up Menu](#), [Path Pop-Up Menu \(Mac OS\)](#), [File Path Caption](#) and [Line Number Button](#), refer to [“Guided Tour of the Editor Window” on page 137](#).

The topics in this section include:

- [Error Button](#)
- [Warning Button](#)
- [Project Information Caption](#)
- [Extra Information Button](#)
- [Stepping Buttons](#)
- [Message List Pane](#)
- [Source Code Disclosure Triangle](#)
- [Source Code Pane](#)
- [Pane Resize Bar](#)

Error Button



The Error Button in the Message window toggles the view of error messages on and off. This is useful if you have changed the view of the window to something else and want to get back to viewing the error messages.

To learn more about seeing error messages in the Message window, refer to [“Seeing Errors and Warnings” on page 383](#).

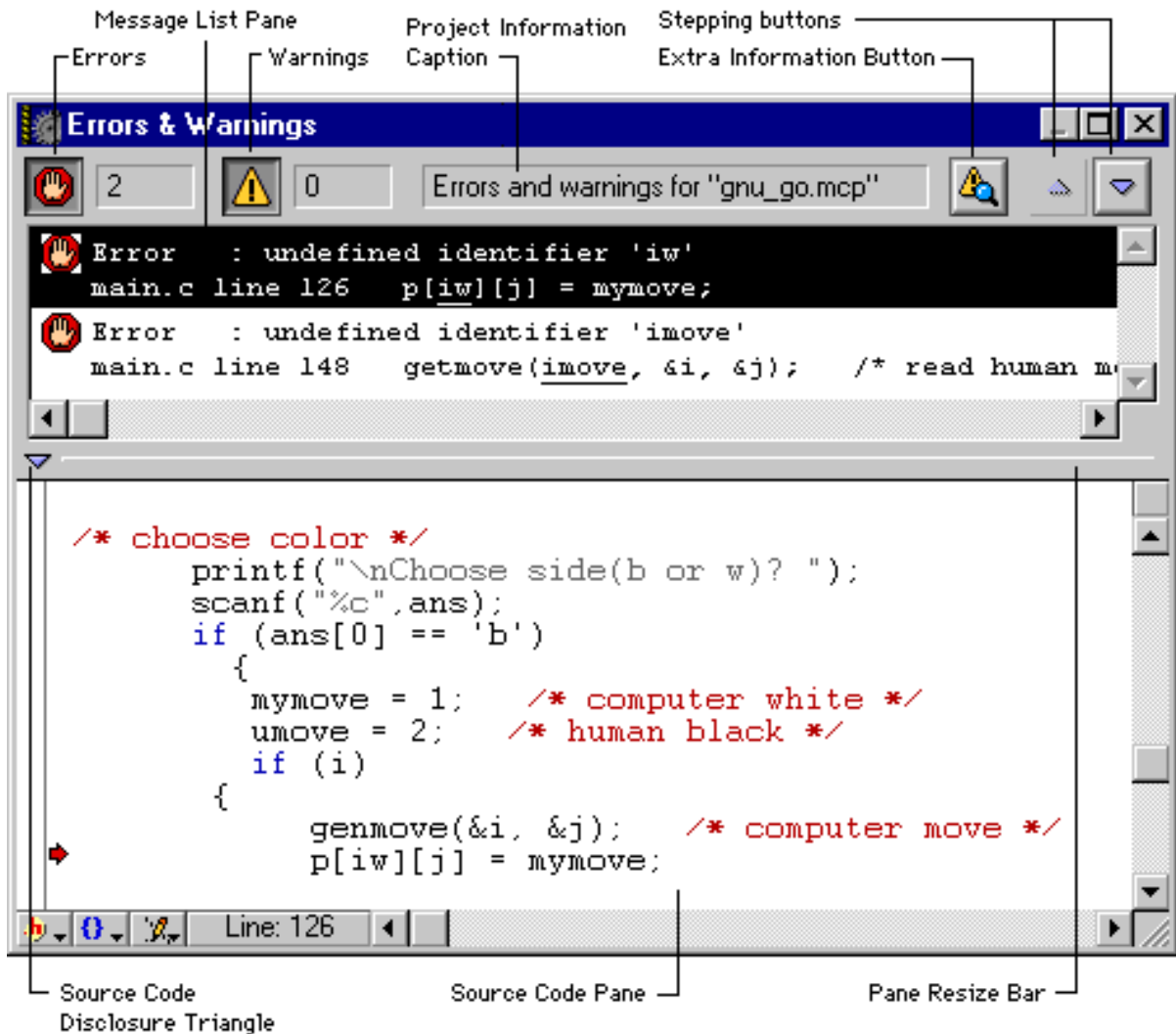
Warning Button



The Warning Button in the Message window toggles the view of warning messages on and off.

To learn more about seeing error messages in the Message window, refer to [“Seeing Errors and Warnings” on page 383](#).

Figure 10.3 The CodeWarrior Message Window



Project Information Caption

The Project Information Caption gives a short description of the view you are looking at in the Message window. Your project name will appear here.

Extra Information Button

The Extra Information Button expands a message to show information about the project, target, and file that caused the message.

Stepping Buttons

The Stepping Buttons allow you to step up or down through the messages in the window.

To learn more about stepping through messages in the Message window, refer to [“Stepping Through Messages” on page 384](#).

Message List Pane

The Message List Pane displays your messages.

To learn more about seeing messages in the Message window, refer to [“Seeing Errors and Warnings” on page 383](#).

Source Code Disclosure Triangle

The Source Code Disclosure Triangle allows you to hide the Source Code Pane of the Message window.

Source Code Pane

The Source Code Pane of the Message window allows you to view the source code at the location referred to by a message. To learn more about the view in this window, refer to [“Seeing Errors and Warnings” on page 383](#).

Pane Resize Bar

The Pane Resize Bar allows you to reallocate the amount of space in the Message window given to the Source Code Pane and Message List Pane. By clicking and dragging this bar up or down you will change the amount of space on your computer screen that is allocated to both panes.

Using the Message Window




While compiling your project, the CodeWarrior IDE may detect a syntax error or other type of compiler error in one of your project's source code files. If this happens, the Message window displays the total number of errors and warnings, and information about each one. See [“Guided Tour of the Message Window” on page 379](#) for information on the interface items in this window.

In this section, you will learn how to interpret, navigate, and use the information that appears in the Message window. The topics in this section include:

- [Seeing Errors and Warnings](#)
- [Stepping Through Messages](#)
- [Correcting Compiler Errors and Warnings](#)
- [Correcting Linker Errors](#)
- [Correcting Pascal Circular References](#)
- [Saving and Printing the Message Window](#)
- [Locating Errors in Modified Files](#)

Seeing Errors and Warnings

The Message window displays several types of messages:

Select this ...	To display this...
 Errors	Either compiler or linker errors. Both types of errors prevent the compiler and linker from creating a final binary.
 Warnings	Either compiler or linker warnings. Neither type prevents the CodeWarrior IDE from creating a binary. However, they indicate potential problems during run time. You can specify which conditions lead to warning messages or you can upgrade all warnings to errors.
 Notes	All other types of messages issued in the Message window. For example, results of a batch find are notes messages.

To close the Message window, click its close box or select [Close](#) in the [File Menu](#) while the Message window is the active window. If you close the message window and want to see it again, choose the [Errors & Warnings Window](#) command from the [Window Menu](#) to reopen it.

To see only error messages in the [Message List Pane](#), click on the [Error Button](#) and turn off the [Warning Button](#).

To see only warnings in the [Message List Pane](#), click the [Warning Button](#) and turn off the [Error Button](#).

To see both errors and warnings in the [Message List Pane](#), click both buttons. Notes do not appear in the Errors & Warnings window.

You'll also see other types of messages from time to time in a Message window, such as:

- During [Add Window](#) or [Add Files](#) when a file being added does not reside on an existing access path.
- During linking when a project contains conflicting resources.
- During a Find when the [Batch checkbox](#) is selected in the Find dialog box.
- (Mac OS) During a **Find Reference** when more than one definition for the same function is found (multiple definitions).
- (Mac OS) During a **Find Reference** on a C++ function that has been overridden, the Message window appears with a warning that there are two or more instances.

Stepping Through Messages

When the compiler finds errors during a build, or the CodeWarrior IDE search command finds text you asked it to look for when [Using Batch Searches](#), you will see the message window.

The window is divided into two panes:

- [Message List Pane](#), which lists the messages, or
- [Source Code Pane](#), which displays the source code for the selected message.

See [“Guided Tour of the Message Window” on page 379](#) for information on the interface items in this window.

To step through the list of messages, click the up or down [Stepping Buttons](#) or click the error message you are interested in.

To navigate the source code that is shown in the [Source Code Pane](#) for a given message, you use the [Interface Pop-Up Menu](#), [Routine](#)

[Pop-Up Menu](#), or the [Line Number Button](#). To learn about how to use these navigational features, refer to [“Guided Tour of the Editor Window” on page 137](#).

Correcting Compiler Errors and Warnings

When an error occurs during compilation, the Message window will show you the error message in the [Message List Pane](#). The location in the source code that the message refers to will be shown in the [Source Code Pane](#). You can navigate to the spot in your source code where the message refers to, and inspect or correct your code.

For a complete list of compiler errors and their possible causes, consult the *Error Reference* documentation on your CodeWarrior CD.

Correcting Errors in the Source Code Pane

To correct a compiler error or warning, you must first find the cause. First, make sure that the [Source Code Pane](#) of the Message window is visible. If it isn't visible, refer to [“Source Code Disclosure Triangle” on page 382](#) to learn how to make it visible.

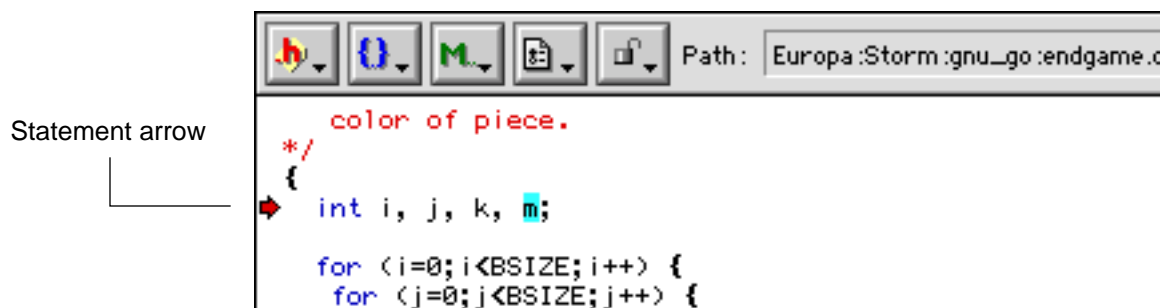
To view the statement that the compiler believes has caused the error or warning, select the message in the [Message List Pane](#) of the Message window. Notice that the [Source Code Pane](#) view now shows the source code that corresponds to the message. A statement arrow points to the line of code that the compiler reports as an error. The statement arrow is shown in [Figure 10.4](#). Using the statement arrow as a guide, you can then edit the erroneous line directly in the Source Code Pane view.

Use the [Interface Pop-Up Menu](#), [Routine Pop-Up Menu](#), or the [Line Number Button](#) in the [Source Code Pane](#) to navigate your code or open interface files. To learn about how to use these navigational features, refer to [“Guided Tour of the Editor Window” on page 137](#).

Opening the File for the Corresponding Message

To open a source code file that corresponds to a given message, select the message in the [Message List Pane](#) and press Enter/Return. You may also double-click the message in the [Message List Pane](#) to open the relevant file.

Figure 10.4 Statement arrow pointing to an error



Correcting Linker Errors

When your project is linked, any errors that may occur can be viewed and corrected.

Viewing Linker Errors

If the linker encounters any errors while linking your project, the Message window appears indicating these errors. This window can be scrolled through by using the scroll bar or [Stepping Buttons](#).

To learn about how to scroll through messages in the Message window, refer to [“Stepping Buttons” on page 382](#). To learn about changing the view of messages in the Message window, refer to [“Seeing Errors and Warnings” on page 383](#).

Since Linker errors are a result of problems in the object code, the CodeWarrior IDE cannot show their corresponding errors in the project’s source code files.

Why Linker Errors Occur

Linker errors are usually the result of one of the following circumstances:

- You have misspelled the name of a library routine. This means that the routine that the Linker is searching for does not exist. Check the name of the routine to make sure it is spelled correctly.
- Your compiled code generates a reference too large to be handled by the selected code model.

- Your project is missing the necessary libraries. Linker error messages of this type occur when the project is missing a library. To find out which libraries or shared libraries should be added to your project, refer to the *CodeWarrior Targeting* manual appropriate for your platform, as described in [Table 1.1 on page 28](#).

NOTE (Mac OS) To find the libraries you should be using, refer to the next section, [“\(Mac OS\) To find the libraries you should be using, refer to the next section, “\(Mac OS\) To find the libraries you should be using, refer to the next section, “\(Mac OS\) To find the libraries you should be using, refer to the next section, “\(Mac OS\) To find the libraries you should be using, refer to the next section, Finding Which Library to Use \(Mac OS\)”Finding Which Library to Use \(Mac OS\)”Finding Which Library to Use \(Mac OS\)”Finding Which Library to Use \(Mac OS\)”Finding Which Library to Use \(Mac OS\)”](#)

Often a linker error occurs when a library is missing from a project. As a result, one or more functions or identifiers cannot be found and the linker reports a problem. The difficulty you face is trying to figure out which library defines the function or identifier.

To find out which libraries or shared libraries should be added to your project, you can refer to the *CodeWarrior Targeting* manual appropriate for your target, as described in [Table 1.1 on page 28](#). In addition, you can use the **Find Library** tool included on the CodeWarrior Tools CD.

Find Library is a set of files with the internal names of functions and global identifiers from many of the libraries included with the CodeWarrior IDE for Mac OS and ANSI Libraries. Use these files as a file set with the CodeWarrior IDE Find dialog box. Search for the missing identifier in these files, and you can quickly find the right library to include in your project.

Here's how to use the Find Library tool.

- 1. Create a File Set**

Use the Find window to create a set that includes all the files in the Find Library folder on your CodeWarrior Tools CD. To learn how to create and use a file set, refer to [“Choosing Files to be Searched” on page 190](#) and [“Saving a File Set” on page 194](#).

2. Set the search parameters to ignore case.

Be sure to select the **Ignore Case** option when doing your search with the Find window. Some identifiers may be all uppercase or lowercase.

3. Search for the symbol that the linker can't find.

Enter the name of the unlinked symbol, and do a batch search of the Find Library file set. When the search has found a matching identifier, you only need to check the file name of the library to determine the name of the library to include in your CodeWarrior project. There is no path information given in the files, so you'll have to locate the correct library in the appropriate CodeWarrior subdirectory on your hard disk.

For example, if you do a batch search for the identifier `num2dec` in this file set, you'll get a match in the `MathLib` file. You now know that if you want to use the `num2dec` function, you must include the `MathLib` file in your project.

WARNING!

The files in the Find Library folder are just compilations of names that have been taken out of the libraries and put into a special naming file. These are *not* the actual library files. Do *not* include the 'Find Library' files in your projects.

NOTE

Some names may be duplicated in more than one file. If the name has a leading period, it is probably a declaration in that file, not a definition. Be sure to use Command-G or batch search to retrieve all files with matching identifiers. Then select the appropriate library for your particular project.

Some of the 68K and PowerPC shared libraries differ slightly, although the library names are the same. To identify the correct library, we've added the suffix `-68K` or `-PPC` as appropriate. For example, `DragLib` is listed in the Find Library as both `DragLib` `-68k` and `DragLib` `-PPC`. Ignore the suffix when looking for the library by name, but make sure you include the correct CPU version of the library for your target platform.

The CodeWarrior IDE will report the lines on which matches occur, but you can ignore those line numbers. The line information reported is irrelevant with respect to the actual library, and refers only to the line number in the special naming file.

There are several versions of the ANSI Libraries. They all contain the same identifiers, so only one version is included. You should assume that the appropriate library for your project will contain the same functionality.

The ANSI C++ Library file will be difficult to understand because of name mangling. However, most linker errors will list a function name as well as the identifier. Searching for that function name should give you good results. For example, you might get an error message saying “__aad__9bitstringFRC9bitstring in bitsand.c” is unknown. You could either search for the mangled name or you could search for `bitsand` to get a match.

Correcting Pascal Circular References

The CodeWarrior Pascal compiler’s [Make](#) and [Run](#) commands build your project by examining every Pascal file in your project file. As this examination is performed, a tree of dependencies is built for the interfaces of your units and for their implementations.

A circular reference occurs when a unit declares something that is used in another unit and that same unit declares something used by the former. To break this loop, the Pascal compiler does not allow such things among the interface parts of units, but it is permitted for implementations.

The example in [Listing 10.1](#) is perfectly valid, since both A’s and B’s interfaces depend on C’s, but are independent from one another. Knowing everything that was declared, A’s implementation depends on all interfaces, the same is true for B’s and C’s. For this example, the make utility will ask the compiler to compile [Listing 10.1](#) in the following order:

Listing 10.1 A valid example of circular referencing

UNIT A;	UNIT B;	UNIT C;
INTERFACE	INTERFACE	INTERFACE

Compiling and Linking

Using the Message Window

USES C;	USES C;	
TYPE	TYPE	TYPE
A_type = ...	B_type =	C_type = ...
IMPLEMENTATION	IMPLEMENTATION	IMPLEMENTATION
USES B;	USES A;	USES A, B;
....

1. **C's interface is compiled.**
2. **B's interface is compiled.**
3. **All of unit A is compiled (unit and implementation),**
4. **B's implementation is compiled.**
5. **C's implementation is compiled.**

After an interface compilation, the compiler writes a binary symbol table, containing all the declarations of the interface, in an 'sbmf' resource in the project file. This information is read back when the unit's name is encountered in a USES clause for another compilation. A unit is recompiled only when one of the following conditions occur:

- The source was modified,
- The source is currently open and edited, or,
- A unit on which the source depends was recompiled.

Saving and Printing the Message Window

To print or save the contents of the Message window, just follow these steps.

To Print the Message Window:

1. **Make the Message window active.**

To accomplish this, either click on the deactivated Message window, or select the [Errors & Warnings Window](#) command from the [Window Menu](#).

2. **Select the [Print](#) command from the [File Menu](#).**

If you choose the [Print](#) command on the [File Menu](#) to print the Message window, the print dialog box appears. Specify printing options and click **OK**. All the errors, warnings, and messages will be printed.

To learn more about printing, refer to the documentation that came with your printer.

To Save the Message Window:

1. **Make the Message window active.**

To accomplish this, either click on the deactivated Message window, or select the [Errors & Warnings Window](#) command from the [Window Menu](#).

2. **Select the [Save A Copy As](#) command from the [File Menu](#).**

The [Save A Copy As](#) command will display a standard file dialog box.

3. **Specify the name of the file and the location.**

A text file will be saved containing all the errors, warnings, and messages that are listed in the message window.

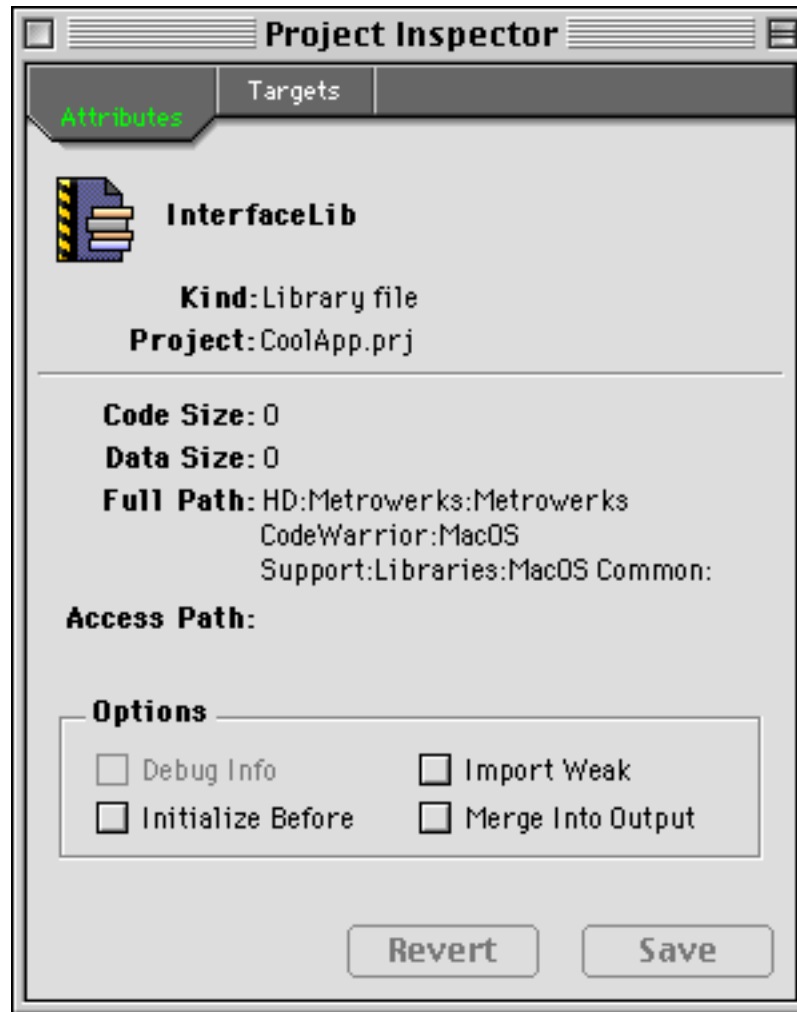
Locating Errors in Modified Files

If an error is corrected or the source code is changed, the compiler may not be able to find other errors in the source code file. This may result in an alert telling the user that the position of the error could not be found. When this happens, recompile your project to update the list of errors in the Message window.

Special Library Options (Mac OS)

There are a few special linker options you can use in your project. These options are only accessible from the [Project Inspector](#) window as shown in [Figure 10.5](#), available from the [Window Menu](#). These options are only available for library files in the project, and are disabled for text files in the project.

Figure 10.5 Special Library Options



The topics in this section are:

- [Import Weak](#)
- [Initialize Before](#)
- [Merge Into Output](#)

Import Weak

Import Weak tells the operating system to ignore unresolved symbols at load time. Use this option for features not always installed on all machines (such as QuickDraw GX or QuickTime).

For more information on shared libraries, see the *Targeting Mac OS* manual.

Initialize Before

Initialize Before indicates which PEF container (shared library or application) gets initialized first. By default, imported PEF containers are initialized before the PEF containers that import them. Use this option for mutually-dependent PEF containers, to specify which gets initialized first.

Merge Into Output

Merge Into Output allows you to put a copy of a shared library into your project's output file. For instance, if you have created your own custom shared library, you can merge it into the application's data fork, guaranteeing that the system finds the custom shared library.

This option may be useful for creating fat libraries as well, where you want to include both CFM68K and PowerPC code.

NOTE When you merge a shared library into your application, you are copying all the code fragments in that library. If the library contains both 68K and PowerPC code fragments, your application's data file gets them all, which is desirable for fat libraries, but in other cases may increase size unnecessarily.

Debugging Source Code



This chapter describes the CodeWarrior source-level debugger. To a great extent, this debugger works for all supported chips, operating systems, and languages (C, C++, Pascal, Java, and assembly language).

A debugger controls program execution so that you can see what happens internally as your program runs. You use a debugger to find problems while a program executes. The debugger can execute your program one statement at a time, suspend execution when control reaches a specified point, or interrupt a program that changes the value of a designated memory location. When the debugger stops a program, you can view the chain of function calls, examine and change the values of variables, and inspect the contents of the processor's registers.

NOTE This chapter describes the common functionality of the debugger for all platform targets. However, there might be some differences in debugger features for your particular build target. You should read the appropriate CodeWarrior targeting documentation to cover the specific differences for your build target. For information on which *Targeting* manual to read, see [“Targeting Documentation” on page 27](#).

This chapter discusses the following topics:

- [Symbolics Files](#)
- [Preparing for Debugging](#)
- [Using the Debugger](#)
- [Guided Tour of the Debugger](#)
- [Basic Debugging](#)
- [Expressions](#)

- [Troubleshooting](#)

Symbolics Files

A symbolics file contains information the debugger needs to debug most types of project files. Symbolics files include items such as the names of routines and variables (the symbols), their locations within the source code, and their locations within the object code.

The debugger uses the symbolics file to show the source code that corresponds to your object code. For example, when you stop program execution, the debugger shows you the source code.

Additionally, the debugger lets you view assembly-language instructions and memory addresses. See [“Viewing source code as assembly” on page 407](#) for more information.

CodeWarrior generates several symbolics formats to support a variety of build targets. [Table 11.1](#) lists some of the supported formats.

NOTE (Embedded) For some build targets, it is possible to debug an executable file that does not contain symbolics information. To learn more, refer to the *Targeting* manual appropriate for your particular build target.

Table 11.1 Supported symbolics file formats

Format	Principal Build Target
CodeView	Win32
DWARF	Embedded systems
SYM	Mac OS

See [“Preparing for Debugging” on page 397](#) for information on setting up projects and source files to create symbolics files.

For more information on compiler and linker settings, refer to [“Configuring Target Options” on page 321](#).

To learn more about symbolics files for a particular build target, see the corresponding *Targeting* manual.

Preparing for Debugging

To debug the code generated by a particular build target within your CodeWarrior project file, both the build target and the individual source files within it must be configured for debugging. After the build target and source files are configured properly, CodeWarrior generates symbolics information for use with the debugger.

NOTE (Embedded) It is possible to debug some build targets without first creating a project file. In such cases, the debugger automatically creates a default project file and uses default debug settings. For more information, refer to the *Targeting* manual appropriate for your particular build target.

This section describes the following topics:

- [Setting Up a Build Target for Debugging](#)
- [Setting Up a File for Debugging](#)
- [Generating Symbolics Information](#)

Setting Up a Build Target for Debugging

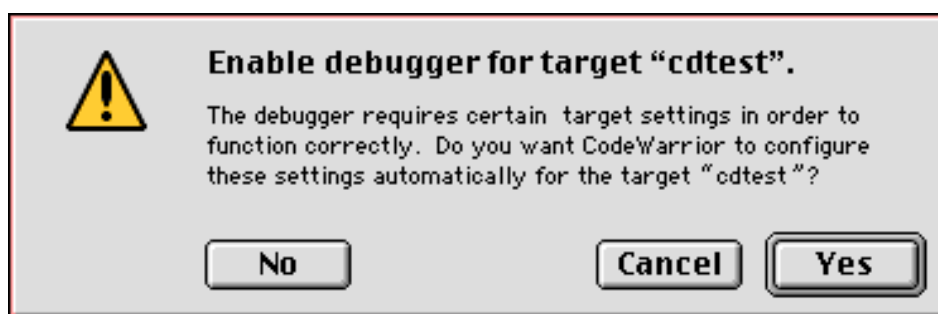
To prepare a build target for debugging, make sure it is the active build target. Then, choose the **Enable Debugger** command from the Project menu. When debugging is enabled for a build target, the menu command changes to **Disable Debugger**. Choosing **Disable Debugger** turns off debugging for the build target and changes the menu command back to **Enable Debugger**.

The **Enable Debugger** command activates the debugger. When your project's target settings are correctly configured, the compiler and linker generate a symbolics file containing information for debugging at the source-code level. By default, this symbolics file is saved in the same location as the output directory.

See [“Configuring Target Options” on page 321](#) for more information about build target settings. To learn about symbolics files and their use within the debugger, refer to [“Symbolics Files” on page 396](#).

When you choose **Enable Debugger**, you might see an alert ([Figure 11.1](#)). CodeWarrior is reminding you that it will modify the build target settings to prepare the project for debugging. Click **Yes** to apply the debugging changes to your build target.

Figure 11.1 **Accepting changes set by Enable Debugger**

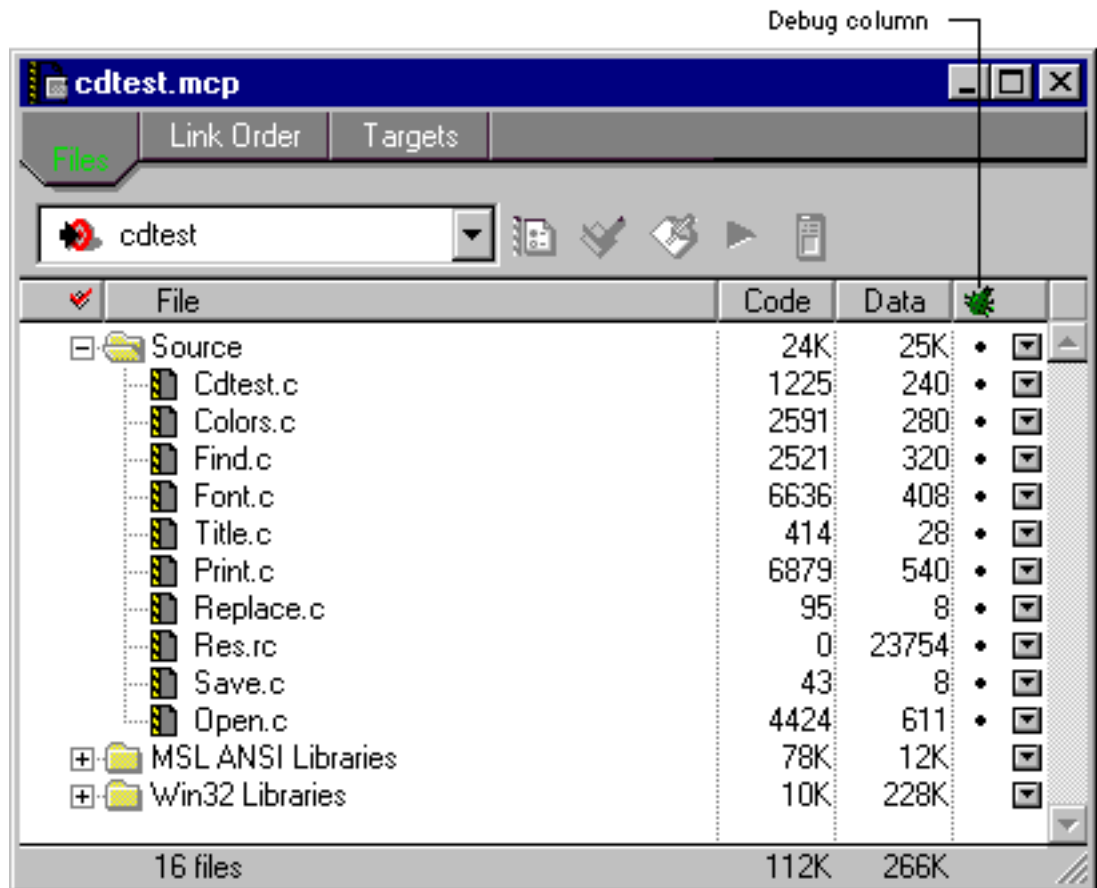


Setting Up a File for Debugging

After you enable debugging for the current build target, you have to set up your individual files for debugging. If you intend to debug your program, you typically enable debugging for all of your source files.

The Project window includes a debug column, as shown in [Figure 11.2](#). For files, a mark in this column means that symbolic information is generated; no mark means that symbolic information is not generated. For group names, a mark indicates that debugging is enabled for every file in the group and no mark means that debugging is disabled for one or more files in the group.

Figure 11.2 Setting debugging in the Project window



To toggle debugging for a file, click in the Debug column for that file. Clicking in the Debug column for a group name enables or disables debugging for all files in that group. If a file cannot be debugged (because it is a library or other non-source file), you cannot enable debugging for that file.

Generating Symbolics Information

To generate symbolics information, both the current build target and the source files within that target must be prepared for debugging. See [Setting Up a File for Debugging](#) to learn how to do this.

NOTE (Embedded) For some build targets, it is possible to debug an executable file that does not contain symbolics information. To learn

more, refer to the *Targeting* manual appropriate for your particular build target.

After you prepare the current build target and its source files, choose the **Make** command from the Project menu to compile and link your final code.

For more information on compiling and linking, see [“Configuring Target Options” on page 321](#) as well as the *Targeting* manual for your particular platform target.

Using the Debugger

To debug a project, you must first enable debugging for that project. Choose [Enable Debugger](#) from the Project menu to enable debugging. After debugging is enabled, the **Enable Debugger** command changes to **Disable Debugger**.

When you enable debugging, the **Run** command in the Project menu changes to **Debug**. For most projects, choosing the **Debug** command causes the IDE to compile and link your project, generate a symbolics file, and activate the debugger. The debugger automatically loads the symbolics file so that you can debug your project. See [“Preparing for Debugging” on page 397](#) for more information on this topic.

NOTE (Mac OS) If the debugger is currently disabled, you can Option-click the **Project** menu to choose the **Debug** command.

There are many build targets or kinds of code that the IDE cannot launch. For example, if you are writing an application plug-in, the plug-in cannot run on its own. In addition, the CodeWarrior IDE does not know which application must be running to invoke the plug-in. In such cases, you would need to specify the application that must launch in order to invoke the plug-in.

Usually, you open the symbolics file to debug the plug-in. Alternatively, you can follow these steps:

1. **Open the Target Settings window for the project.**

Choose **Target Settings** from the Project menu, where *Target* is the name of the current build target. For example, if your current build target is named `MyAppRelease`, you would choose **MyAppRelease Settings** from the Project menu.

2. **Display the Runtime Settings panel.**

Scroll through the list of panels on the left side of the Target Settings window. Click **Runtime Settings** to display that panel on the right side of the window.

3. **Click the Choose button.**

Use the dialog box that appears to choose an application for use with debugging shared libraries, dynamic link libraries (DLLs), and code resources. The IDE launches the application you specify when debugging the plug-in.

Guided Tour of the Debugger

This section explains the various windows, panes, and displays you can use when debugging. The topics discussed in this section include:

- [Stack Crawl Window](#)
- [Source Code Browser Window](#)
- [Symbol Hint](#)
- [Debugger Contextual Menu](#)
- [Processes Window](#)
- [Expressions Window](#)
- [Global Variables Window](#)
- [Breakpoints Window](#)
- [Watchpoints Window](#)
- [Register Details Window](#)
- [Register Windows](#)
- [Log Window](#)
- [Variable Window](#)
- [Array Window](#)

- [Memory Window](#)

Stack Crawl Window

When the debugger opens a symbolics file, it opens a Stack Crawl window (formerly known as the Program window). This window is shown in [Figure 11.3](#). The debugger allows more than one symbolics file to be open at a time: that is, you can debug more than one program at a time. For example, you can use this feature to debug an application and its plug-ins. A Stack Crawl appears for each active symbolics file and each thread.

The Stack Crawl window shows debugging information about the currently suspended process. The Stack Crawl window has four points of interest:

- [Debugger toolbar](#)
- [Stack Crawl pane](#)
- [Variables pane](#)
- [Source pane](#)

You can resize panes by clicking and dragging the resize bar between them. Use the Tab key to switch keyboard focus between panes.

Mac OS The active pane has a heavy border.

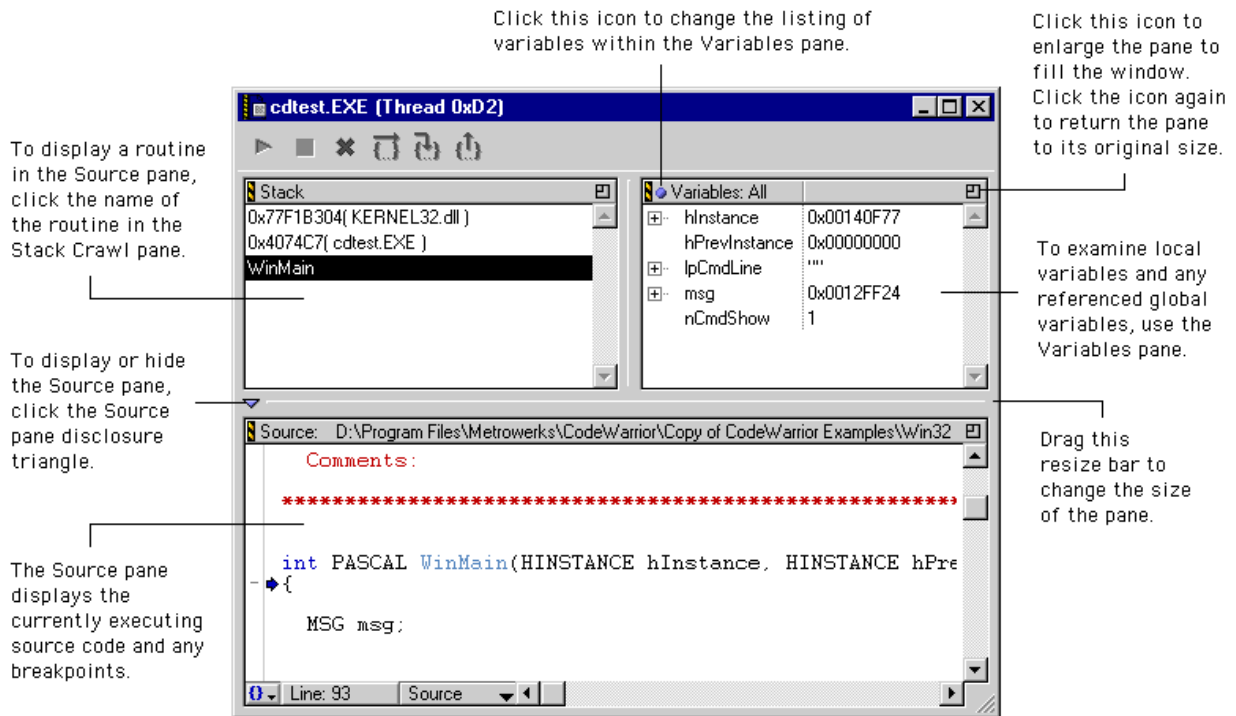
To select items in a focused Stack Crawl pane or Variables pane, type the first few characters of the item's name. Alternatively, use the arrow keys to navigate through the items in an active pane.

Mac OS You can also use Control-Tab as another way to navigate through items in an active Stack Crawl pane or Variables pane. To learn more about customizing key bindings, see [“Customizing Key Bindings” on page 304](#).

There are additional controls along the very bottom of the Source pane, to the left of the horizontal scroll bar:

- the [Routine Pop-Up Menu](#)
- the [Line Number Button](#)
- the [Source pane](#) pop-up menu

Figure 11.3 Parts of the Stack Crawl window



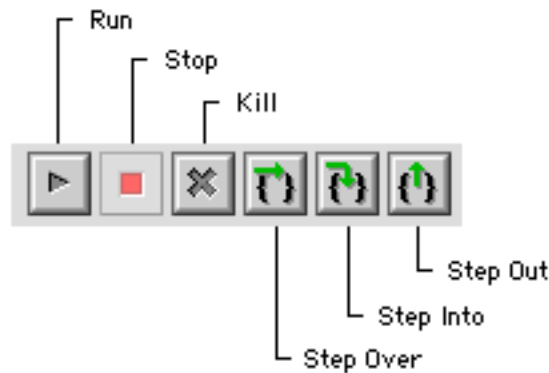
Debugger toolbar

The debugger toolbar ([Figure 11.4](#)) contains shortcut buttons for commonly used commands from the Project and Debug menus: **Run**, **Stop**, **Kill**, **Step Over**, **Step Into**, and **Step Out**.

Choose **Window > Toolbar > [Show Window Toolbar](#)** to display the toolbar. To hide the toolbar, choose **Window > Toolbar > [Hide Window Toolbar](#)**.

For more information, see [“Basic Debugging” on page 439](#).

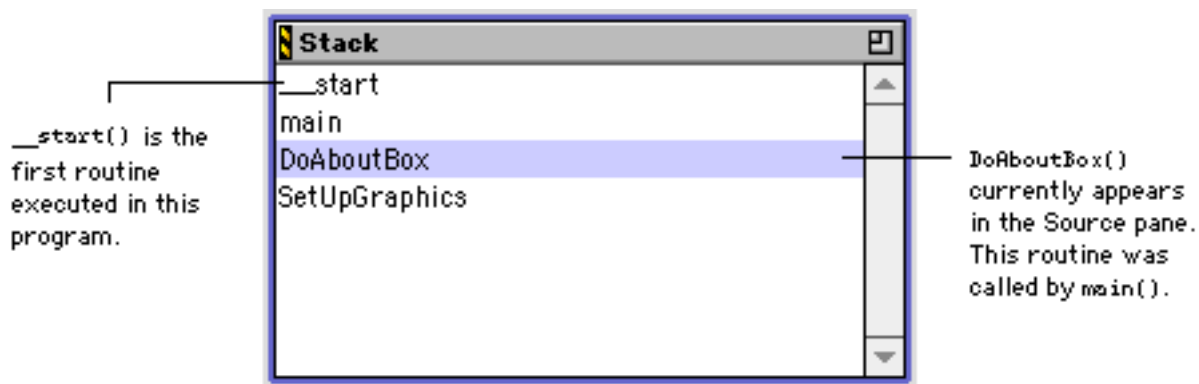
Figure 11.4 Debugger toolbar



Stack Crawl pane

The Stack Crawl window includes a Stack Crawl pane. This pane displays the current subroutine calling chain ([Figure 11.5](#)). Each subroutine is placed below the routine that called it. Select any routine in the Stack Crawl pane to display its code in the Source pane.

Figure 11.5 Stack Crawl pane



Variables pane

The Variables pane displays a list of local variables used in the program. The listing of these variables is controlled by the Variables Pane Listing icon (🔍), shown in [Figure 11.6](#). Clicking this icon toggles the list in the Variables pane between **Variables:All** and **Variables:Auto**. These list types are described in [Table 11.2](#).

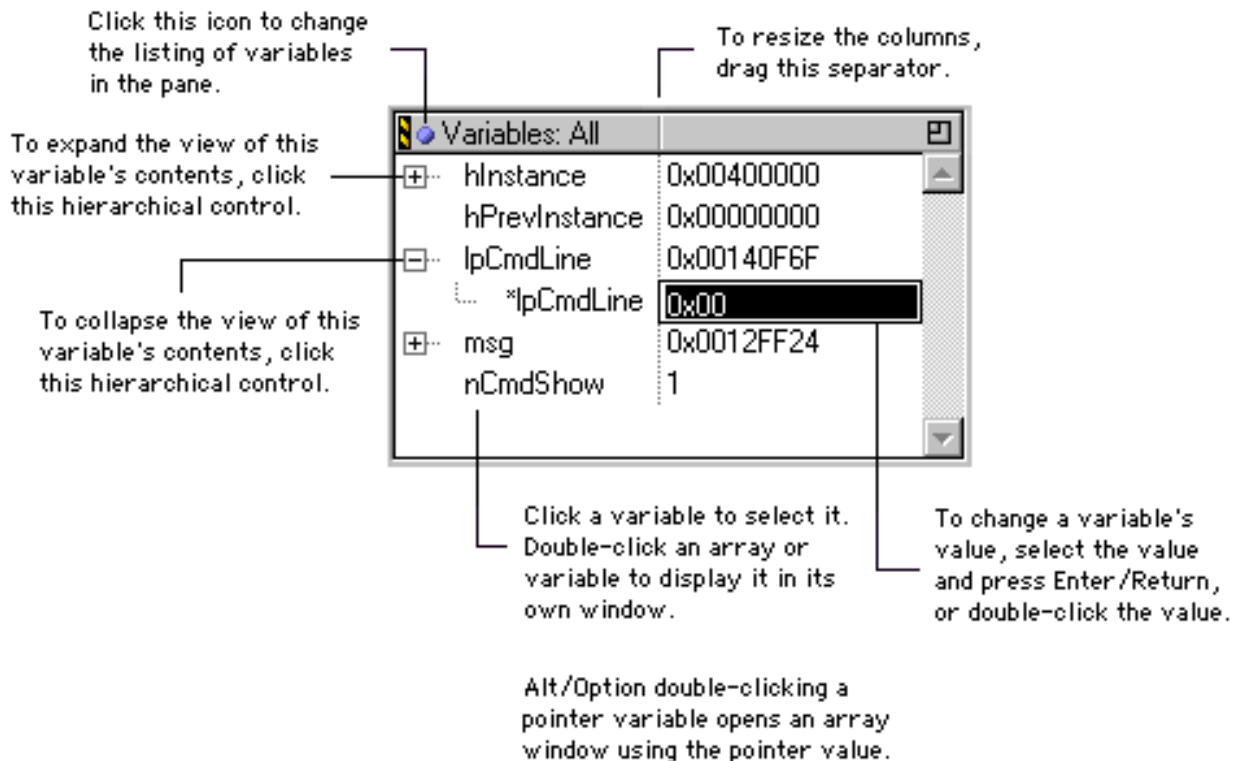
Table 11.2 Variables pane list types

Type	Variables Listed
Variables:All	All local variables in the code
Variables:Auto	Local variables of the routine in which the Current-statement arrow is currently located

Mac OS The Variables pane also displays any global variables referred to by the routine. A dashed line separates local and global variables.

The Variables pane lists the variables in outline form. Click the hierarchical control next to an entry to expand or collapse the view of its contents.

Figure 11.6 Variables pane



For example, in [Figure 11.6](#), clicking the hierarchical control next to the variable `msg` hides its members. Click the hierarchical control again to display the members. To dereference multiple levels of pointers and get directly to the data, hold down the Ctrl/Option key while expanding an entry. This feature is useful for expanding a handle to a structured type and viewing the structure members.

For more information about expanding and collapsing entries, see [“Expanding and Collapsing Groups” on page 75](#).

NOTE Use the General Registers and FPU Registers windows to view the contents of the central-processor and floating-point registers. If the current build target does not have an FPU, the FPU Registers window is not available. For more information, see [“Register Windows” on page 430](#).

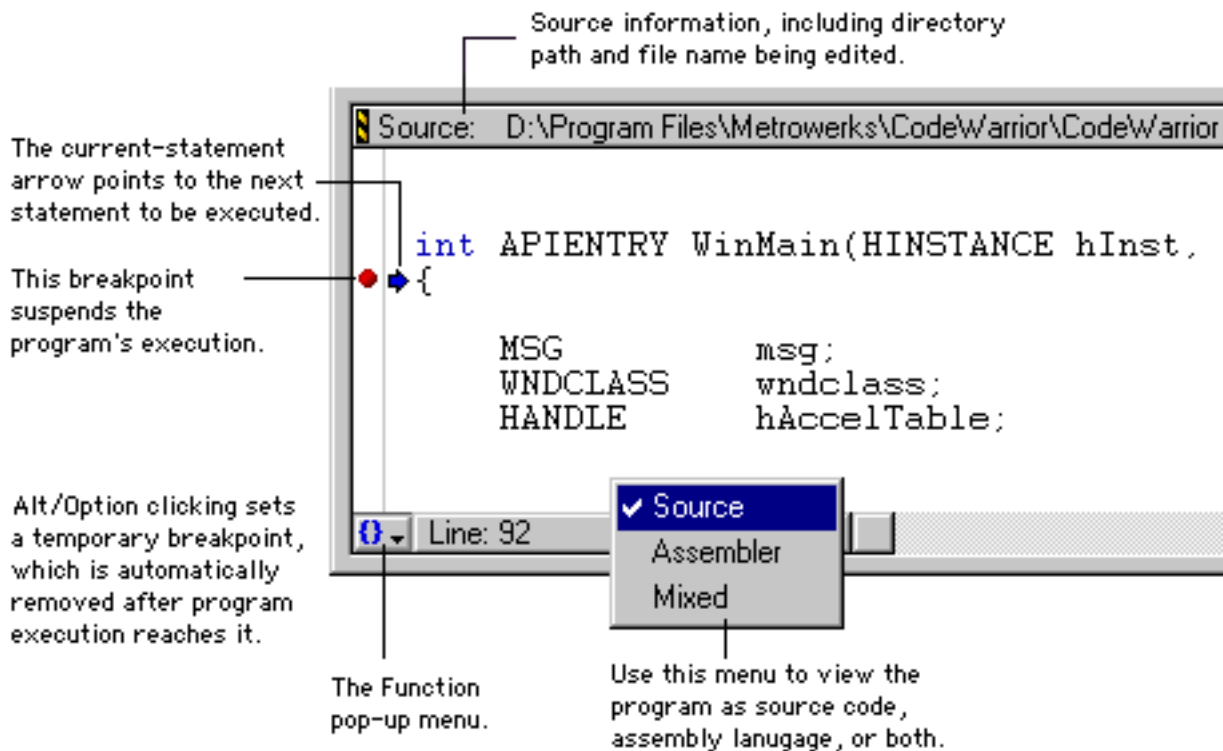
Source pane

The Source pane in the Stack Crawl window displays source code. The debugger takes the source code directly from the source code files in the current build target, including any comments and white space. The pane shows C/C++, Pascal, Java, and in-line assembly code exactly as it appears in your program’s source code ([Figure 11.7](#)), using the font and color specified in the [Editor Settings](#) preference panel.

Click the Source pane disclosure triangle, shown in [Figure 11.3 on page 403](#), to show or hide the Source pane. If you hide the Source pane, the debugger opens both an editor window and a Stack Crawl window after you step once through your code. The editor window shows the same source code and breakpoints as the Source pane.

The Source pane lets you step through the source code line by line. The current-statement arrow, which always points to the next statement to be executed, indicates your progress. The program counter in the central processing unit of your computer is set to the instruction address pointed to by the current-statement arrow. Thus, the current-statement arrow is also known as the program counter.

Figure 11.7 Source pane in the Stack Crawl window



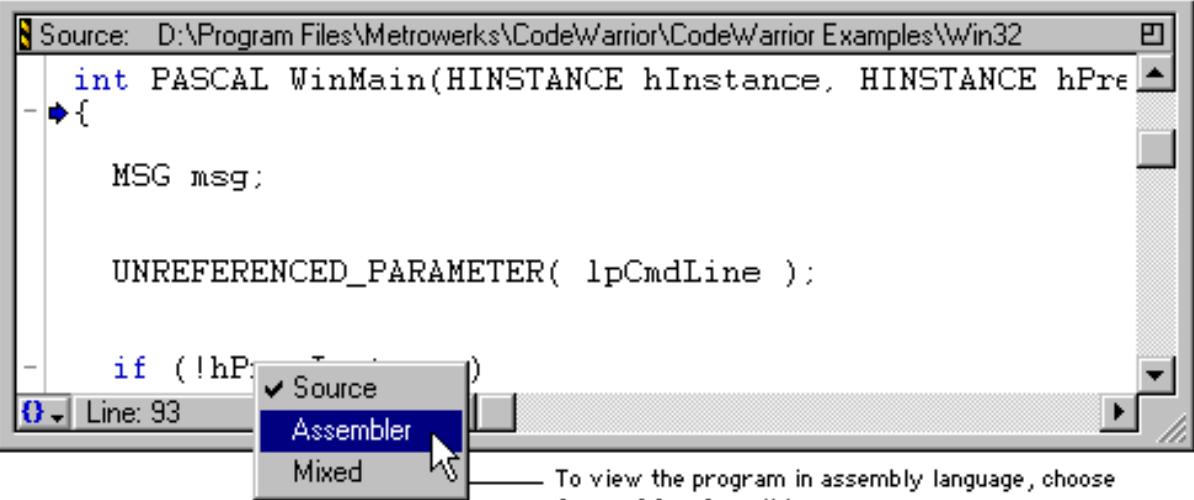
The debugger executes all statements in a source-code line with each step through your code. The current-statement arrow dims whenever the program counter is within, but not at the beginning of, a source-code line. If the source-code line contains more than one routine, you must step over, step into, or step out of each routine of interest.

Viewing source code as assembly

To view your source code as assembly language, click the Source pop-up menu at the bottom of the Stack Crawl window. Choose **Assembler** to display the contents of the Source pane as assembly code ([Figure 11.8](#)). While viewing assembly code, the debugger lets you step through the code and set breakpoints, one routine at a time.

Figure 11.8 Source and assembly views

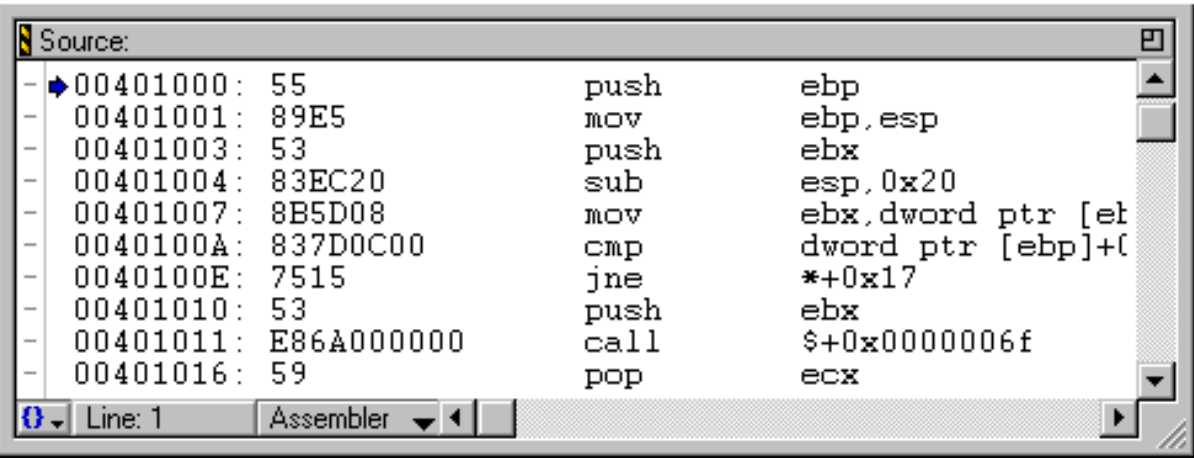
Before



To view the program in assembly language, choose **Assembler** from this pop-up menu.

Breakpoints and stepping are allowed in the Assembler view.

After

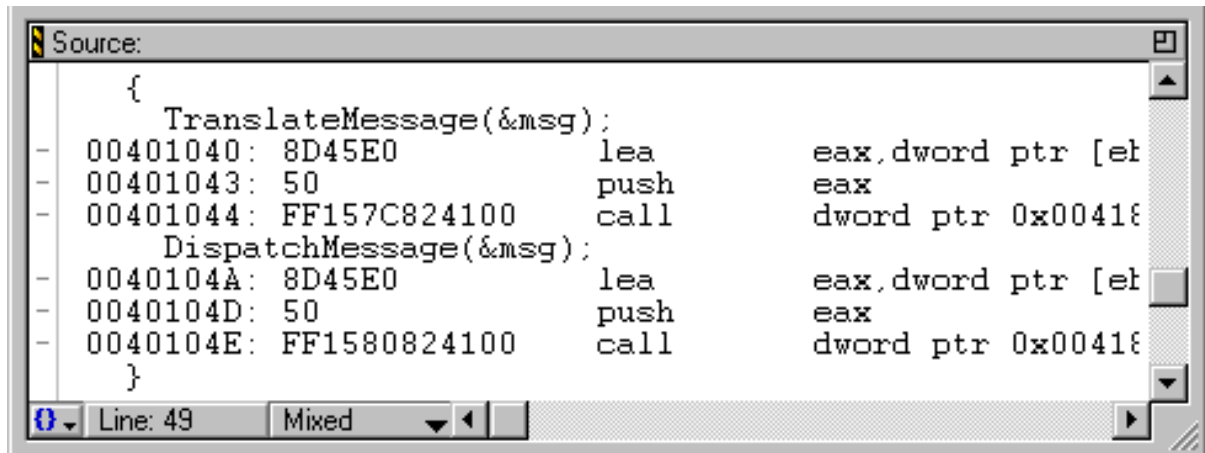


NOTE Use the General Registers and FPU Registers windows to view the contents of the central-processor and floating-point registers. If the current build target does not have an FPU, the FPU Registers window is not available. For more information, see [“Register Windows” on page 430](#).

Viewing source with mixed assembly

To view your source code and assembly language together, click the Source pop-up menu at the bottom of the Stack Crawl window. Choose **Mixed** to display the source code of the current routine intermixed with assembly code ([Figure 11.9](#)). The source code that produced the assembly instructions appears before the assembly itself. When viewing code in mixed view, you can set breakpoints and step through code, but only for assembly language instructions. Notice you cannot set breakpoints on source-code lines, as shown in [Figure 11.9](#).

Figure 11.9 Viewing mixed code



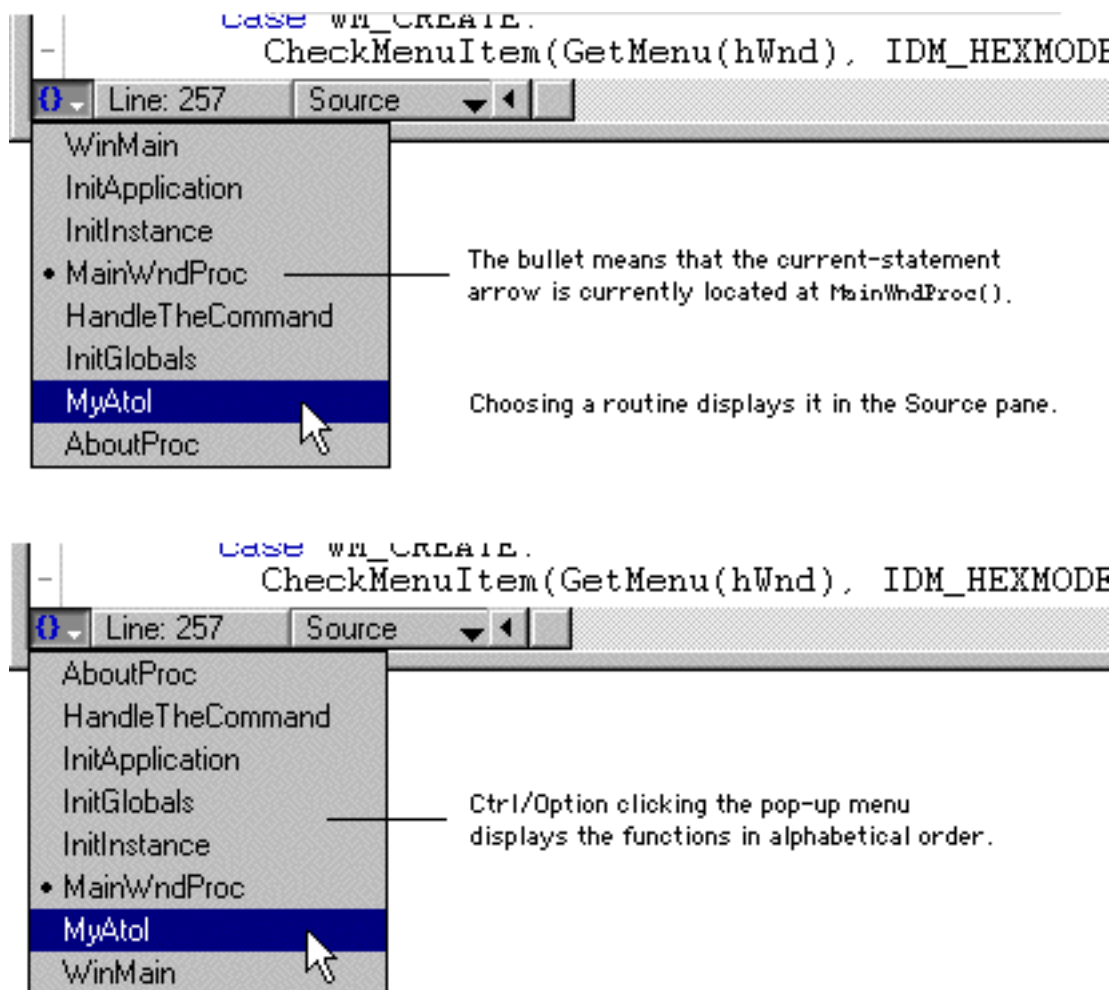
If no symbolics information is available for the object code, the display reverts to Assembly. There is no syntax coloring for this view, so the Source pane displays all text in plain format.

Function pop-up menu

The Function pop-up menu, at the bottom-left corner of the Source pane, contains a list of the routines defined in the source file selected in the Source pane ([Figure 11.10](#)). Selecting a routine in the Function pop-up menu displays that routine in the Source pane.

Alt/Option click the Function pop-up menu to display the menu items in alphabetical order.

Figure 11.10 Function pop-up menu



Source Code Browser Window

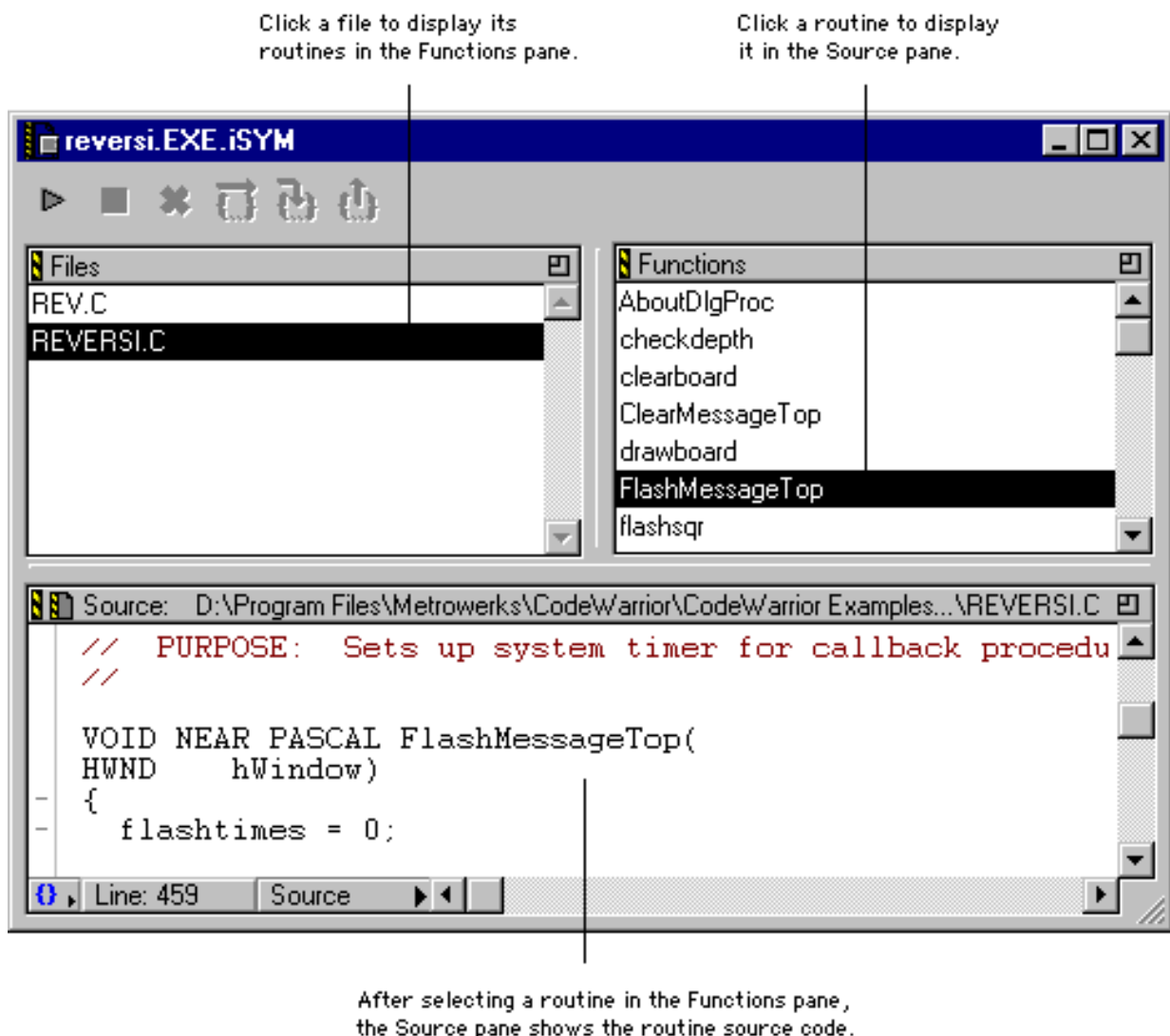
When you double-click a symbolics file, the debugger opens a Source Code Browser window, as shown in [Figure 11.11](#).

The Source Code Browser window somewhat resembles the Stack Crawl window in both appearance and functionality, but displays different information. The Source Code Browser window lets you view any file in the build target for which the symbolics file was generated, whereas the Stack Crawl window only displays the file containing the routine selected from the Stack Crawl pane. The Source Code Browser window lets you view or edit the values of all global variables in your program; the Stack Crawl window lets you

change only those global variables referenced by active routines in the call chain. For more information about viewing global variables, see [“Global Variables Window” on page 421](#).

NOTE Do not confuse the Source Code Browser window with the Class Browser window. Although the two look similar, the Source Code Browser window opens only when you double-click a symbolics file.

Figure 11.11 Source Code Browser window



TIP You can use the Global Variables window as an alternative to the Source Code Browser window. For more information, see [“Global Variables Window” on page 421](#).

The Source Code Browser window has three panes:

- the [Files pane](#) at the top left
- the [Functions pane](#) at the top right
- the [Source pane](#) at the bottom

Like the Stack Crawl window, the Source Code Browser window has a debugger toolbar, a Function pop-up menu, a line number button, and a Source pop-up menu. The Source Code Browser window lets you resize panes by clicking and dragging the resize bar between them. You can switch between panes with the Tab key.

Mac OS The active pane has a heavy border.

To select items in an active Files pane or Functions pane, type the first few characters of the item name. Alternatively, use the arrow keys to navigate through the items in an active pane.

Mac OS You can also use Control-Tab as another way to navigate through items in an active Files pane or Functions pane.

The debugger allows more than one symbolics file to be open at a time: that is, you can debug more than one program at a time. You can use this feature, for example, to debug an application and its plug-ins. A Source Code Browser window appears for each opened symbolics file.

To learn more about the contents of the Stack Crawl window, refer to [“Stack Crawl Window” on page 402](#).

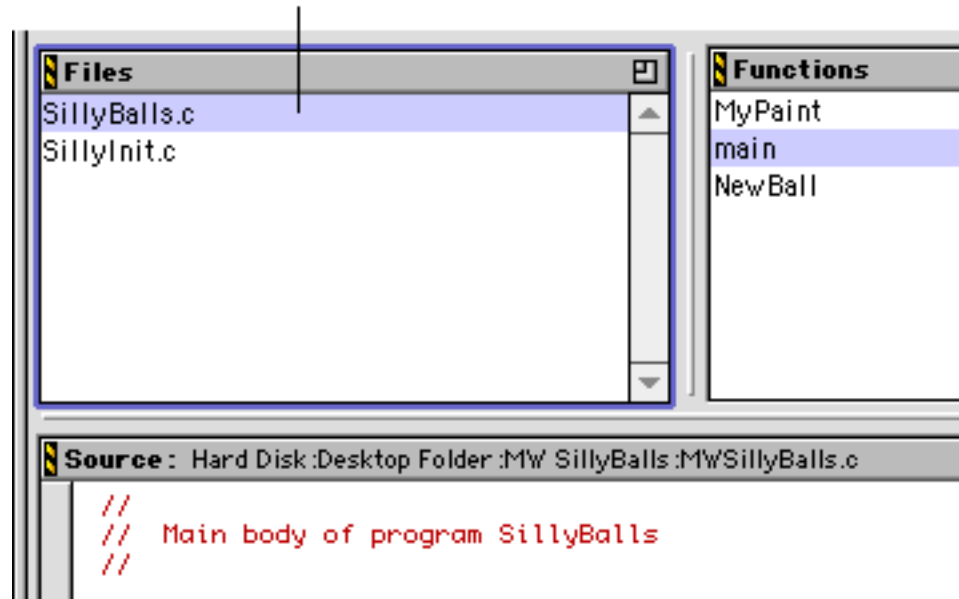
Files pane

The Files pane ([Figure 11.12](#)) in the Source Code Browser window displays a list of all source files in the build target being debugged. When you select a file name in this pane, the Functions pane displays a list of the routines declared in the file. You use the Files pane in conjunction with the Functions and Source panes to set breakpoints in your program.

For more information, see [“Global Variables Window” on page 421](#) and [“Breakpoints” on page 457](#).

Figure 11.12 Files pane

The Files pane lists the program's source code files that have symbolics information. The highlighted file, in this case `SillyBalls.c`, is displayed in the Source pane.

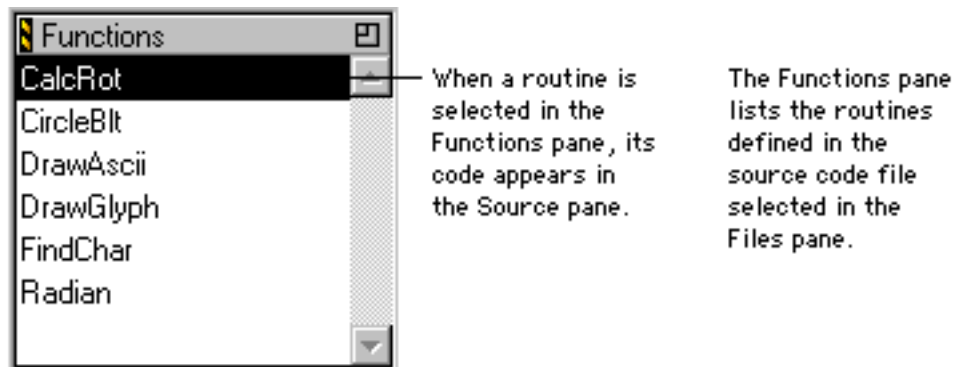


Functions pane

When you select a source-code file in the [Files pane](#), the Functions pane ([Figure 11.13](#)) presents a list of all routines defined in that file. Clicking a routine name displays that routine in the Source pane at the bottom of the window.

NOTE If your code is written in C++ or Object Pascal, the **Sort functions by method name in browser** option in the Display Settings preference panel (see [“Display Settings” on page 283](#)) alphabetizes function names which have the form `className::methodName` by `methodName` instead of by `className`.

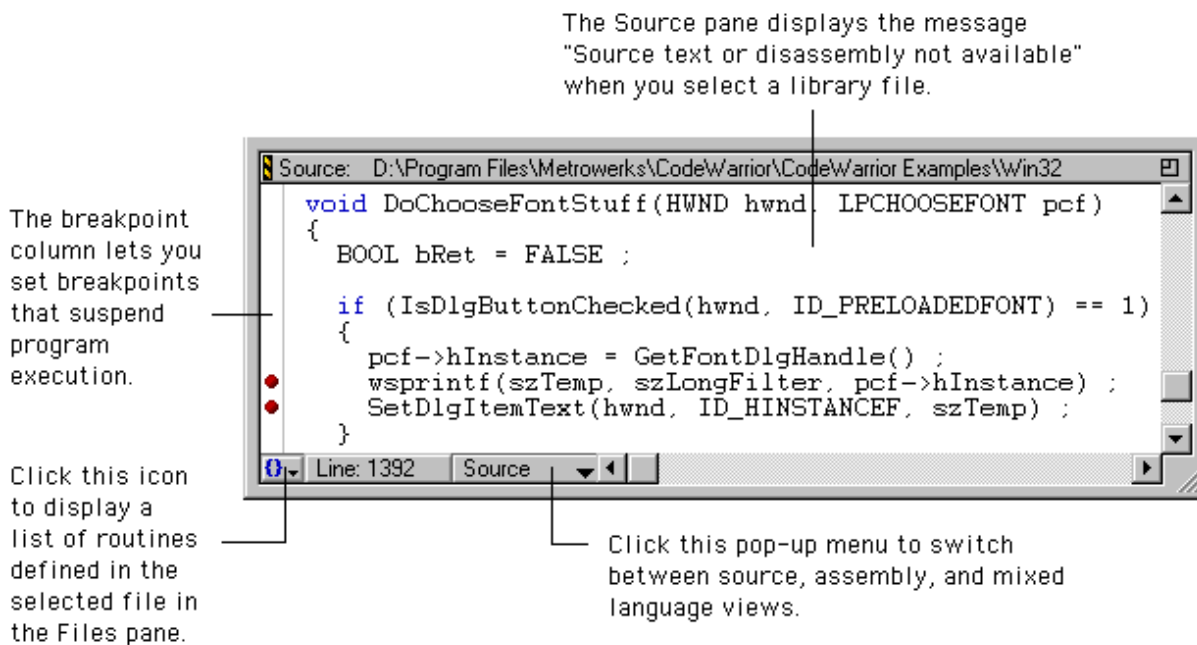
Figure 11.13 Functions pane



Source pane

The Source pane in the Source Code Browser window allows you to browse the contents of the source-code file selected in the [Files pane](#) (Figure 11.14). You can use the Source pane to set breakpoints in any file listed in the Files pane. To learn about setting breakpoints, refer to [“Breakpoints” on page 457](#).

Figure 11.14 Source pane in the Source Code Browser window



Notice, however, that the Source pane does not show the currently executing statement. The current-statement arrow in the Stack Crawl window shows the currently executing statement. You can also use the Stack Crawl window to view local variables. For more information, see [“Stack Crawl Window” on page 402](#). The Source pane displays code in the font and colors specified in the panels of the IDE Preferences window. For more information, see [“Editor Settings” on page 273](#) and [“Display Settings” on page 283](#).

If an item selected in the Files pane does not contain source code, the Source pane displays the message “Source text or disassembly not available.”

The bottom of the Source Code Browser window has a Source pop-up menu like the one in the Stack Crawl window (see [“Viewing source code as assembly” on page 407](#)). Choose **Assembler** to display the contents of the source pane as assembly code, as shown earlier in [Figure 11.8 on page 408](#). You can set breakpoints in assembly code, just as you can in source code. Choose **Mixed** to display source code intermixed with assembly language, as shown earlier in [Figure 11.9 on page 409](#).

Function pop-up menu

The Function pop-up menu, at the bottom-left corner of the Source pane, contains a list of the routines defined in the source file selected in the [Files pane](#). Selecting a routine in the Function pop-up menu displays that routine in the Source pane, just as if you had clicked the same routine in the Functions pane.

Alt/Option click the Function pop-up menu to display the menu sorted alphabetically, as shown earlier in [Figure 11.10 on page 410](#).

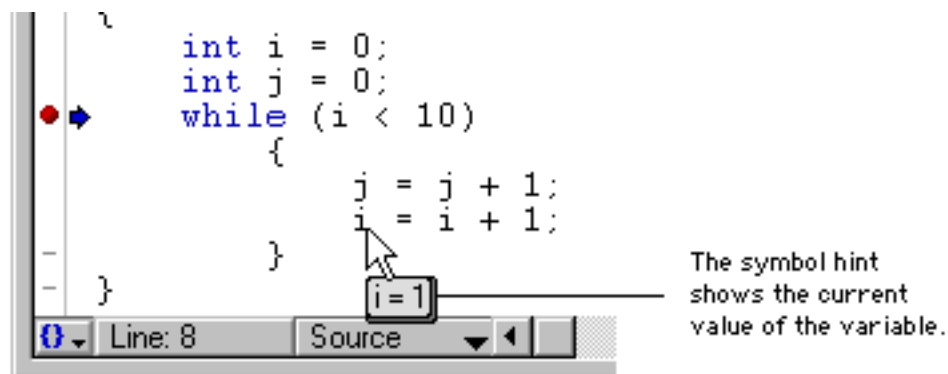
NOTE The Function pop-up menu does nothing if there is no source code displayed in the Source pane.

Symbol Hint

When you debug source code, the debugger shows useful information about your program’s variables. This information appears automatically as you move the cursor over particular variables in source-code views. For example, suppose you start a

debugging session and examine your source code in the [Source pane](#) of the Stack Crawl window. When you position the cursor over the variable `i`, as shown in [Figure 11.15](#), the debugger displays the current value of that variable.

Figure 11.15 **Symbol hint**



NOTE The symbol hint is only available when the debugger is active. If you do not see information about your program's variables, you need to activate the debugger. For more information, see [“Preparing for Debugging” on page 397](#).

Debugger Contextual Menu

The debugger includes a contextual menu ([Figure 11.16](#)) that provides convenient access to various debugging commands. To display the contextual menu, perform the following steps:

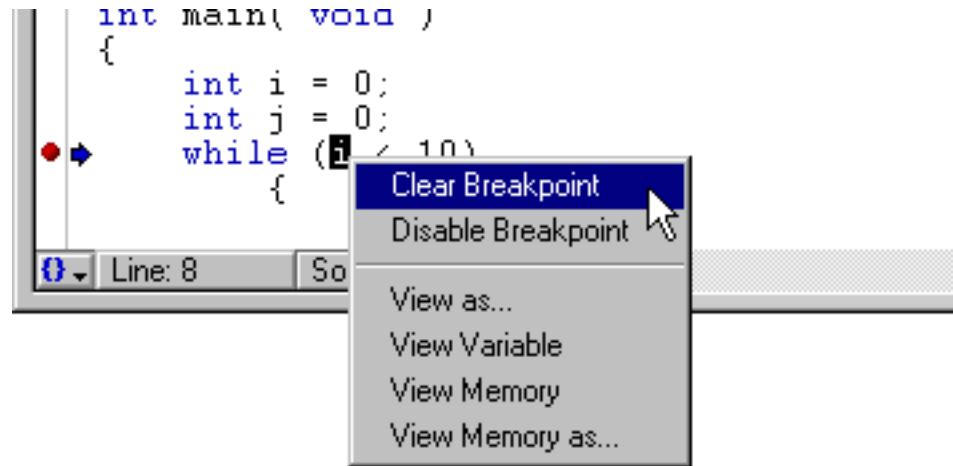
Windows Right-click an item.

Mac OS Control-click an item, or click and hold on an item.

Solaris Control-click an item, or click and hold on an item.

The debugger uses the context of the item you select to determine the commands that appear in the contextual menu. For example, if you choose a variable, the contextual menu includes commands for viewing the contents of that variable.

Figure 11.16 Debugger contextual menu



TIP The debugger contextual menu is useful in a variety of situations. For instance, you can use the contextual menu in the [Variables pane](#) and the [General Registers window](#) to view variables in different formats. You can also use the contextual menu in the [Source pane](#) to manipulate breakpoints and move the program counter. Experiment with the contextual menu to discover all of its features.

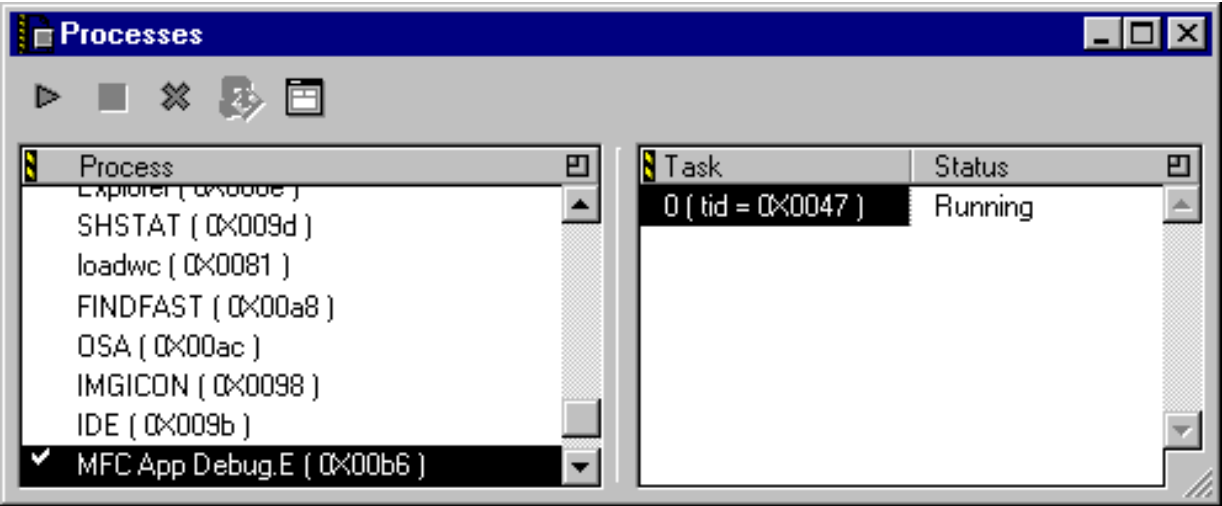
Processes Window

The Processes window ([Figure 11.17](#)) lists currently running processes, including some hidden processes. The Processes window also lists tasks for a selected process. To open the Processes window, choose **Processes Window** from the Window menu.

The Processes window has two panes and a toolbar:

- [Processes window toolbar](#)—allows you to run, stop, or kill processes and tasks under debugger control. The toolbar also lets you attach processes and open Stack Crawl windows.
- [Process pane](#)—shows currently running processes
- [Task pane](#)—shows tasks in a selected process

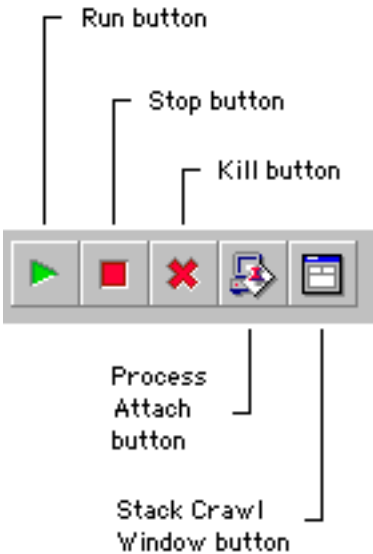
Figure 11.17 Processes window



Processes window toolbar

The Processes window toolbar ([Figure 11.18](#)) has controls to **Run**, **Stop**, and **Kill** a process under debugger control. These controls have no effect on any other currently running processes or tasks. The Process Attach and Stack Crawl window controls affect selected processes.

Figure 11.18 Processes window toolbar



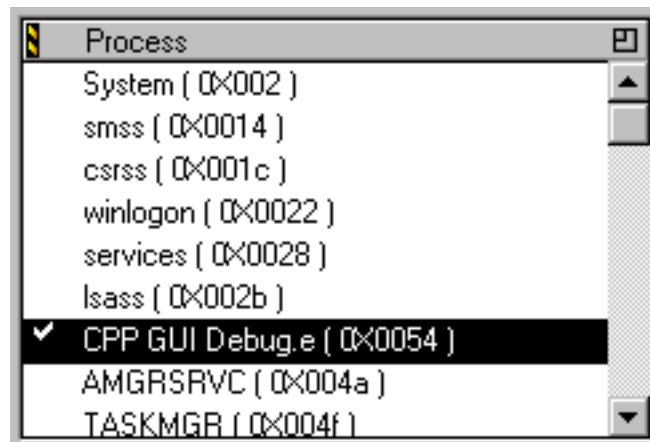
The Process Attach button assigns a debugging session to the selected process in the [Process pane](#). This button is useful for debugging processes that the debugger cannot recognize normally. For example, when debugging a program that uses dynamic link libraries or shared libraries, you can use the Process Attach button to place those libraries under debugger control. A check mark appears next to attached processes.

The Stack Crawl window button opens a Stack Crawl window for the selected process or task. You can have multiple Stack Crawl windows open at a time. After clicking the button, the Stack Crawl window for the selected process or task appears frontmost.

Process pane

The Process pane lists all currently running processes. A checkmark appears next to each process under debugger control. To place a selected process under debugger control, click the Process Attach button. Double-clicking a process name activates that process.

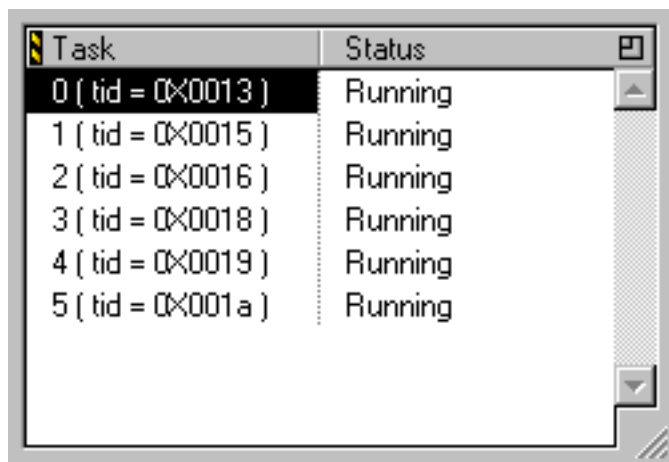
Figure 11.19 **Process pane**



Task pane

The Task pane lists all the currently running tasks for a given process. Only tasks from programs under debugger control are shown. Double-clicking a task name opens a Stack Crawl window with the code for that task. You can also choose a task and then click the Stack Crawl window button ([Figure 11.18](#)).

Figure 11.20 Task pane



There are two columns in the Task pane. The left column shows the task ID. The right column indicates the task status as either running, stopped, or crashed.

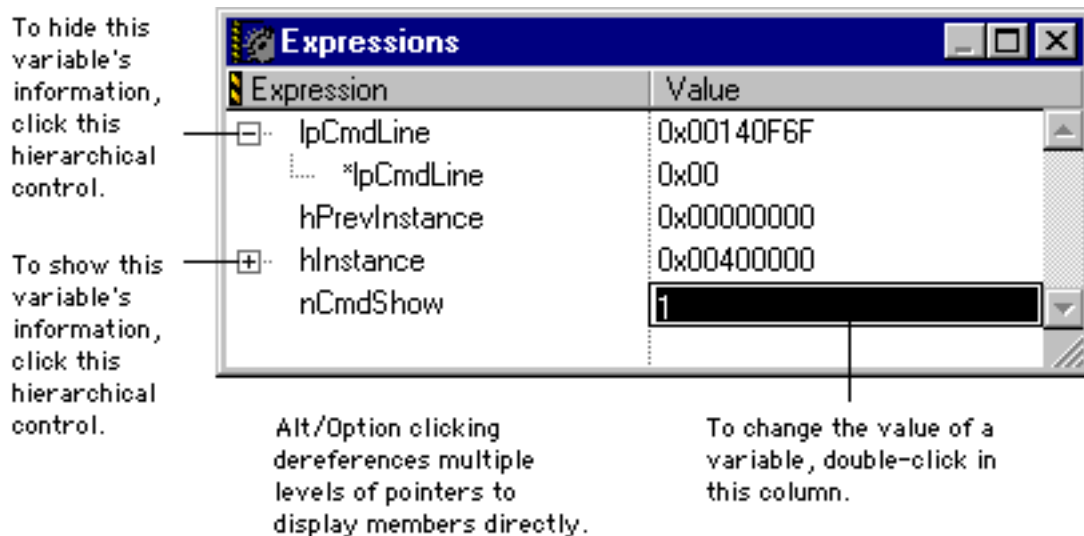
Expressions Window

The Expressions window ([Figure 11.21](#)) provides a convenient location to place frequently used local variables, global variables, structure members, and array elements.

To open the Expressions window, choose **Expressions Window** from the Window menu.

Use the **Copy to Expression** command in the Data menu to add selected items to the Expressions window. Alternatively, you can select the same command from the debugger contextual menu (see [“Debugger Contextual Menu” on page 416](#)). You can also use the mouse to drag and drop items from other variable panes and windows into the Expressions window. To reorder items in the Expressions window, drag each item to its new position in the list.

Figure 11.21 Expressions window



To remove an item from the Expressions window, select the item and press the Backspace/Delete key. You can also choose **Delete** (Windows) or **Clear** (Mac OS, Solaris, Linux) from the Edit menu to remove the item.

Unlike local variables displayed in an ordinary Variables window, local variables in the Expressions window are not removed on exit from the routines in which they are defined.

To learn about the Expressions window, refer to [“Using the Expressions window” on page 476](#).

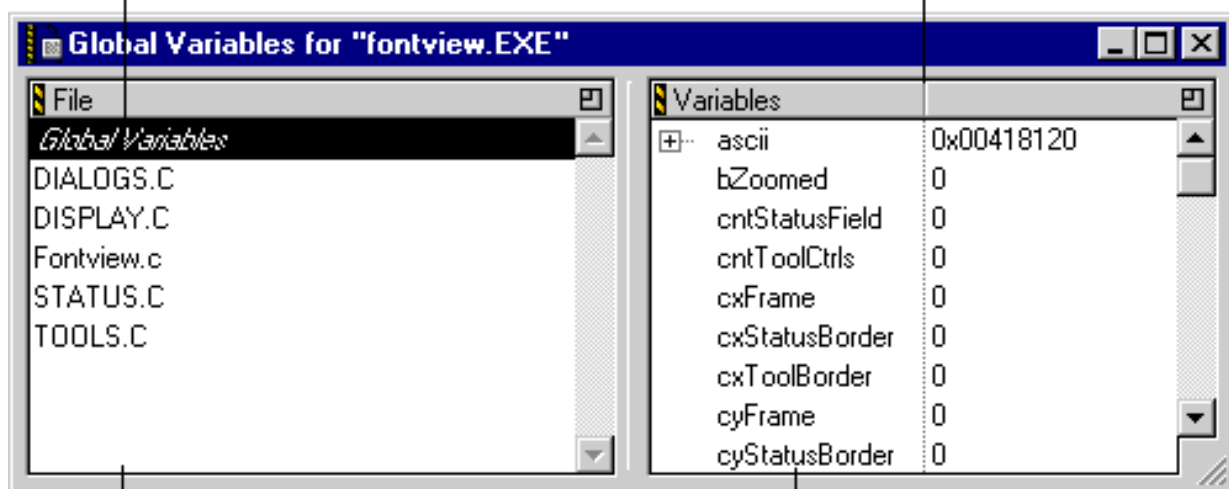
Global Variables Window

The Global Variables window ([Figure 11.22](#)) shows all global variables used by your program. You can also view static variables by selecting a file in the File pane. The static variables appear in the Variables pane.

Figure 11.22 Global Variables window

Click *Global Variables* to view all global variables in the program.

To resize column widths, drag this separator.



Click a file to display its static variables in the Variables pane.

Click a variable to select it. Double-click an array or variable to display it in its own window.

Alt/Option double-click a pointer variable to open an Array window using the pointer values.

Placing globals in a separate window

To display a global variable in its own window, double-click the name of the global variable in the Variables pane. A new Variable window opens and shows the name and value of the global variable. You can also open a Variable window by selecting the desired variable in the Variables pane and choosing the **View Variable** or **View Array** commands from the Data menu or the debugger contextual menu (see [“Debugger Contextual Menu” on page 416](#)). A global variable displayed in its own window can be viewed and edited the same way as in the Variables pane. You can also add global variables to the Expressions window.

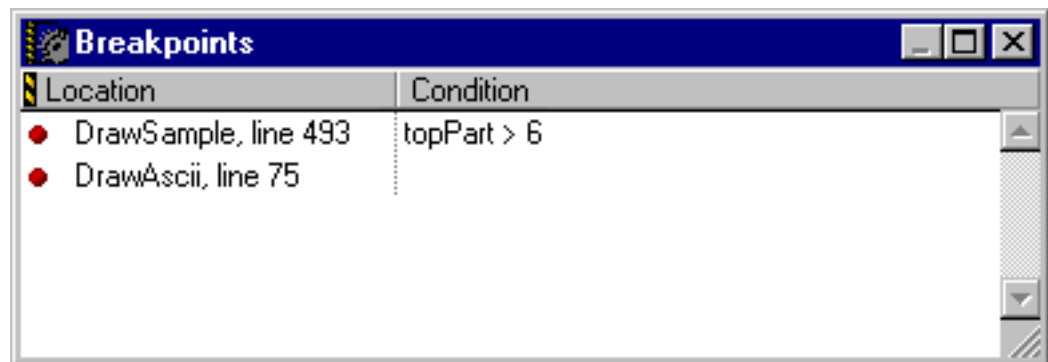
Windows containing global variables remain open for the duration of the debugging session. To close all open Variable windows, make sure that one of the Variable windows is frontmost, then choose **Close All Variable Windows** from the File menu.

For more information, refer to [“Expressions Window” on page 420](#), [“Variable Window” on page 435](#), and [“Array Window” on page 436](#).

Breakpoints Window

The Breakpoints window ([Figure 11.23](#)) lists the breakpoints in your current build target by source file and line number. To open the Breakpoints window, choose **Breakpoints Window** from the Window menu.

Figure 11.23 Breakpoints window



A breakpoint marker appears to the left of each listing. Clicking a breakpoint marker toggles its current status. A solid dot indicates that the breakpoint is active, while a circle that indicates that the breakpoint is inactive. In either case, the IDE remembers the position of the breakpoints in the program. Alt/Option click one of the breakpoint markers in the Breakpoints window to simultaneously make all of the listed breakpoints active or inactive. Double-click a breakpoint listing to display the corresponding line of source code in an editor window.

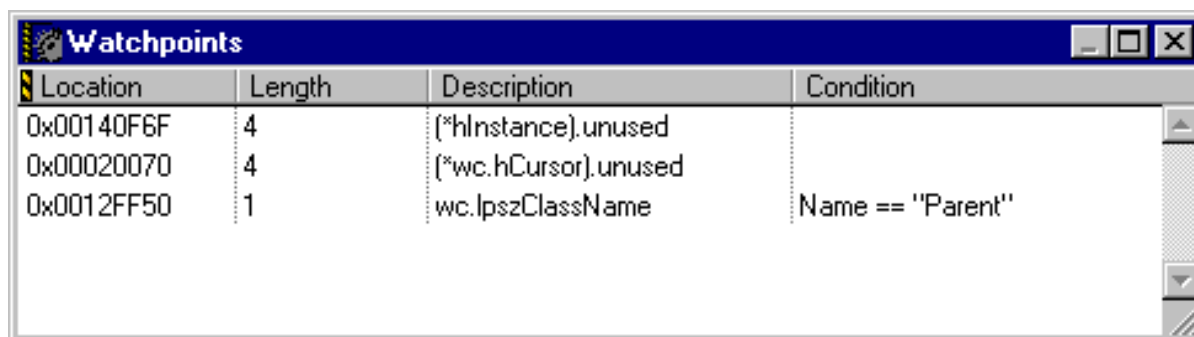
Each breakpoint can have an attached condition. If the condition is true and the breakpoint is active, the breakpoint stops program execution. If the breakpoint is inactive or the condition is false, the breakpoint has no effect.

For more information, see [“Breakpoints” on page 457](#), [“Show Breakpoints” on page 640](#), [“Hide Breakpoints” on page 640](#), and [“Conditional breakpoints” on page 461](#).

Watchpoints Window

The Watchpoints window ([Figure 11.24](#)) lists the watchpoints in your current build target by memory address. To open the Watchpoints window, choose **Watchpoints Window** from the Window menu.

Figure 11.24 Watchpoints window



You can clear a watchpoint by selecting it and performing any of the following steps:

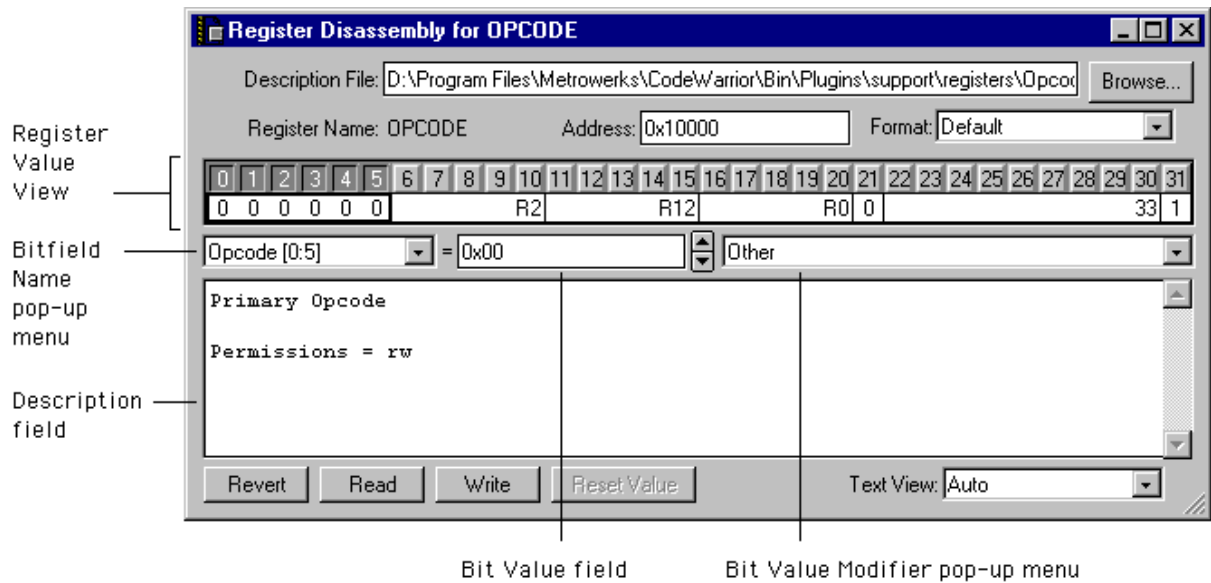
- Choose **Clear Watchpoint** from the Debug menu.
- Choose **Delete** (Windows) or **Clear** (Mac OS, Solaris, Linux) from the Edit menu.
- Press the Backspace/Delete key.

For more information, see [“Watchpoints” on page 465](#).

Register Details Window

The Register Details Window ([Figure 11.25](#)) allows you to view detailed information about individual register bits in 32-bit registers. This window shows information for both system registers and memory-mapped registers. To open the Register Details window, choose **Register Details Window** from the Window menu.

Figure 11.25 Register Details window



The Register Details window contains fields that describe the register, its bitfields, and the values of those bitfields. The information that appears in these fields is derived from Extensible Markup Language (XML) files in the Registers folder. The Support folder, which is inside the Plugins folder of your CodeWarrior installation, contains the Registers folder.

NOTE For more information about creating XML files for use with the Register Details window, consult the CodeWarrior White Paper entitled *Register Details Window XML File Format*.

The Register Details window consists of the following parts:

- [Description File field](#)
- [Browse button](#)
- [Address field](#)
- [Format pop-up menu](#)
- [Register Value view](#)
- [Bitfield Name pop-up menu](#)
- [Bit Value field](#)
- [Bit Value Modifier pop-up menu](#)

- [Description field](#)
- [Revert button](#)
- [Read button](#)
- [Write button](#)
- [Reset Value button](#)
- [Text View pop-up menu](#)

Description File field

To view information for a specific register, type the name of the register in the **Description File** field and press Enter/Return. Alternatively, type the full path to the register description file on your hard disk and press Enter/Return. The debugger ignores the case of the text you type.

After you type a register name or path and press Enter/Return, the debugger searches the Registers folder and the project access paths for an XML file whose name matches the name you typed. If the debugger finds a matching name, the corresponding description file appears in the Register Details window. If the debugger does not find a matching name, an error message appears.

For example, to view the contents of the Opcode register, type Opcode in the **Description File** field and press Enter/Return. Alternatively, type the full path to the `opcode.xml` file in the Registers folder and press Enter/Return. In either case, the debugger matches the name you typed with the `opcode.xml` file and displays the Opcode register information.

After displaying the register information, the debugger attempts to read the current register values and display them in the Register Details window. An error message appears if this attempt fails.

Browse button

Click the Browse button to display an Open dialog box. Use the dialog box controls to locate an XML register description file on your hard disk. The description file you select appears in the Register Details window.

Address field

Enter in this field the starting address of the register you wish to see in the [Register Value view](#). If you enter an invalid or incorrect address, an error message appears.

Format pop-up menu

Choose from this pop-up menu the data format in which you want to see the contents of the Register Details window. You can choose from binary, character, decimal, unsigned decimal, hexadecimal, or default formats. The default format causes the debugger to select the format most appropriate for the data in the register.

Register Value view

The Register view shows the current contents of 32 bits of register data, starting at the address specified in the [Address field](#). The data appears according to the format selected in the [Format pop-up menu](#).

The Register Value view groups the 32 bits of data into the appropriate register bitfields. Clicking one of the bits selects its associated bitfield. Additional information about the bitfield, such as its name and permissions, appears in the [Description field](#) of the Register Details window.

Bitfield Name pop-up menu

Use the Bitfield Name pop-up menu to select bitfields within the [Register Value view](#). The pop-up menu shows the name of the bitfields and their corresponding bit positions. A selected bitfield is shown in [Figure 11.25 on page 425](#).

To select a different bitfield, choose its name from the [Bitfield Name pop-up menu](#). Alternatively, click the desired bitfield in the [Register Value view](#) to select it. To de-select all bitfields, choose **None** from the [Bitfield Name pop-up menu](#).

Bit Value field

The Bit Value field shows the current value of the selected bitfield in the [Register Value view](#). The [Format pop-up menu](#) determines the value format.

To change the current value, type a new value into the Bit Value field and press Enter/Return. The [Register Value view](#) updates to reflect your changes.



You also can click the increment and decrement buttons next to the Bit Value field to change the current value.

NOTE Changing the data in the Bit Value field only changes the data shown in the [Register Value view](#). To change the data in the register itself, you must click the [Write button](#).

Bit Value Modifier pop-up menu

The Bit Value Modifier pop-up menu shows additional options for changing the currently selected bitfield value in the [Register Value view](#). Choose the desired value from the pop-up menu. Sometimes, the pop-up menu also describes the meaning of bit values.

NOTE Changing the data in the Bit Value Modifier pop-up menu only changes the data shown in the [Register Value view](#). To change the data in the register itself, you must click the [Write button](#).

Description field

The Description field provides additional details for various items in the Register Details window. Available details include register descriptions, bitfield descriptions, and register details. Use the [Text View pop-up menu](#) to choose from these available options. As you select different items in the Register Details window, the text in the Description field updates.

Revert button

Click the Revert button to change a modified value back to its original value.

WARNING! If you modify values and then click the [Write button](#), you cannot revert the modified values back to their original values.

Read button

Click the Read button to display the current register bit values in the [Register Value view](#).

Write button

Normally, modifying values in the [Register Value view](#) does not modify the corresponding values in the actual register. If you wish to transfer your modifications to the actual register, you must perform one additional step. Click the Write button to write the bitfield values shown in the [Register Value view](#) to the actual register.

WARNING!

After clicking the Write button, you cannot recover the original register data by clicking the [Revert button](#).

Reset Value button

Some registers include default bit values. To restore these default values, click the Reset Value button. If the selected register does not include default values, the Reset Value button is grayed out.

Text View pop-up menu

The Text View pop-up menu changes the information displayed in the [Description field](#). The following choices are available:

- **Auto**—This option automatically shows the most appropriate information for the selected register or bitfield. The debugger makes its choice from the other options described below.
- **Register Description**—This option shows information about the entire register. Typical descriptions include the register name and the meaning of its contents.
- **Bitfield Description**—This option shows information about the selected bitfield in the [Register Value view](#). Typical descriptions include the name of the bitfield and access permissions.
- **Register Details**—This option shows detailed information about the current register. Typical descriptions include register names, register values, and bit-value explanations.

Register Windows

A Register window displays register contents and allows you to edit those contents. Different types of Register windows are available for different build targets. This section describes the following Register windows:

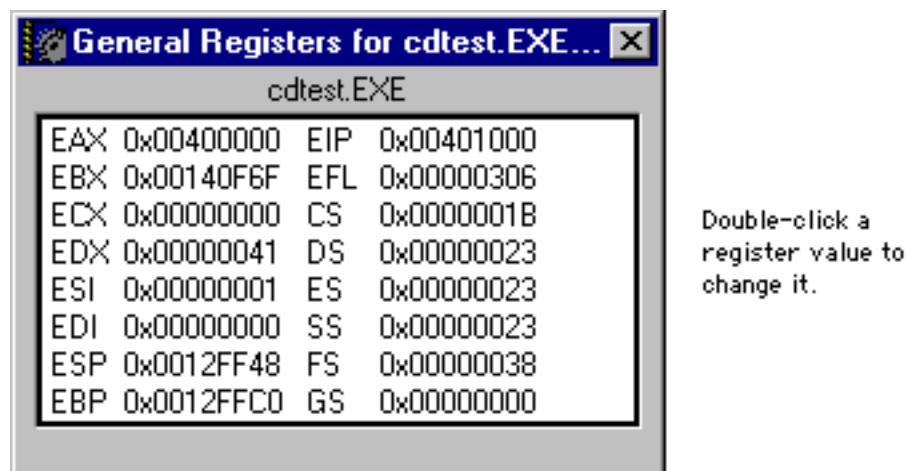
- [General Registers window](#)
- [FPU Registers window](#)
- [AltiVec Registers window](#)

General Registers window

The General Registers window ([Figure 11.26](#)) displays CPU registers and allows you to edit register contents. To open a General Registers window, choose **Window > Registers Windows > General Registers**.

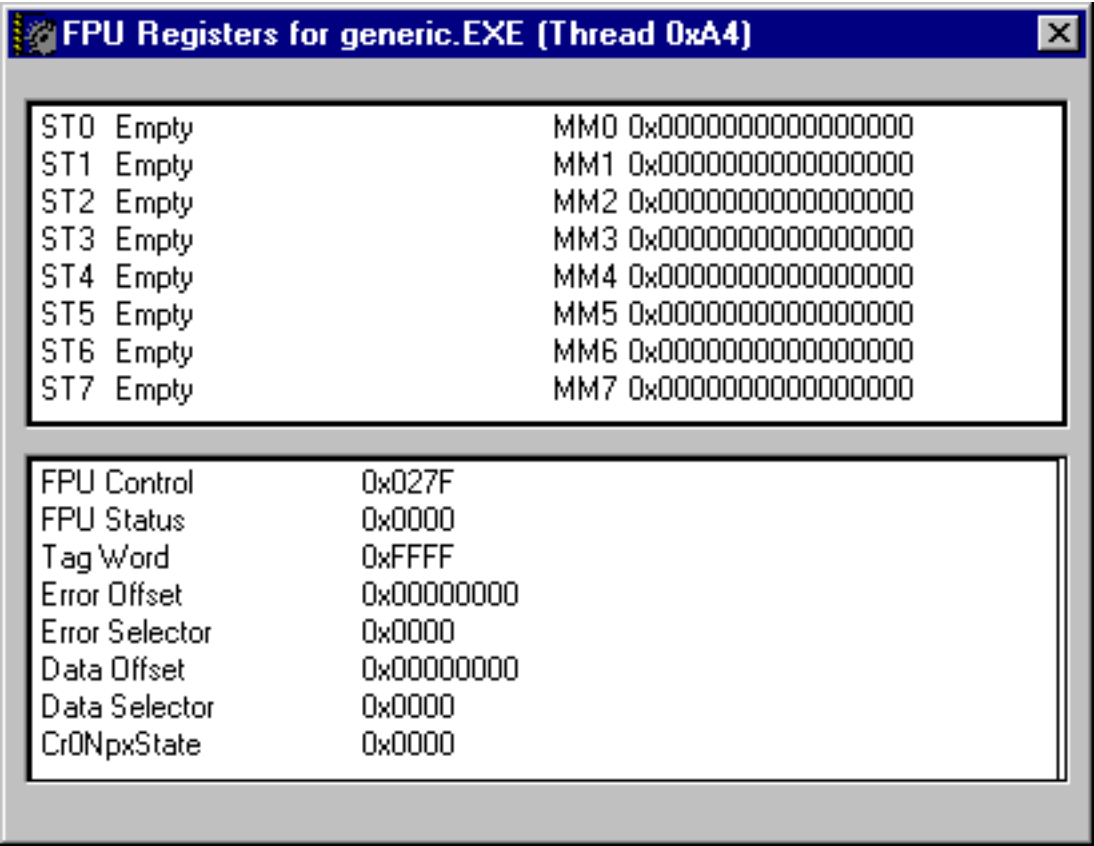
NOTE The appearance of the General Registers window depends on the build target. The build target also determines the commands in the Registers Windows submenu. If a submenu is unavailable, simply choose **Register Window** to see the window shown in [Figure 11.26](#).

Figure 11.26 General Registers window (Windows)



To change a register value, double-click the value or select the register and press Enter/Return. You can then type in a new value.

Figure 11.28 FPU Registers window (Windows)



To change a register value, double-click the value or select the register and press Enter/Return. You can then type in a new value.

WARNING!

Changing the value of a register is a very dangerous thing to do. The changes could corrupt your data and memory, or otherwise cause a crash.

Figure 11.29 FPU Registers window (Mac OS)

FPU Registers for ExamplePPC (Thread 0x64)			
ExamplePPC		FPSCR	
FP0	0.961215510650133	FP16	0.0000000000000000
FP1	-3.387775385466168e-20	FP17	0.0000000000000000
FP2	1.040349421040485	FP18	0.0000000000000000
FP3	-1.322259771485674e-20	FP19	0.0000000000000000
FP4	NaN	FP20	0.0000000000000000
FP5	1.0000000000000000e+14	FP21	0.0000000000000000
FP6	1.0000000000000000e+20	FP22	0.0000000000000000
FP7	1.0000000000000000e+13	FP23	0.0000000000000000
FP8	1.0000000000000000e+21	FP24	0.0000000000000000
FP9	1000000.0000000000000000	FP25	0.0000000000000000
FP10	1.0000000000000000e+07	FP26	0.0000000000000000
FP11	100000.0000000000000000	FP27	0.0000000000000000
FP12	1.0000000000000000e+08	FP28	0.0000000000000000
FP13	10000.0000000000000000	FP29	0.0000000000000000
FP14	0.0000000000000000	FP30	0.0000000000000000
FP15	0.0000000000000000	FP31	0.0000000000000000
		FX	1
		FEX	0
		VX	0
		OX	0
		UX	0
		ZX	0
		XX	1
		VXSNAN	0
		VXISI	0
		VXIDI	0
		VXZDZ	0
		VXIMZ	0
		VXVC	0
		FR	0
		FI	1
		FPRF	0x4
		VXCVI	0
		VE	0
		OE	0
		UE	0
		ZE	0
		XE	0
		RN	0x0

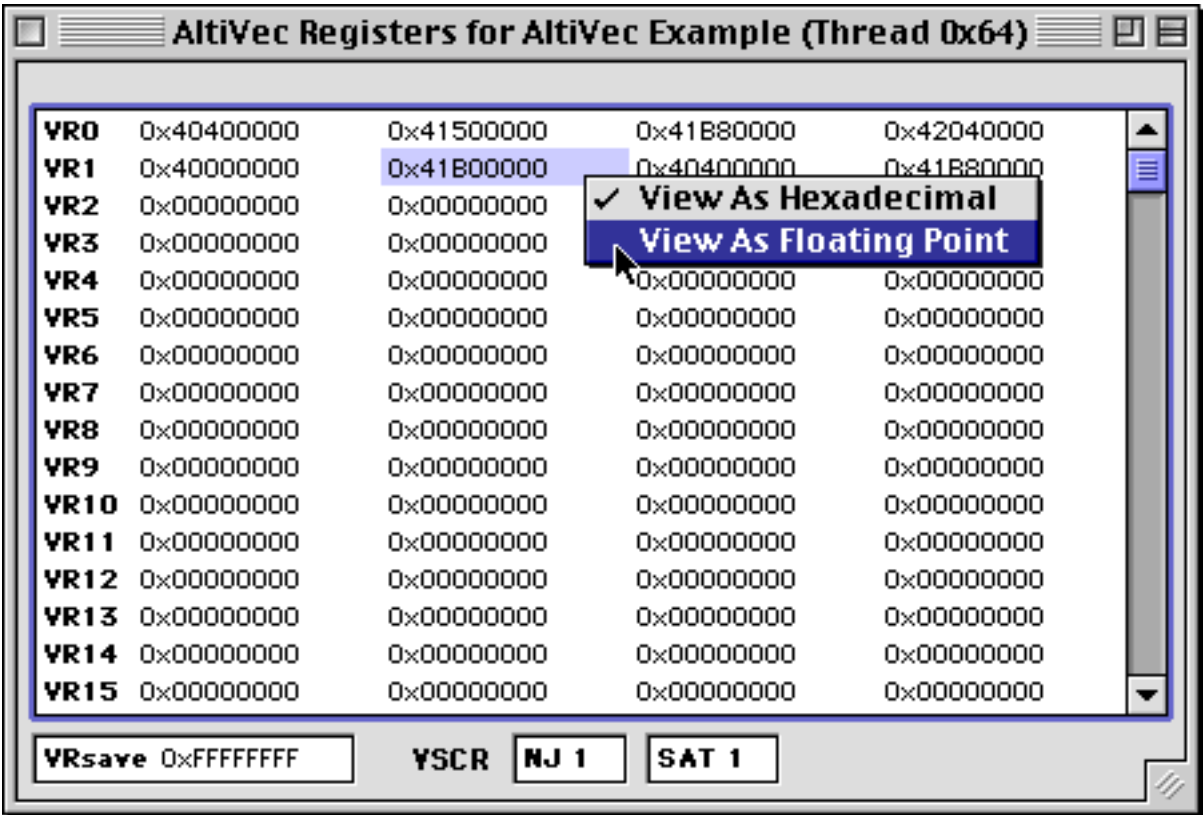
AltiVec Registers window

The AltiVec Registers window ([Figure 11.28](#)) displays AltiVec registers and allows you to edit register contents. To open an AltiVec Registers window, choose **Window > Registers Windows > AltiVec Registers**.

To change a register value, double-click the value or select the value and press Enter/Return. You can then enter a new value.

TIP Use the debugger contextual menu to change the format of the displayed data, as shown in [Figure 11.30](#). This technique is a convenient Data menu alternative.

Figure 11.30 **Altivec Registers window**



Log Window

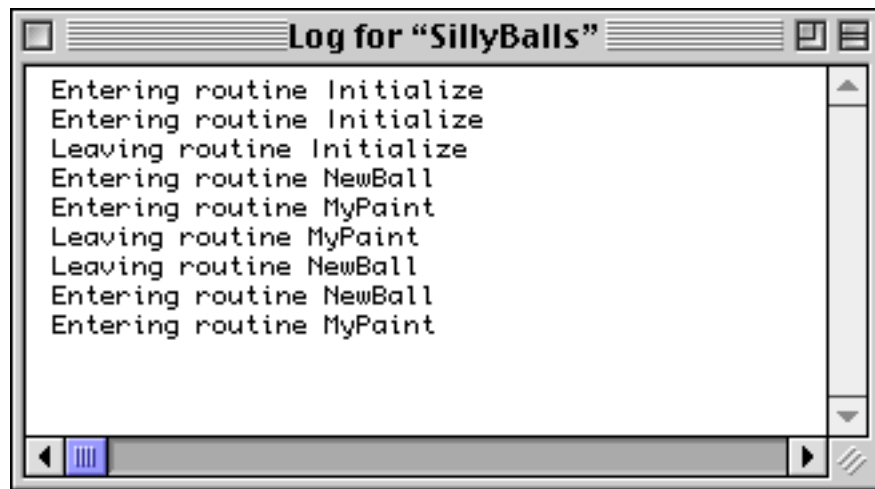
The Log window ([Figure 11.31](#)) displays messages as your program makes calls to system DLLs or starts new tasks.

To use the Log window, you must enable the **Log System Messages** preference in the Debugger Settings preference panel. See [“Debugger Settings” on page 357](#) for more information.

NOTE In Windows, log information includes messages about the loading and unloading of DLLs, as well as debug `printf()` messages. In

the Mac OS, log information refers to messages about the loading of PowerPC code fragments, as well as `DebugStr()` messages.

Figure 11.31 Log window



You can copy selected text from the Log window with the **Copy** command in the Edit menu. You can also use the **Save As** command in the File menu to save the Log window contents to a text file for later analysis.

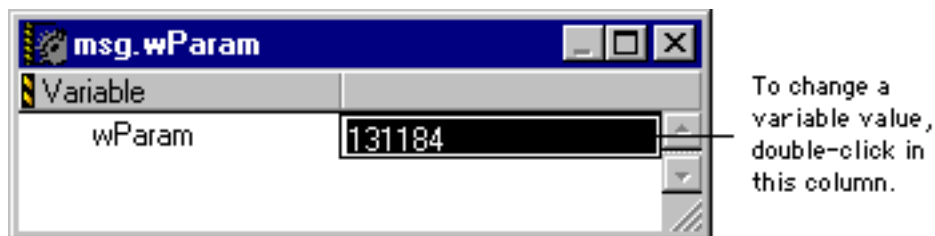
Windows If you choose to save the log to a text file, be sure to add an appropriate extension to the file name, like this: "log.txt"

TIP (Mac OS) To learn how to print out the messages in the Log window, see *Targeting Mac OS*.

Variable Window

The Variable window ([Figure 11.32](#)) displays a single variable and allows you to edit its contents. A Variable window containing a local variable closes on exit from the routine in which the variable is defined.

Figure 11.32 Variable window



Array Window

The Array window ([Figure 11.33](#)) displays a contiguous block of memory as an array of elements and allows you to edit the contents of those elements. To open the Array window, select an array variable in a Variables pane (either locals or globals) and then choose **View Array** from the Data menu.

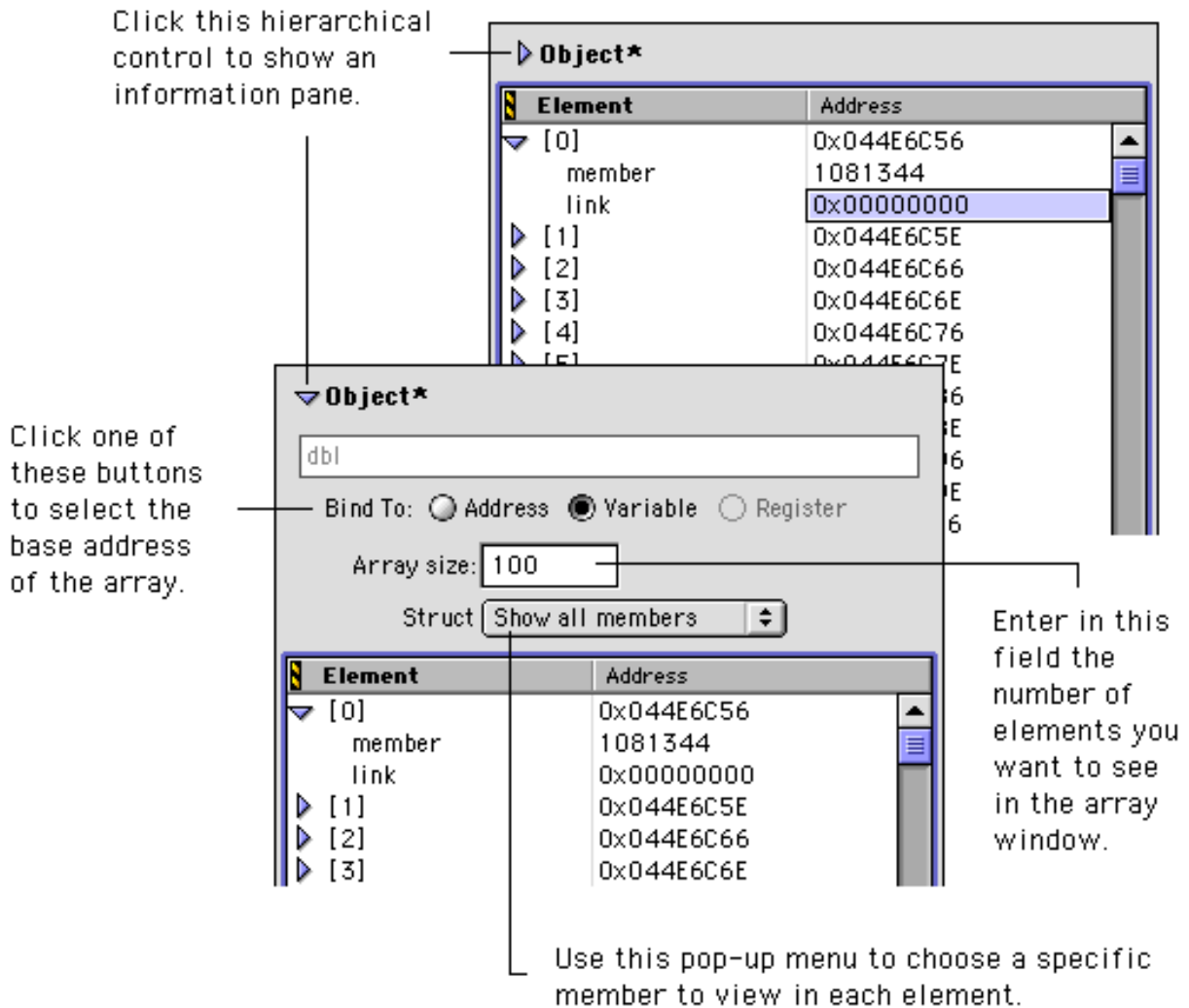
You can also use the **View Memory As** command in the Data menu to open an Array window. This command opens a dialog box that lets you select a data type, then opens an Array window interpreting memory as an array of that type. For more information, see [“View Memory As...” on page 643](#).

The Array window title bar describes the base address to which the array is bound. A base address can be bound to an address, a variable, or a register. Dragging a register name or variable name from a Variables or Registers pane to an Array window sets the array address. An array bound to a local variable closes when the routine in which the local variable is defined returns to its caller.

The Information pane shows the data type of the array elements, along with the base address of the array. To view more information about the array, click the hierarchical control in the Information pane. From the expanded Information pane, you can select the base address, the array size, and specific members to view if the array elements are of a structured type.

The Array window lists array contents sequentially, starting at element 0. If array elements are cast as structured types, a hierarchical control appears to the left of each array element, allowing you to expand or collapse the element.

Figure 11.33 Anatomy of an Array window

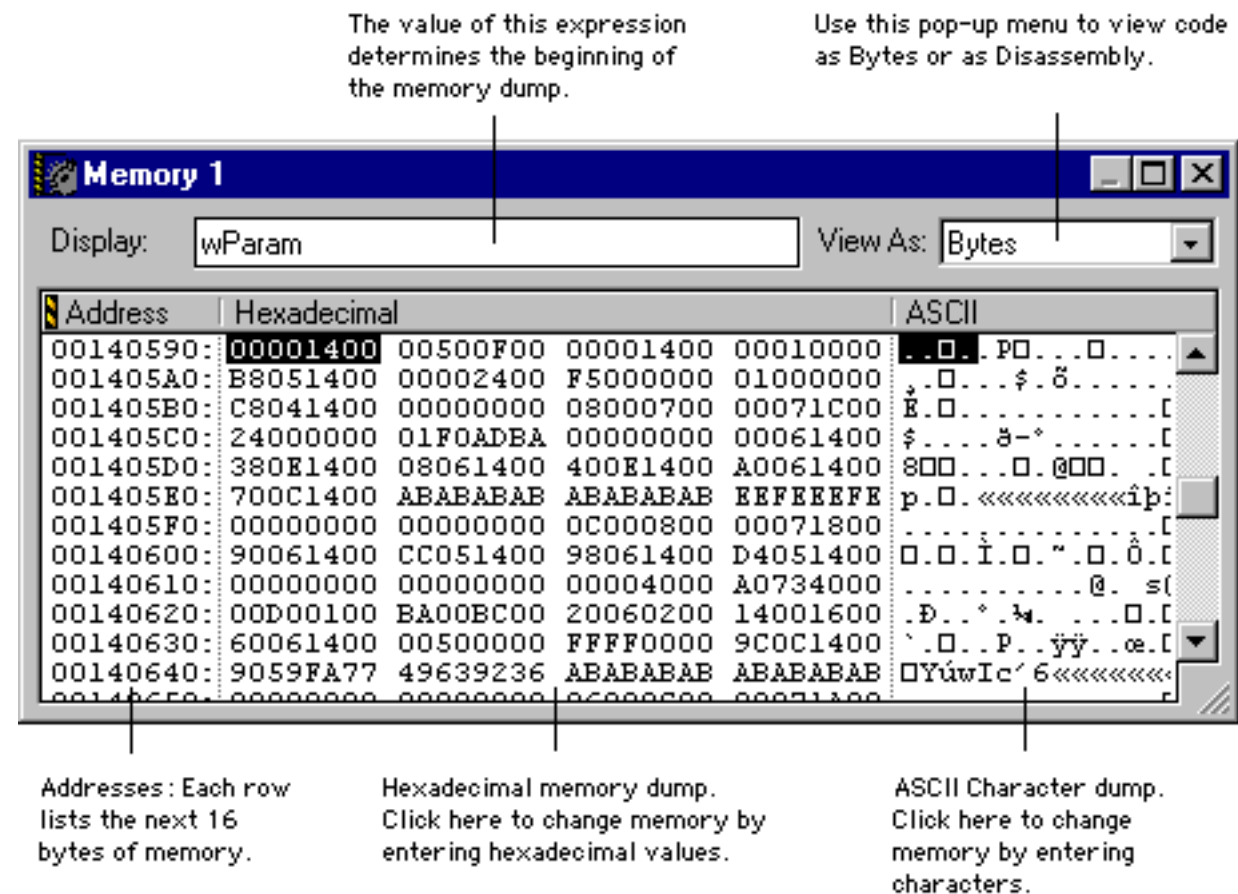


Memory Window

The Memory window displays the contents of memory in hexadecimal and corresponding ASCII character values ([Figure 11.34](#)). To open a Memory window, select a variable, routine name, or expression representing the desired base address in the Stack Crawl, Source Code Browser, or Expressions windows and choose **View Memory** from the Data menu.

NOTE The **View Memory As** command opens an Array window (see [“Array Window” on page 436](#)) displaying memory as an array of data of a type you specify.

Figure 11.34 Memory window



The source of the base address (which can be a variable, a routine, an expression, or a raw address like 0xC4F64C) appears at the top of the window. The Memory window appears blank if the IDE cannot access the base address.

To change the base address, enter a new expression in the **Display** field. You can also drag an expression to the field. If the expression does not produce an object in memory (an lvalue), then the base address uses the expression value. For example, the Memory window expression

`PlayerRecord`

shows memory beginning at the address of `PlayerRecord`.

If the expression result is an lvalue, then the base address uses the object address. For example, the expression

`*myArrayPtr`

shows memory beginning at the address of the object pointed to by `myArrayPtr`.

You can use a Memory window to change the values of individual bytes in memory. Click in the displayed data to select a starting point and start typing. If you select a byte in the hexadecimal display, you are restricted to typing hexadecimal digits. If you select a byte in the ASCII display, you can type any alphanumeric character. Certain keys (such as Backspace/Delete, Tab, Enter/Return, and so forth) do not work. New data you type overwrites the data already in memory.

Mac OS If the expression is a pointer-sized register, then the base address uses the register contents. For example, `@A0` (type Option-r for the first character) shows memory beginning at the address contained in the 68K register A0. Note that 64-bit floating-point registers cannot be used on a computer with a 32-bit central processing unit (CPU). The register size must have the size of the address bus of the CPU, so neither floating-point nor AltiVec registers can be used in the **Display** field.

WARNING!

Arbitrarily changing the contents of memory can be very dangerous to computer stability and can result in a crash. Be sure you understand the consequences of any changes you make.

Basic Debugging

This section describes the means by which the debugger helps you locate and solve problems in your source code. This section covers the following principal topics:

- [Starting Up](#)—things to note when starting the debugger

- [Running, Stepping, and Stopping Code](#)—controlling program execution one line at a time
- [Navigating Code](#)—moving around and finding code
- [Breakpoints](#)—stopping execution at specific points and times
- [Watchpoints](#)—stopping execution when the contents of a memory location changes
- [Viewing and Changing Data](#)—seeing your variables and modifying them at will
- [Editing Source Code](#)—editing source code while in a debugging session

This section assumes you are familiar with the debugger user interface, described in [“Guided Tour of the Debugger” on page 401](#). To learn how to prepare a build target for debugging, see [“Preparing for Debugging” on page 397](#). For information on setting debugging preferences, see [“Debugger Preferences” on page 283](#).

Starting Up

The steps you follow to start up the debugger depend on the build target you wish to debug. Usually, the debugger requires a project file and a symbolics file in order to successfully debug a project. Some embedded build targets do not require these files to start a debugging session.

Refer to [“General start-up process” on page 440](#) to learn about the typical start-up process for debugging a project. If you wish to debug an embedded build target, refer to [“Embedded start-up process” on page 443](#) and the *Targeting* manual appropriate for your particular build target.

General start-up process

To use the debugger, first open a project. Next, choose **Enable Debugger** from the Project menu. Then, choose the **Debug** command from the Project menu to debug the project.

When using the debugger with your code, you should pay careful attention to what happens. The following situations can occur:

- If you choose **Debug** from the Project menu to debug a project, the debugger opens the [Stack Crawl Window](#). Typically, the

debugger stops program execution at the first line of code in the `main` function. If you encounter this situation, all is well. Your program code is stopped and ready to run.

- If you open a symbolics file, the debugger opens the [Source Code Browser Window](#). In this case, you must choose **Debug** from the Project menu or click the Run button in the [Debugger toolbar](#). Then, the debugger launches the build target, takes control of that build target, brings the Stack Crawl window forward, and stops the program at the first line.

The debugger might ask you for the location of a particular file, as shown in [Figure 11.35](#) (Windows) or [Figure 11.36](#) (Mac OS).

You might see this dialog either upon startup (the debugger is looking for the file with the main entry point) or by clicking on a specific file in the [Source Code Browser Window](#).

The debugger might ask you to locate files in these situations:

- a file has been moved to a different directory
- you received the project from another person in your team and the paths are different
- you selected a file belonging to a compiled library and you do not have the source files

The last case is most common if you compile a build target with symbolics-file generation enabled for a library. This case is especially true with some of the libraries distributed with CodeWarrior.

Once you find the file, the debugger remembers the new location, even between debugging sessions.

For more information, see [“Generating Symbolics Information” on page 399](#).

Figure 11.35 Find the file (Windows)

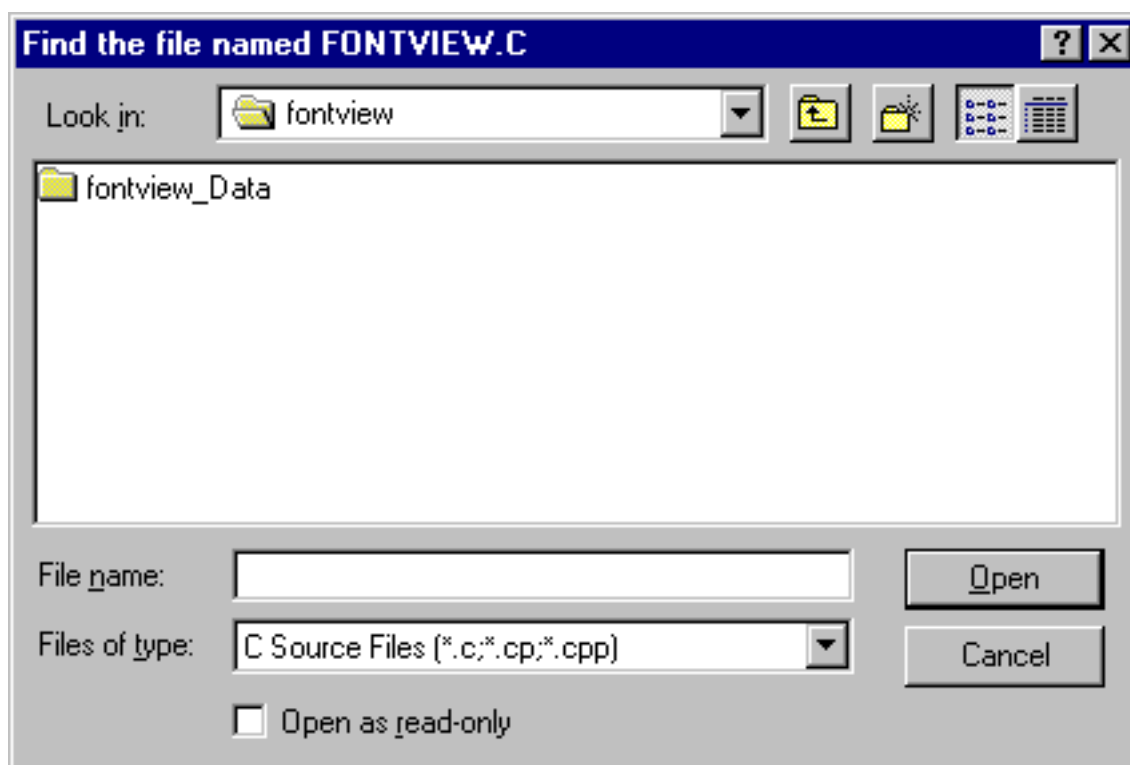
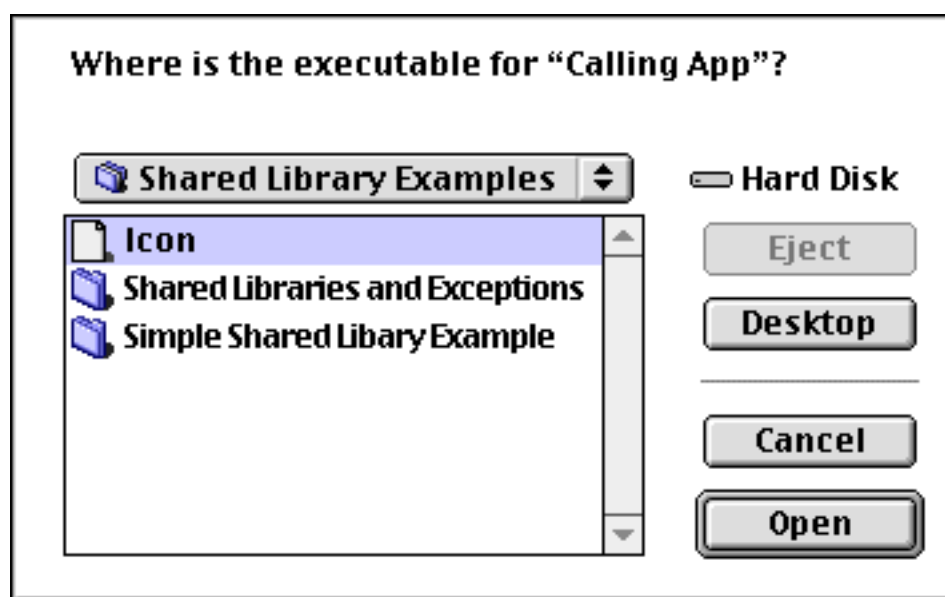


Figure 11.36 Where is the executable? (Mac OS)



Embedded start-up process

The typical debugging procedure described in [“General start-up process” on page 440](#) applies to most embedded projects as well. However, for some embedded build targets, you do not need to create a project file or generate symbolics information in order to debug a project. In such cases, the IDE creates a default project with default debugging settings. The debugger uses these default settings during the debugging session.

Additionally, the debugger lets you download and run an executable file without using symbolics information. For more information, refer to the *Targeting* manual appropriate for your particular build target.







Running, Stepping, and Stopping Code

This section explains how to run your code, step through it line by line, and stop or kill the build target when you want to stop debugging.

Moving through code line by line is often called “stepping” through code. It is a linear approach to navigation, where you start at the beginning and move steadily through the code. This concept is important for understanding code navigation. The debugger lets you navigate to any location directly, stop your code at specific locations when certain conditions are met, and view as well as modify your data.

To step through your code, you can use the debugger toolbar buttons, the keyboard, or choose the appropriate menu command. [Table 11.3](#) lists the debugger toolbar buttons along with their Debug menu commands and their default keyboard equivalents.

Table 11.3 **Debugging buttons and their default keyboard equivalents**

Button	Menu Command	Windows Keyboard Equivalent	Mac OS Keyboard Equivalent	Solaris Keyboard Equivalent
	Run	F5	Command-R	Command-R
	Stop		Control-P	Control-P
	Kill	Shift-F5	Control-K	Control-K
	Step Over	F10	Control-S	Control-S
	Step Into	F11	Control-T	Control-T
	Step Out	Shift-F11	Control-U	Control-U

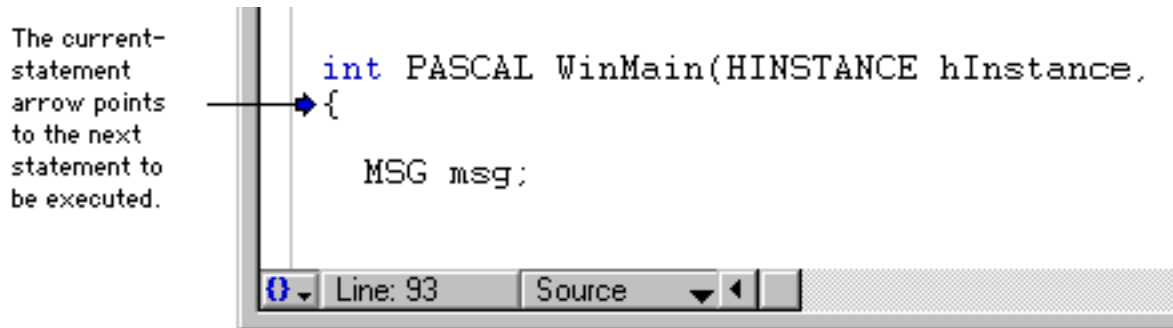
This section discusses the following topics:

- [Current-statement arrow](#)
- [Running your code](#)
- [Stepping through a single line](#)
- [Stepping into routines](#)
- [Stepping out of routines](#)
- [Skipping statements](#)
- [Stopping execution](#)
- [Killing execution](#)

Current-statement arrow

The current-statement arrow in the Stack Crawl window ([Figure 11.37](#)) indicates the next statement to be executed. It represents the processor program-counter register. When you start a debugging session, the current-statement arrow points to the first line of executable code in your program.

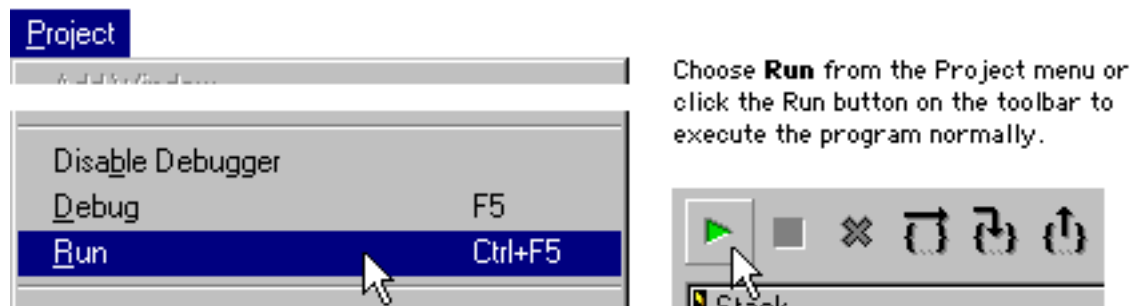
Figure 11.37 The current-statement arrow



Running your code

If the program is running but its execution has stopped, use the **Run** command (Figure 11.38) to restart your program. The program resumes execution at the current-statement arrow.

Figure 11.38 The Run command



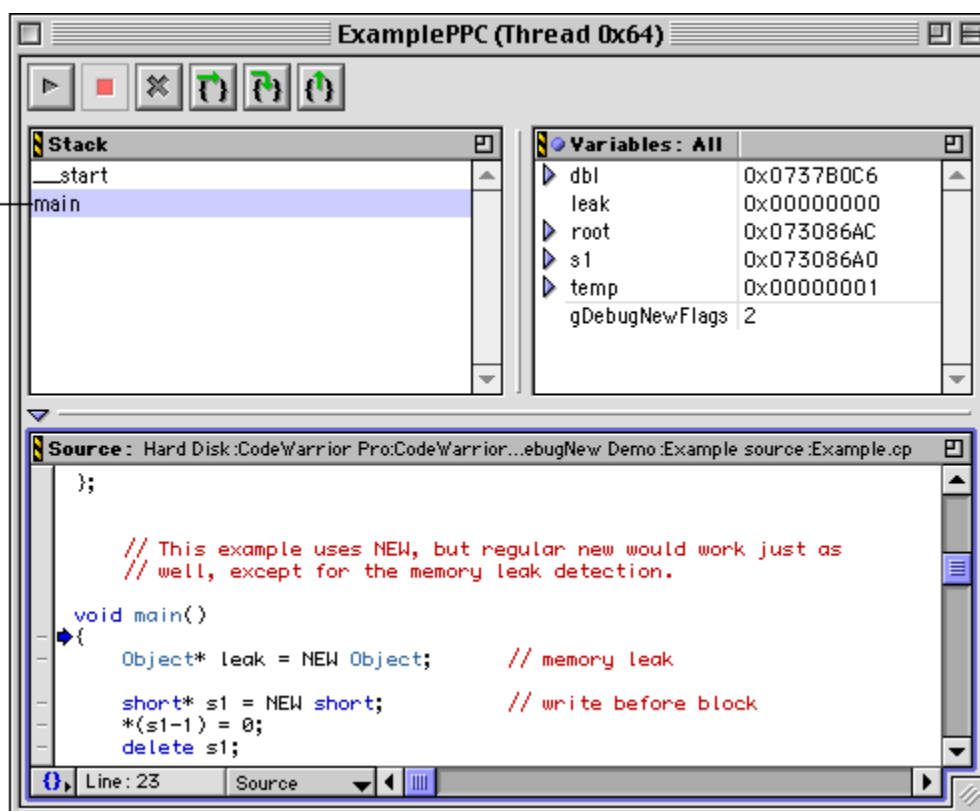
After a breakpoint or a **Stop** command, the debugger regains control and displays the Stack Crawl window with the current-statement arrow and the current values of variables. The debugger places an implicit breakpoint at the main entry point of the program and stops there (Figure 11.39). Issuing another **Run** command resumes program execution from the point of the interruption. After a **Kill** command, **Run** restarts the program from its beginning.

TIP You can inhibit the automatic launch of the program by holding down the Alt/Option key while opening the symbolics file. You can also change the [Automatically launch applications when SYM file opened](#) preference (see [“Global Settings” on page 288](#)). One use

for this feature is to debug C++ static constructors, which are executed before entering the main routine of the program.

Figure 11.39 Starting the program's execution

When first launching a program from the debugger, execution automatically stops at the main entry point of the program.



Stepping through a single line

To execute one statement, use the **Step Over** command ([Figure 11.40](#)). If the statement is a routine call, the entire called routine executes and the current-statement arrow proceeds to the next line of code. In other words, the **Step Over** command executes a routine call without visiting the code in the called routine. This behavior holds true if the routine does not have breakpoints and does not call other routines. When you step over code and reach the end of a routine, the current-statement arrow returns to the routine caller.

Figure 11.40 The Step Over command



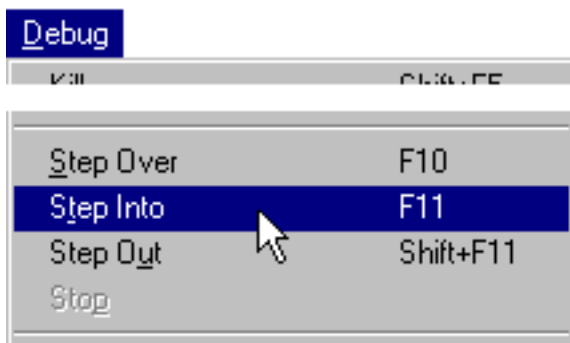
Choose **Step Over** from the Debug menu or click the Step Over button on the toolbar to execute the next statement. Routine calls are executed but not visited.



Stepping into routines

Sometimes you want to follow execution into a called routine. To execute one statement at a time and follow execution into a routine call, use the **Step Into** command ([Figure 11.41](#)).

Figure 11.41 The Step Into command



Choose **Step Into** from the Debug menu or click the Step Into button on the toolbar to execute the next statement, visiting routine calls.



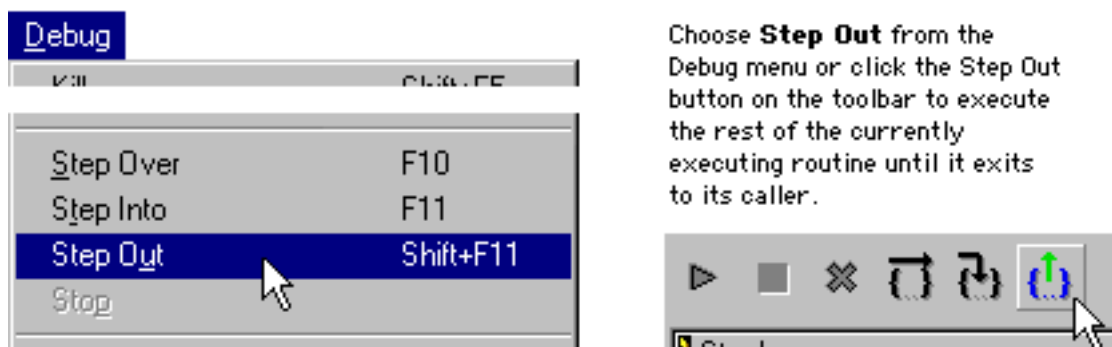
Step Into moves the current-statement arrow down one statement, unless the current statement contains a routine call. When **Step Into** reaches a routine call, the debugger follows execution into the routine being called.

Stepping out of routines

To execute statements until the current routine returns to its caller, use the **Step Out** command ([Figure 11.42](#)). **Step Out** executes the rest of the current routine normally and stops the program when the routine returns to its caller. You go one level back *up* the calling

chain. See [“Call-chain navigation” on page 453](#) for more information.

Figure 11.42 The Step Out command



Skipping statements

Sometimes you might want to skip statements altogether and not execute them at all. You can move the current-statement arrow to a different part of the currently executing source code by using the following techniques:

- [Manipulating the current-statement arrow](#)
- [Using the Change Program Counter dialog box](#)

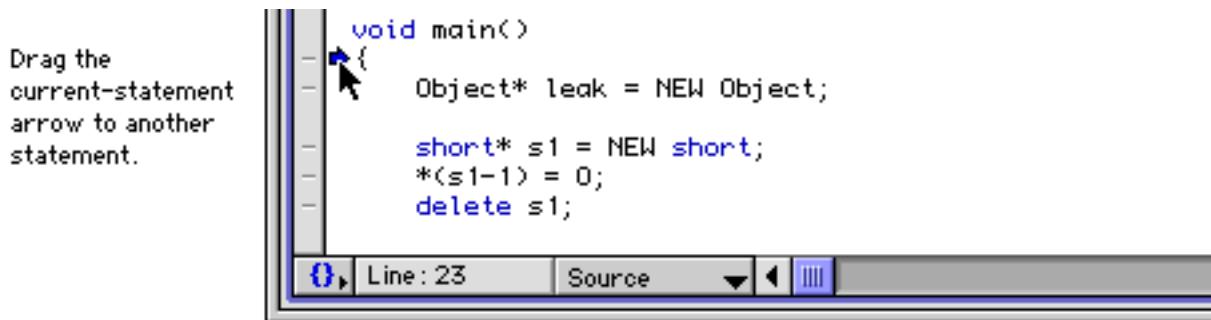
WARNING!

Skipping statements is equivalent to deliberately changing the program counter in a Register window. This change is very dangerous, because you might corrupt the stack by skipping routine calls and returns. The debugger cannot prevent you from corrupting your run-time environment.

Manipulating the current-statement arrow

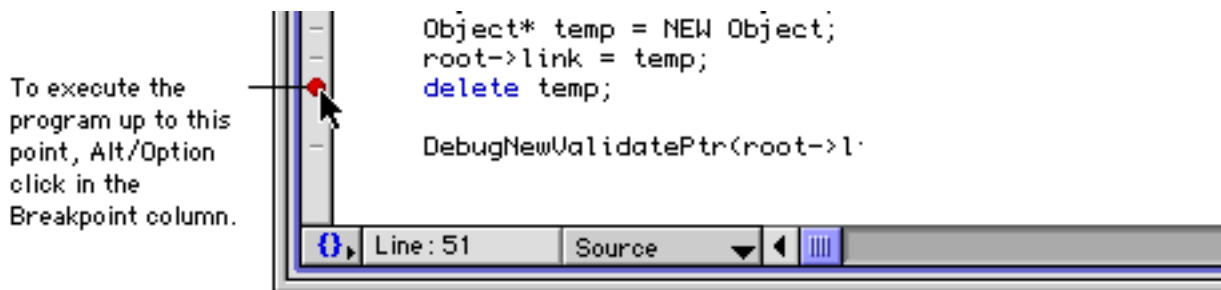
One way of skipping statements is to drag the current-statement arrow to a new position, as shown in [Figure 11.43](#). Note that dragging the current-statement arrow up or down *does not execute* the statements between the original arrow position and its new position.

Figure 11.43 Dragging the current-statement arrow



To move the current-statement arrow without potentially corrupting the run-time environment, Alt/Option click a statement in the Breakpoint column ([Figure 11.44](#)). Alt/Option clicking the statement sets a temporary breakpoint: Execution proceeds normally until the current-statement arrow reaches the temporary breakpoint, then stops. After execution stops, the debugger automatically removes the temporary breakpoint. For more information, see [“Breakpoints” on page 457](#).

Figure 11.44 Setting a temporary breakpoint



Dragging the current-statement arrow is convenient when you want to skip only a few statements at a time. However, if you want to skip several statements, the dragging technique becomes inconvenient. Instead of directly manipulating the current-statement arrow, you can also use the Change Program Counter key binding. The default key bindings for this command are as follows:

Windows Ctrl-Shift-F10

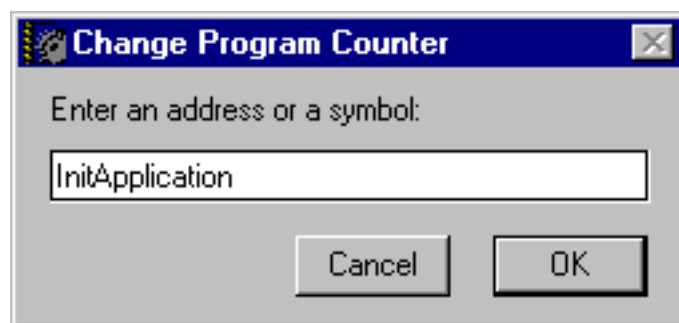
Mac OS, Solaris, Linux You can assign a custom key binding to this command. See [“Customizing the IDE” on page 296](#) for more information.

To use this command, click the line of source code to which you want to skip. Next, press the Change Program Counter key binding. If possible, the debugger moves the current-statement arrow to the line that you clicked. If the debugger cannot move the arrow to the specified line, an error message appears.

Using the Change Program Counter dialog box

If you want to move the current-statement arrow by specifying a new address or by specifying a symbol, use the Change Program Counter dialog box shown in [Figure 11.45](#). To open this dialog box, choose **Change Program Counter** from the Debug menu. Type in the field the address or symbol name to which you want to skip, then click **OK**. If possible, the debugger moves the current-statement arrow to the specified address or symbol. If the debugger cannot move the arrow to the destination you specified, or if the symbolics file does not contain the address or symbol that you entered, an error message appears.

Figure 11.45 **Change Program Counter dialog box**



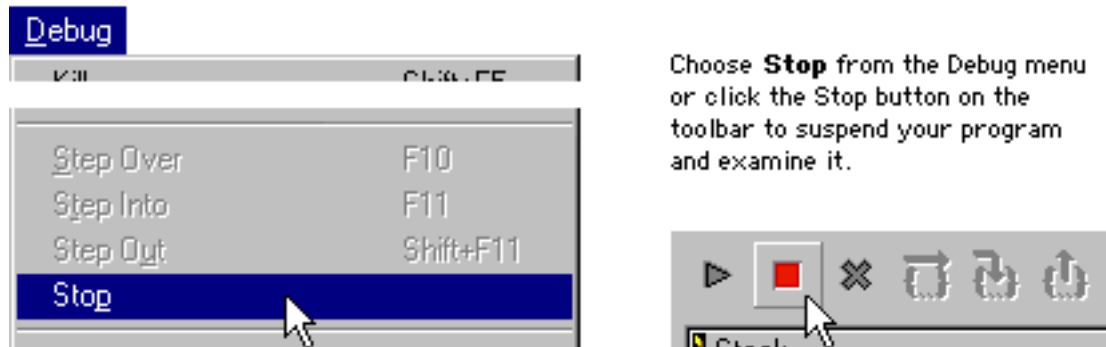
Stopping execution

While your program is running, you can use the **Stop** command ([Figure 11.46](#)) to suspend execution and explore your code with the debugger. You can then step through your code from that point, or use the **Run** command to resume execution.

Stopping in this manner is not very precise. Code executes very quickly, and there is no telling exactly where the debugger suspends your program when you issue the **Stop** command. Instead of stopping code, you can use breakpoints, which allow you to

suspend program execution at a specific point. To learn more, see [“Breakpoints” on page 457](#).

Figure 11.46 The Stop command



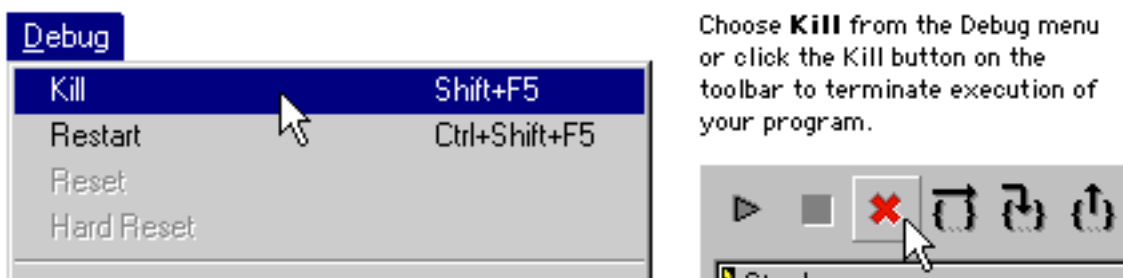
NOTE The **Stop** command is not available for some build targets because it is dependent on operating system services. For details on any particular build target, see the corresponding *Targeting* manual as described in [“Targeting Documentation” on page 27](#).

TIP (Mac OS) If your program hangs in an infinite loop, you can sometimes regain control by pressing Command-Control-/ on your keyboard. The debugger attempts to interrupt the program after you press this key binding. If the debugger successfully regains control, you can examine your code and remove the infinite loop.

Killing execution

Sometimes you want to terminate your program completely and end the debugging session. The **Kill** command ([Figure 11.47](#)) ends the program and returns control to the debugger. The Stack Crawl window tells you when the program is running, and to choose **Stop** from the Debug menu to stop its execution.

Figure 11.47 The Kill command



Killing a program is not the same as stopping a program. The **Stop** command suspends execution only temporarily: you can resume from the point at which you stopped execution. The **Kill** command permanently terminates a program.

Restarting execution

The **Restart** command achieves the equivalent of choosing the **Kill** command, followed by choosing the **Debug** command. Choose **Restart** to terminate the current debugging session and immediately begin a new debugging session.

Navigating Code

This section discusses the various ways you can move around in your code. This skill is vital when you want to set breakpoints at particular locations. The debugger lets you navigate code in the following ways:

- [Linear navigation](#)—stepping through code
- [Call-chain navigation](#)—moving to active routines
- [Source Code Browser window navigation](#)—moving to code in the Source Code Browser window
- [Changing font and color](#)—modifying font styles and text colors to make your code easier to read

Linear navigation

You can “walk” through your code by using the **Step Over**, **Step Into**, and **Step Out** commands as needed until you reach the place you want. This technique is useful for short stretches of code, but

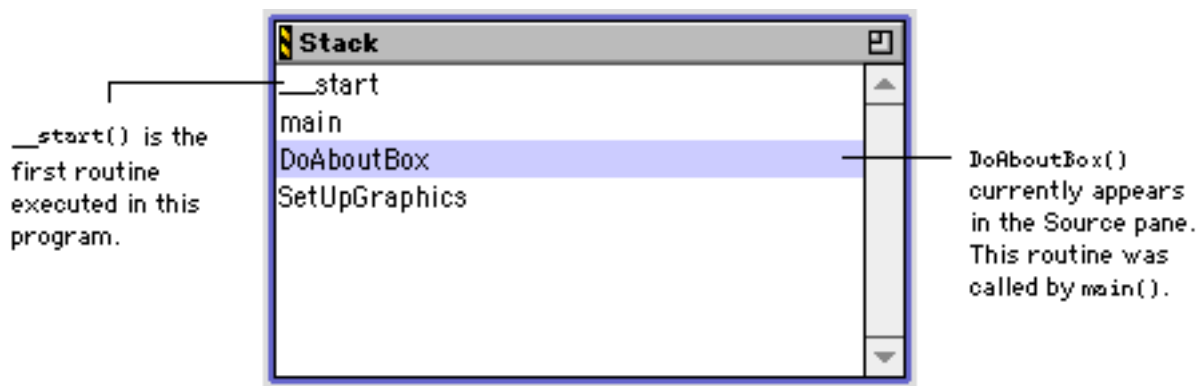
not very helpful when you want to get to a specific location several steps away.

[“Stepping through a single line” on page 446](#), [“Stepping into routines” on page 447](#), and [“Stepping out of routines” on page 447](#) describe various ways to linearly navigate your code.

Call-chain navigation

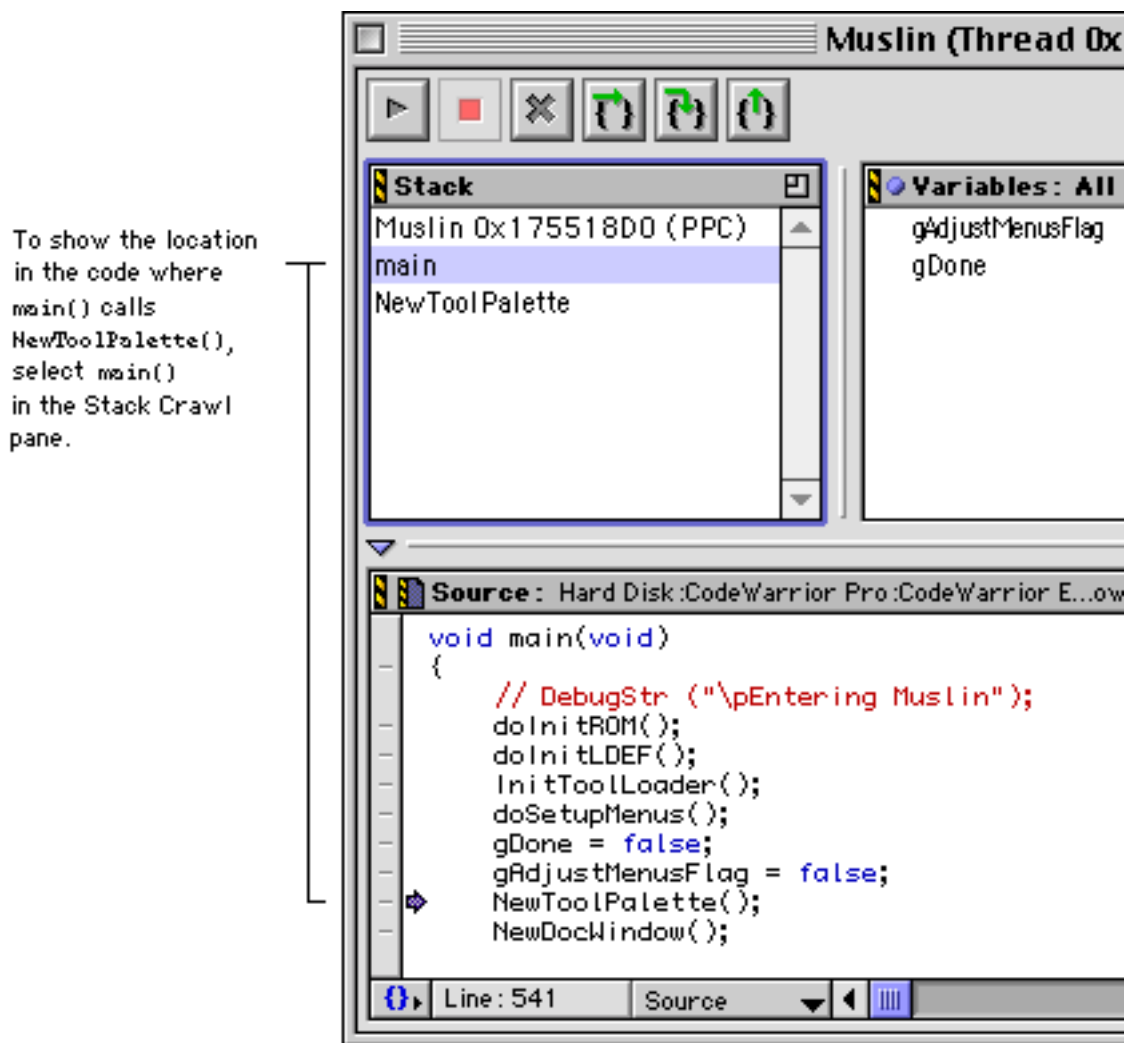
The Stack Crawl pane of the Stack Crawl window ([Figure 11.48](#)) shows routine call chain. Each routine in the chain appears below its caller, so the currently executing routine is at the bottom of the chain and the first routine to execute in the program is at the top.

Figure 11.48 The stack crawl pane



You can use the Stack Crawl pane to navigate to the routines that called the halted routine. To find the point where a routine in the Stack Crawl pane is called, click the name of the routine caller. The source code for the caller appears in the Source pane at the point of the call ([Figure 11.49](#)).

Figure 11.49 Finding the point where a routine is called



Source Code Browser window navigation

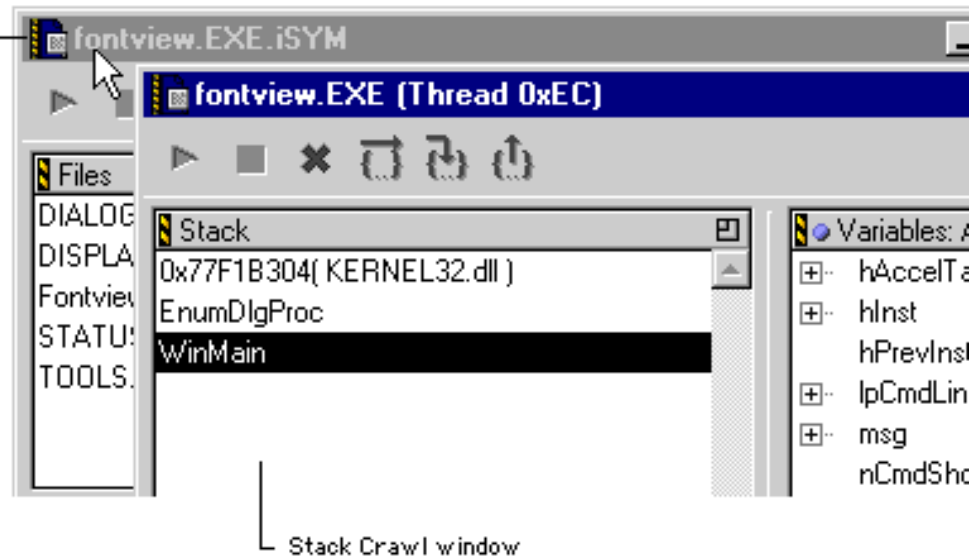
You can use the Source Code Browser window to jump to any location in your source code. The debugger displays this window only when you open a symbolics file. To view a specific routine, follow these steps:

1. Make the Source Code Browser window active ([Figure 11.50](#)).

Figure 11.50 Activating the Source Code Browser window

Click in the Source Code Browser window to make it active.

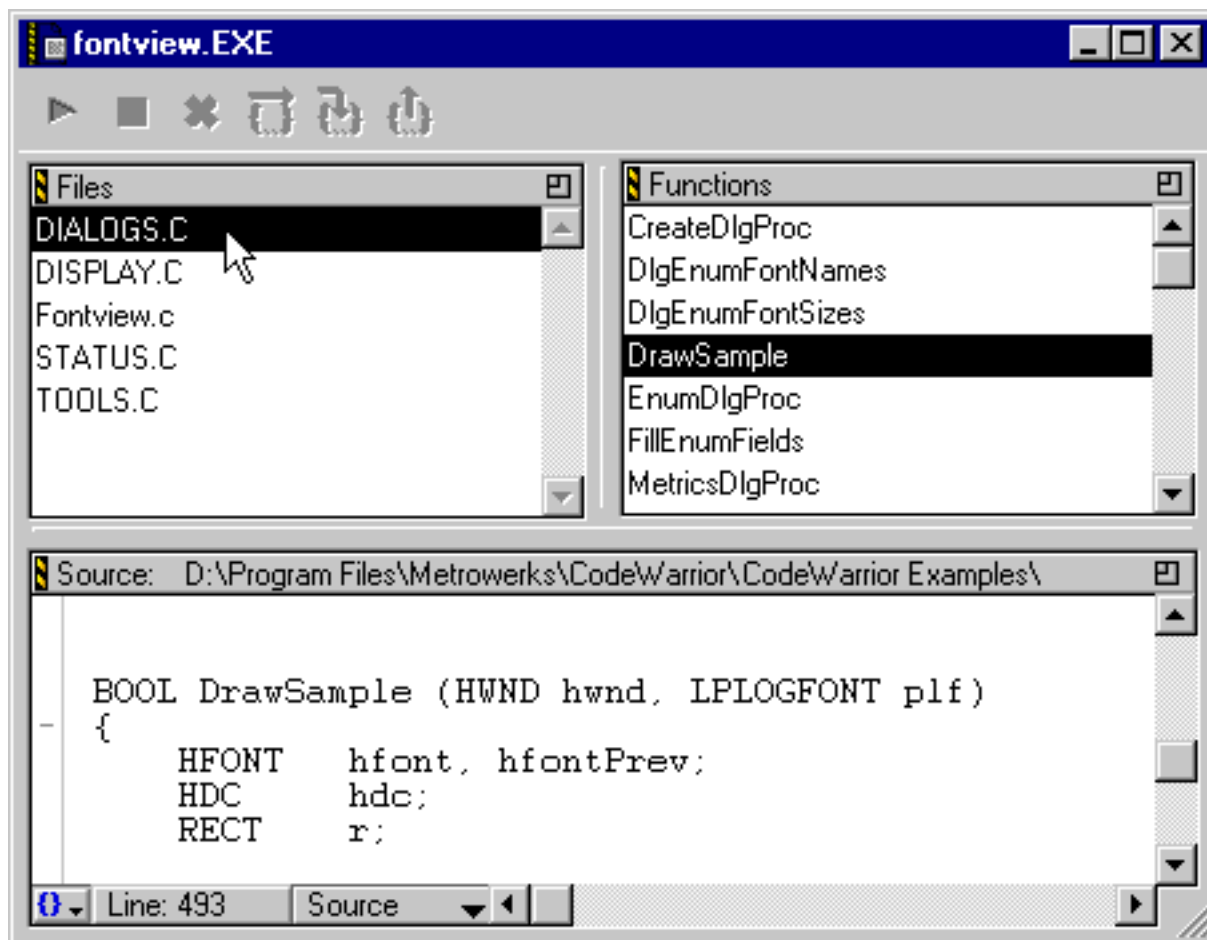
The Source Code Browser window lets you view any file in the project.



2. In the Source Code Browser window's Files pane, select the file where the routine is defined ([Figure 11.51](#)).

Simply click the desired file, or use the arrow keys to scroll through the list. The source code for that file appears in the Source pane. You can also type the name of the file.

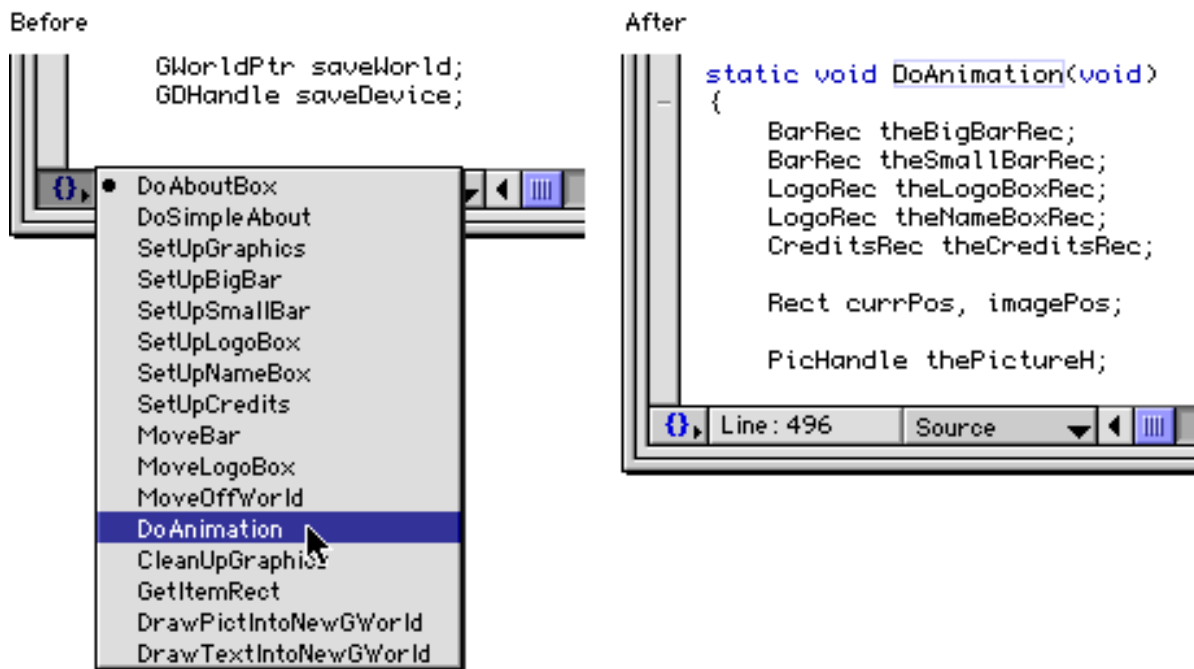
Figure 11.51 **Selecting a file to view its contents**



3. Locate the desired code in the source file.

Scroll through the Source pane to find the code you want. A more useful technique is to use the Source Code Browser window's Functions pane or Function pop-up menu to select the desired routine ([Figure 11.52](#)). The routine is displayed in the Source pane, allowing you to set and clear breakpoints. For more information, see ["Breakpoints" on page 457](#).

Figure 11.52 Choosing a routine to view



Changing font and color

The debugger displays source code in the font and color specified in the Editor preference panels of the IDE Preferences window.

For more information, see [“Editor Preferences” on page 272](#).

Breakpoints

A breakpoint suspends execution of a program and returns control to the debugger. When the debugger reaches a statement with a breakpoint, it stops the program before the statement is about to execute. The debugger then displays the routine containing the breakpoint in the Stack Crawl window. The current-statement arrow moves to the line containing the breakpoint and points to the next statement that is ready to execute.

This section discusses:

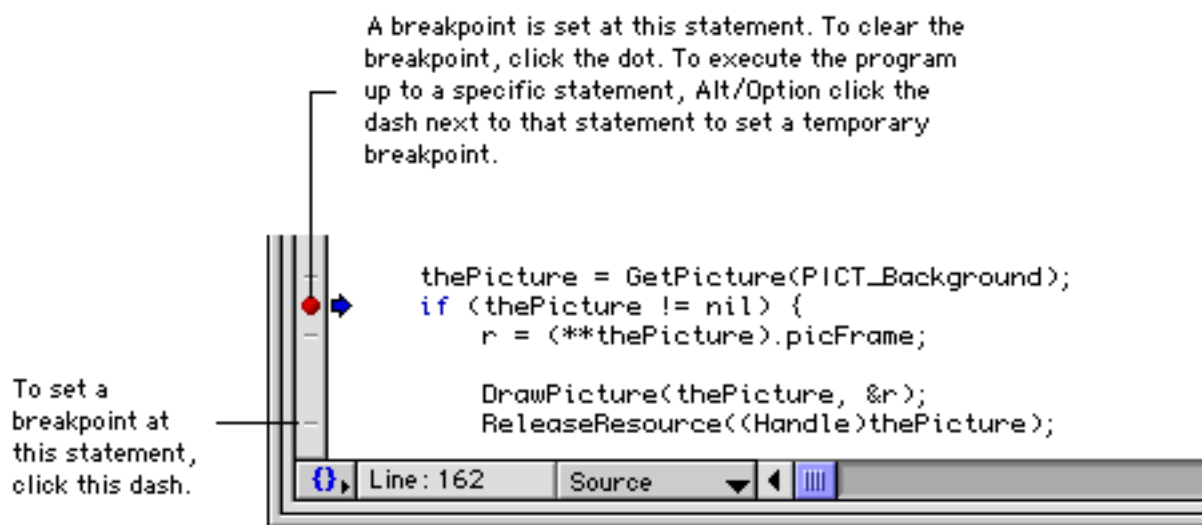
- [Setting and clearing breakpoints](#)
- [Temporary breakpoints](#)

- [Viewing breakpoints](#)
- [Conditional breakpoints](#)
- [Setting breakpoints for templated or redefined functions](#)
- [Impact of optimizing code on breakpoints](#)

Setting and clearing breakpoints

From the Source pane of the editor window, Stack Crawl window, or Source Code Browser window, you can set a breakpoint on any line with a dash marker—the short line to the left of a statement in the Breakpoint column (see [Figure 11.53](#)). The dash becomes a dot (on color monitors, the default dot color is red). This dot indicates that a breakpoint has been set at this statement. Execution will stop just before this statement is executed.

Figure 11.53 **Setting breakpoints**



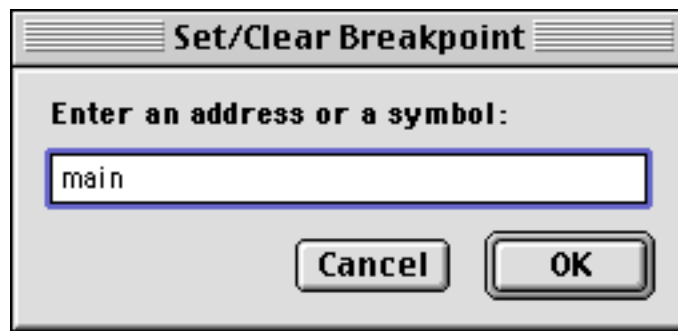
Another way to set a breakpoint is to click a particular line of source code in the Stack Crawl window and choose **Set Breakpoint** from the Debug menu.

A third way to set a breakpoint is to choose the **Set/Clear Breakpoint at** command from the Debug menu. The Set/Clear Breakpoint dialog box, shown in [Figure 11.54](#), appears. This dialog box lets you specify an address or symbol at which to set a breakpoint. After clicking **OK**, the debugger attempts to set a

breakpoint at the specified address or symbol. If the debugger cannot set the breakpoint at the address you specified, or if the symbolics file does not contain the address or symbol that you entered, an error message appears.

TIP The Set/Clear Breakpoint dialog box also lets you clear previously set breakpoints. Enter the address or symbol at which a breakpoint is currently set, and click **OK**. The debugger removes the breakpoint.

Figure 11.54 Set/Clear Breakpoint dialog box



To clear a single breakpoint, click the breakpoint dot in the Source pane. The dot turns back into a dash, indicating that you have removed that breakpoint.

You can manipulate breakpoints with the following commands in the Debug menu:

- **Clear Breakpoint**—removes the breakpoint from the current line
- **Enable Breakpoint**—makes the breakpoint on the current line active
- **Disable Breakpoint**—makes the breakpoint on the current line inactive
- **Clear All Breakpoints**—removes all breakpoints from the current program
- **Show Breakpoints**—displays the Breakpoint column when it is currently hidden

- **Hide Breakpoints**—removes the Breakpoint column when it is currently visible

In addition, you can use the debugger contextual menu in source code views and the Breakpoints window to choose the same menu commands. For more information, see [“Debugger Contextual Menu” on page 416](#).

TIP Put one statement on each line of code. This makes your code not only easier to read, but also easier to debug. The debugger allows only one breakpoint per line of source code, no matter how many statements a line contains.

Temporary breakpoints

Regular breakpoints stop a program’s execution each time you debug your project. However, you can also set temporary breakpoints that stop a program’s execution only once. When the temporary breakpoint is reached, the debugger stops the program and then removes the temporary breakpoint.

To set a temporary breakpoint, Alt/Option click the breakpoint dash to the left of the desired statement.

NOTE If there is already a regular breakpoint on a particular line, Alt/Option clicking removes the regular breakpoint, but the temporary breakpoint still works.

If the debugger encounters another breakpoint before reaching the temporary breakpoint, the program stops at the first breakpoint. The temporary breakpoint remains in place. After the temporary breakpoint is reached, it is triggered and then removed.

Viewing breakpoints

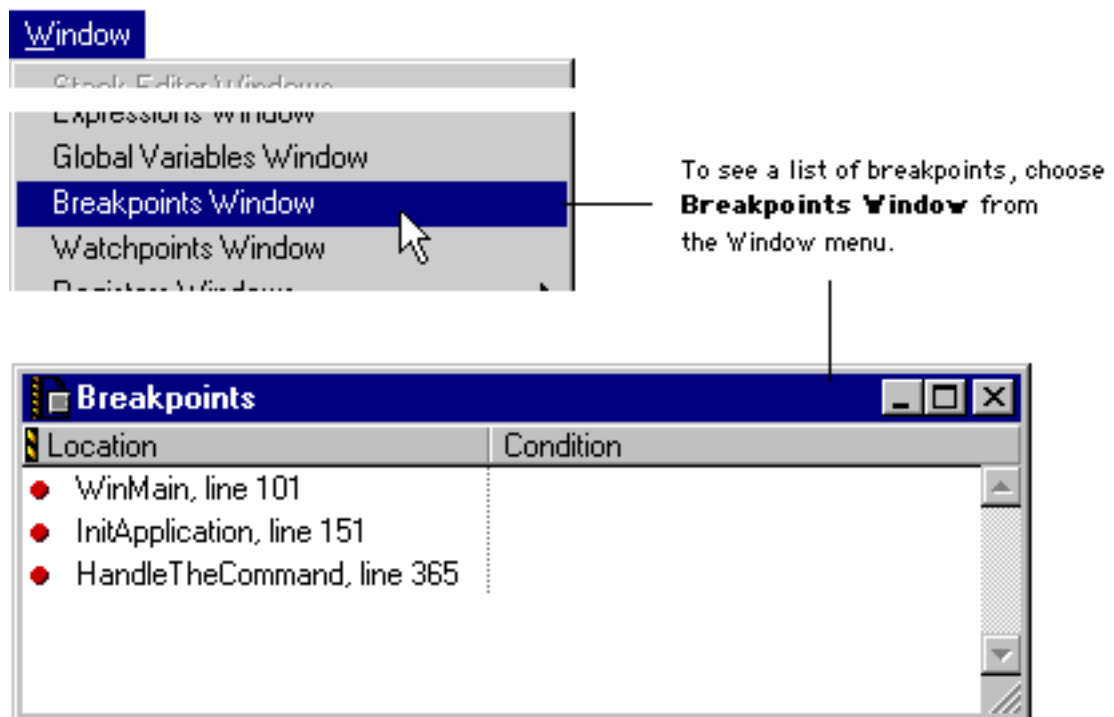
To see a list of all active breakpoints in your program, choose the **Breakpoints Window** command from the Window menu. The debugger displays a window that lists the source file and line number for each breakpoint ([Figure 11.55](#)). Clicking a breakpoint marker in the Breakpoints window toggles the breakpoint’s status between active and inactive. A dot indicates that the breakpoint is

active. The debugger stops the program's execution upon reaching an active breakpoint. A circle indicates that the breakpoint is inactive. The debugger continues to execute the program without stopping at an inactive breakpoint. The debugger remembers the position of inactive breakpoints in the program.

NOTE Double-clicking a breakpoint location in the Breakpoints window will take you to that line of code in the Stack Crawl window or Source Code Browser window.

For more information, see [“Breakpoints Window” on page 423](#).

Figure 11.55 Displaying the Breakpoints window



Conditional breakpoints

You can set conditional breakpoints that stop your program's execution at a given point only when a specified condition is met. A conditional breakpoint is an ordinary breakpoint with a conditional expression attached. If the expression evaluates to a true (nonzero) value when control reaches the breakpoint, the program's execution

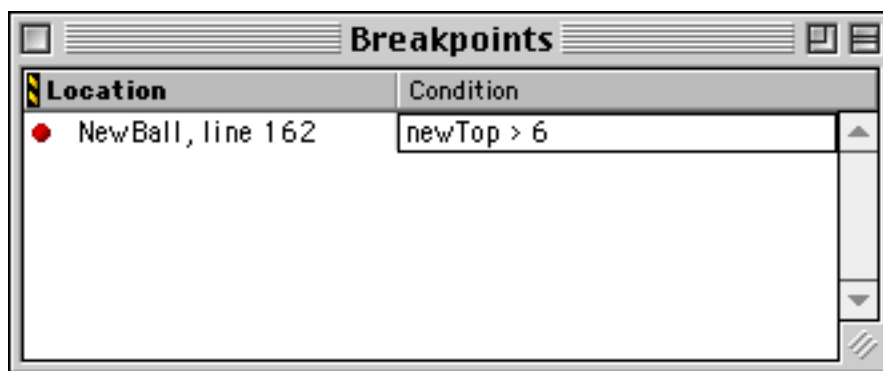
stops; if the value of the expression is false (zero), the breakpoint has no effect and program execution continues.

Conditional breakpoints are created in the Breakpoints window. To specify a conditional breakpoint:

1. **Set a breakpoint at the desired statement.**
2. **Display the Breakpoints window by choosing Breakpoints Window from the Window menu.**
3. **In the Breakpoints window, double-click the breakpoint's Condition field and enter an expression, or drag an expression from a source-code view or from the Expressions window.**

In [Figure 11.56](#), the debugger stops execution at line 162 in the `NewBall()` routine if and only if the variable `newTop` is greater than six.

Figure 11.56 Creating a conditional breakpoint



NOTE Conditional breakpoints are especially useful when you want to stop execution inside a loop, but only after going through several iterations of that loop. You can set a conditional breakpoint inside the loop that stops the program's execution when the loop index reaches the desired value.

Setting breakpoints for templated or redefined functions

You can set breakpoints for templated functions as well as functions that have been redefined.

For example, suppose you have a file named `file.h` that defines a function `void FOO(args)`, where `FOO` is a macro. If you create the lines of code shown in [Listing 11.1](#), a pop-up menu appears in the Breakpoints column during the debugging session. This pop-up menu lets you choose whether to set a breakpoint for `function1` or `function2`.

Listing 11.1 Example for templated and redefined functions

```
#define FOO function1
#include file.h
#define FOO function2
#include file.h
```

Impact of optimizing code on breakpoints

In order to accurately set breakpoints, the debugger relies on a direct correspondence between source code and object code. Optimizing your code can disrupt this relationship and cause problems when setting breakpoints.

If there is no breakpoint dash to the left of a line of source code, you cannot set a breakpoint at that line. The potential causes are:

- Symbolics information was disabled for that line.
- The routine containing the line was unused and was therefore deadstripped by the linker.
- The code has been optimized and the final object code no longer corresponds to the original source code.

The code optimization cause is the most common. For example, the PowerPC compiler lets you set a breakpoint when the start of a source statement corresponds to the start of a “statement block” that has at least one instruction in it. (You do not need to understand the term “statement block” in this example.)

Normally, when debugging information is enabled for a source file, the compiler tries to start a new statement block at each source statement that actually generates some code. For example, consider the source code in [Listing 11.2](#):

Listing 11.2 Sample code with breakpoints

```
- int i = 1;
- if (i)
    {                               //This is the third line.
        int k;
-     int j = 1;
-     i = j;
    }
```

Since the third line does not generate instructions, it does not have a unique object code address. Therefore, the debugger does not permit a breakpoint on that line.

After you start optimizing the code, the situation changes. For example, when you enable Instruction Scheduling in the project's [Global Optimizations](#) preference panel, the compiler no longer starts a new basic block for each source statement. The scheduler gains the maximum flexibility for reordering instructions within the block. The different instructions that correspond to a source statement are no longer consecutive. Instead, they are intermingled with the instructions from other source statements. Because of these optimizations, the debugger no longer displays breakpoint dashes as shown in [Listing 11.2](#).

When you enable additional optimizations, the source statements may not even appear in the generated code. For example, consider the following code:

Listing 11.3 Sample code with a loop

```
i = 0;
j = 0;
while (i < 10)
{
    j = j + 1;
    i = i + 1;
}
```

With minimal optimization, the compiler translates the code from [Listing 11.3](#) into the following equivalent code:

```
j = j + 1;           // duplicate 10 times
j = j + 1;
.
.
.
j = j + 1;
```

With even more optimization, the compiler totally eliminates instructions corresponding to the `while` loop and uses the following equivalent code:

```
j = 10;
```

While debugging your code, you should disable optimizations or use optimizations that are “debug safe.” Different optimizations are available for different build targets. See the appropriate *Targeting* manual for additional details, as described in [“Targeting Documentation” on page 27](#).

You can view the current optimizations for your project in the [Global Optimizations](#) preference panel of the Target Settings window. For more information, refer to [“Configuring Target Options” on page 321](#).

After setting a breakpoint, you can continue debugging the program. Refer to [“Running, Stepping, and Stopping Code” on page 443](#) for additional details.

Watchpoints

A watchpoint is a location or region of memory that you want the debugger to observe for you. Whenever a new value is written to that area of memory, the debugger suspends execution of the program and notifies you with an alert message on your screen ([Figure 11.57](#)). After dismissing the alert, you can examine the call chain, inspect or change variables, step through your code, or use any of the debugger’s other facilities. (In particular, from the debugger level, you *can* change the contents of the location that triggered the watchpoint without triggering it again.) Use the **Run** command (or the Run button on the debugger toolbar) to continue execution from the watchpoint.

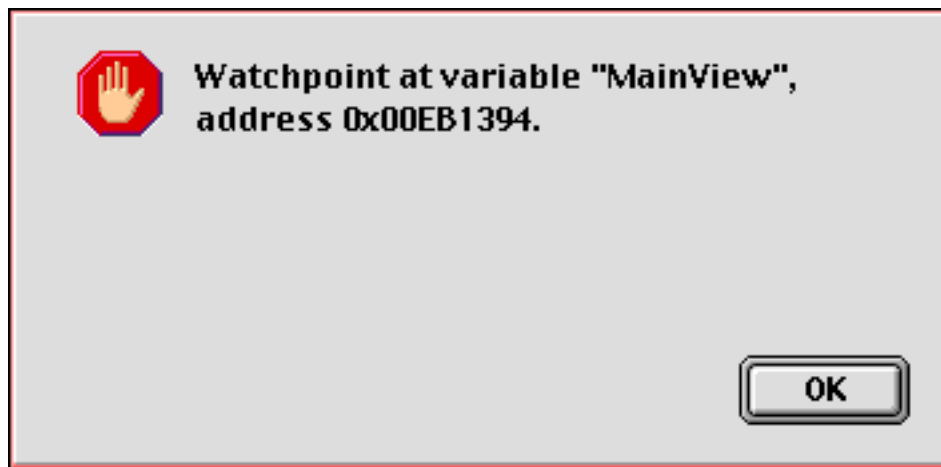
The following sections discuss watchpoints in more detail:

- [Setting and clearing watchpoints](#)
- [Viewing watchpoints](#)

NOTE (Mac OS) Watchpoints require System 7.5 or later, and will not work on 68K computers unless you enable virtual memory. Watchpoints are also known to be incompatible with the Speed Emulator portion of Speed Doubler, and possibly with RAM Doubler as well.

If you have a small program with a small set of global variables for 68K, the global data area is placed against the end of the stack. Since the debugger cannot detect watchpoints for variables on the stack, the global variables cannot be watched. One way to avoid this problem is to declare a global array of type char, about four kilobytes in size, to move the global variables away from the stack.

Figure 11.57 Watchpoint alert



Setting and clearing watchpoints

You can set a watchpoint in any of the following ways:

- Select a variable, in a Variable window or in the Source Code Browser window, and choose **Set Watchpoint** from the Debug menu.

- Drag a variable from another window into the Watchpoints window.
- Select a range of bytes in a Memory window and choose **Set Watchpoint** from the Debug menu.
- Invoke the debugger contextual menu ([Figure 11.16 on page 417](#)) for an appropriate variable or memory range and choose the **Set Watchpoint** command.

Variables or memory locations on which a watchpoint has been set are underlined in red in the Variable and Memory windows. To learn how to change this default color, refer to the section [“Syntax Coloring” on page 279](#).

WARNING!

There are some significant restrictions on where in memory you can place a watchpoint. You can use watchpoints only on global variables or on objects allocated from your application heap. You cannot set a watchpoint on a stack-based local variable or on a variable being held in a register

Mac OS You cannot set a watchpoint anywhere in low memory or the system heap. If you attempt to set a watchpoint in this area of memory, an alert box displays with the following text: “Could not set a watchpoint because the page containing that memory location overlaps low memory or the system heap.”

NOTE

(Mac OS) When debugging small 68K projects, you might see an alert box with the following text when trying to set watchpoints on a global variable: “Could not set a watchpoint at that location because it is on the stack.” This is a limitation of the classic 68K runtime architecture, not the debugger.

You can clear a watchpoint in any of the following ways:

- After triggering the watchpoint, choose the **Clear Watchpoint** command from the Debug menu.
- Select a variable, in a Variable window or in the Source Code Browser window, and choose **Clear Watchpoint** from the Debug menu.
- Select a range of bytes in a Memory window and choose **Clear Watchpoint** from the Debug menu.

- Select an existing watchpoint in the Watchpoints window and
 - Choose **Clear Watchpoint** from the Debug menu
 - Choose **Clear** from the Edit menu
 - Press the Backspace/Delete key
- Invoke the debugger contextual menu for a selected watchpoint and choose the **Clear Watchpoint** command.

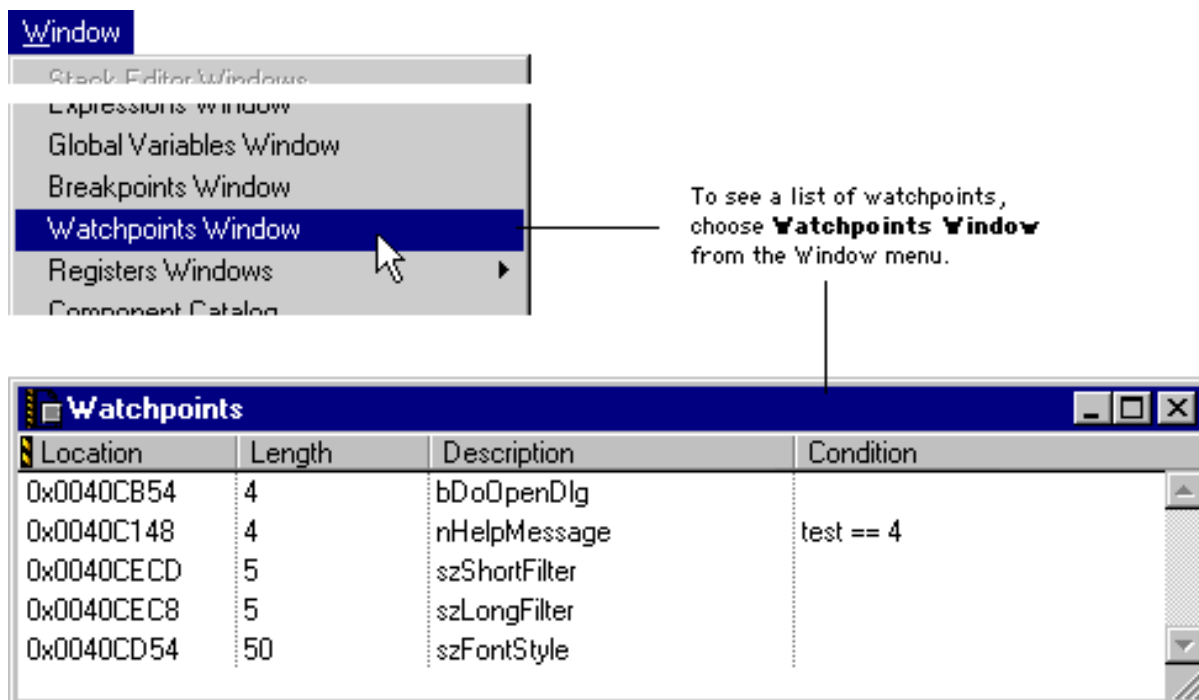
All watchpoints are automatically cleared when the program's execution terminates or is killed by the debugger.

Viewing watchpoints

To see a list of all watchpoints currently set in your program, choose the **Watchpoints Window** command from the Window menu. The debugger displays the Watchpoints window ([Figure 11.58](#)).

For more information, see [“Watchpoints Window” on page 424](#).

Figure 11.58 **Displaying the Watchpoints window**



Viewing and Changing Data

A critical feature of a debugger is the ability to see the current values of variables, and to change those values when necessary. This ability allows you to understand the details of your program's execution, and to experiment with new possibilities.

This section discusses the following topics:

- [Viewing local variables](#)
- [Viewing global variables](#)
- [Placing data in a new window](#)
- [Viewing data types](#)
- [Viewing data in a different format](#)
- [Viewing data as different types](#)
- [Changing the value of a variable](#)
- [Using the Expressions window](#)
- [Viewing raw memory](#)
- [Viewing memory at an address](#)
- [Viewing Processor Registers](#)

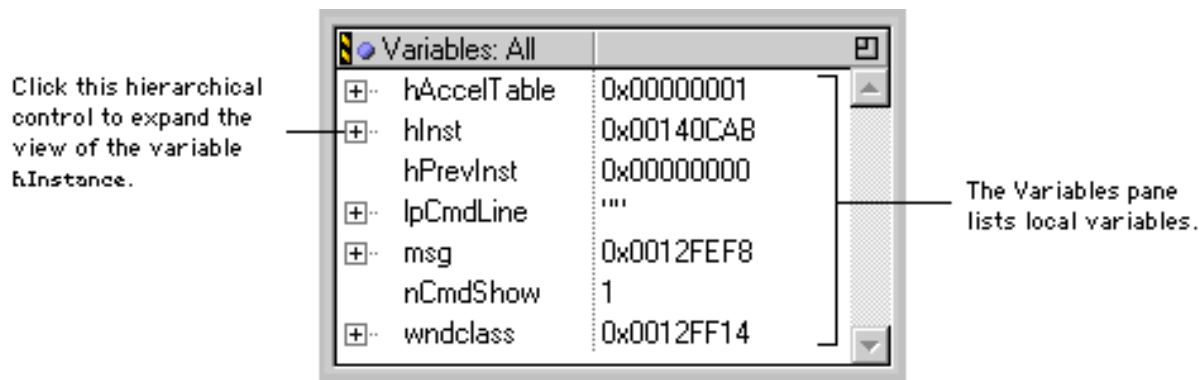
For additional information on viewing and changing data for a particular target, see the corresponding *Targeting* manual.

Viewing local variables

Local variables are displayed in the Variables pane of the Stack Crawl window ([Figure 11.59](#)). If the variable is a handle, pointer, or structure, you can click the hierarchical control to the left of the name to expand the view. This allows you to see the members of the structure, or the data referenced by the pointer or handle.

To learn some useful tips for using hierarchical controls, see [“Expanding and Collapsing Groups” on page 75](#). For more information about the Variables pane of the Stack Crawl window, refer to [“Variables pane” on page 404](#).

Figure 11.59 Viewing local variables

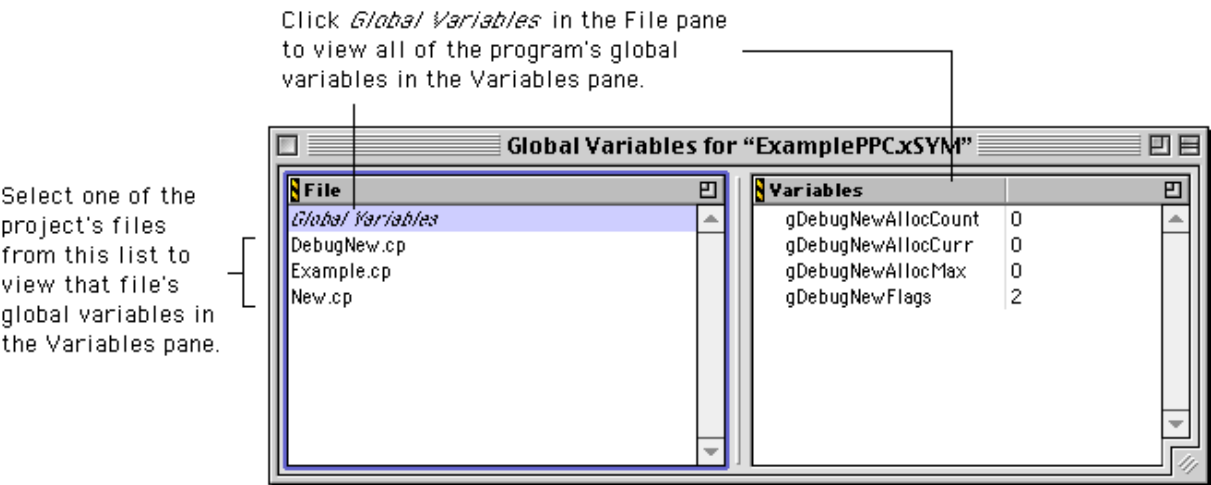


Viewing global variables

To view global variables, choose **Global Variables Window** from the Window menu. Next, click the *Global Variables* item in the File pane (Figure 11.60). Then, the Variables pane displays the global variables in your program.

For more information, refer to [“Global Variables Window” on page 421](#).

Figure 11.60 Viewing global variables



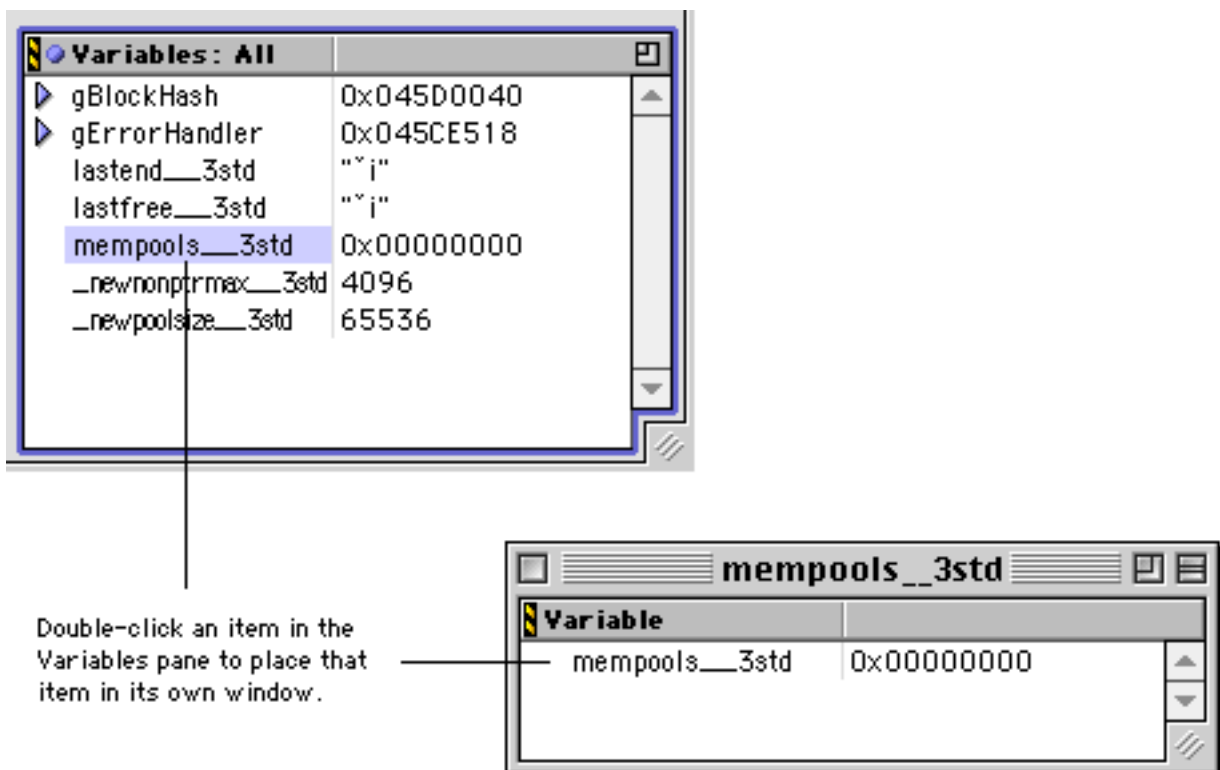
Placing data in a new window

Sometimes the Variables pane and Global Variables window are not the most convenient places to view data. You can place any variable or group of variables in a separate window.

To place a variable or memory location in its own window, double-click its name ([Figure 11.61](#)) or select the name and choose the **View Variable** command from the Data menu or the debugger contextual menu ([Figure 11.16 on page 417](#)). If the variable is an array, use the **View Array** command instead. To view the memory the variable occupies as a memory dump, use either the **View Memory** or **View Memory As** command.

For more information, see [“Variable Window” on page 435](#), [“Array Window” on page 436](#), and [“Memory Window” on page 437](#).

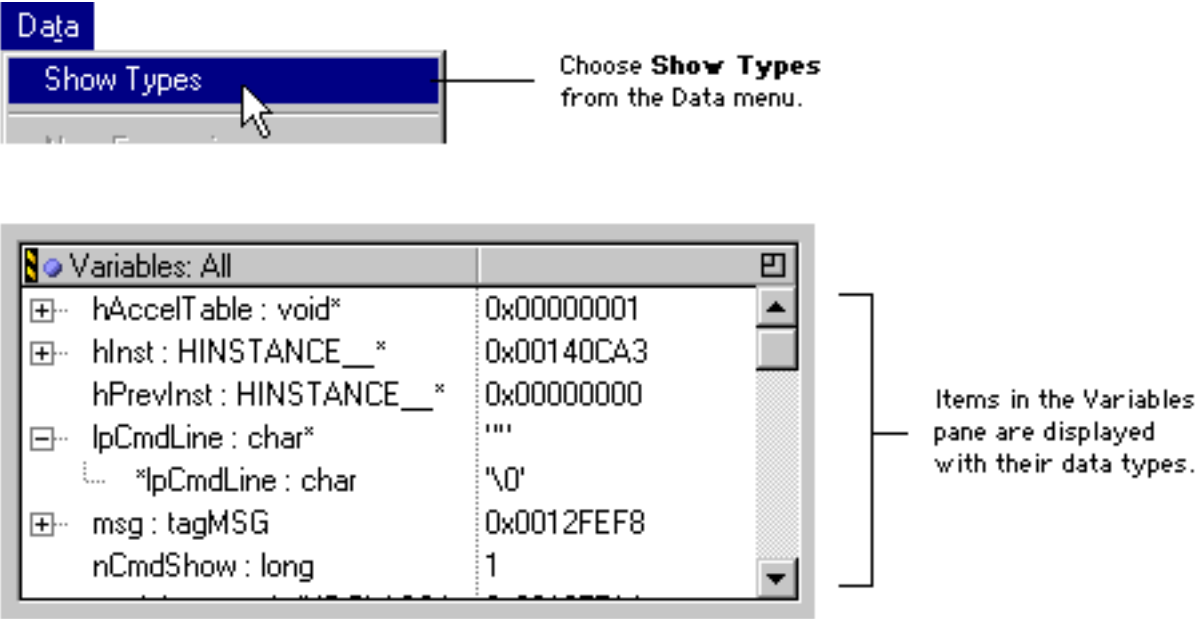
Figure 11.61 Placing a variable in its own window



Viewing data types

If you wish, the debugger can display the data types of variables on a window-by-window basis. Select the window or pane in which you want data types displayed and choose **Show Types** from the Data menu or the debugger contextual menu ([Figure 11.16 on page 417](#)). Then, the names of the variables and memory locations in that window or pane are followed by the relevant data type ([Figure 11.62](#)).

Figure 11.62 Viewing data types

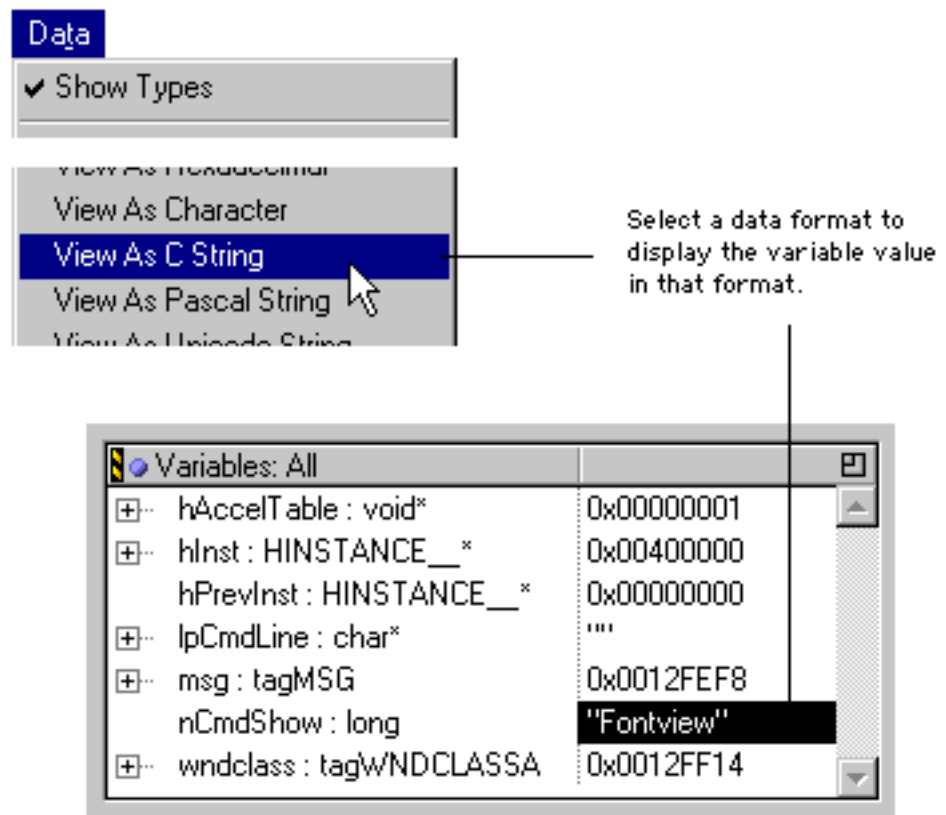


TIP To show data types automatically, enable the following preference in the Display Settings preference panel of the IDE Preferences window:

In variable panes, show variable types by default

See [“Display Settings” on page 283](#) for more information.

Figure 11.63 Selecting a data format



Viewing data in a different format

You display a variable's value in several formats:

- binary
- signed decimal
- unsigned decimal
- hexadecimal
- character
- C string
- Pascal string
- unicode string
- floating point
- enumeration

- Fixed
- Fract

To view data in a particular format, select either the name or the value of the variable in any window in which it is displayed, then choose the format you want from the Data menu ([Figure 11.63](#)) or the debugger contextual menu ([Figure 11.16 on page 417](#)).

Not all formats are available for all data types. For example, if a variable is an integral value (such as type short or long), you can view it in signed decimal, unsigned decimal, hexadecimal, character, or string format. However, you cannot view the same variable in floating-point, Fixed, or Fract format.

Viewing data as different types

The **View As** command in the Data menu lets you change the data type in which a variable, register, or memory value is displayed:

1. **Select the item in a window or pane.**
2. **Choose View As from the Data menu.**

Alternatively, you can use the debugger contextual menu ([Figure 11.16 on page 417](#)) to select the same command. After you select **View As**, the debugger displays the View As dialog box, shown in [Figure 11.64](#).

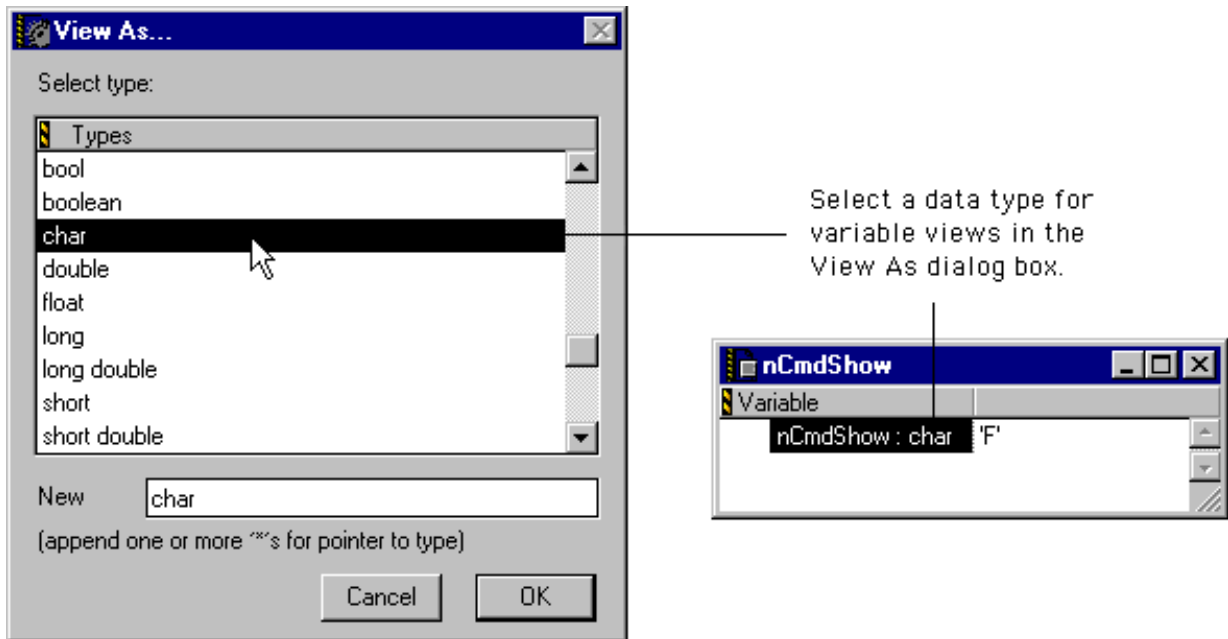
3. **Select the data type by which to view the item.**

The data type you select is displayed in the **New Type** text field near the bottom of the dialog box. If you want to treat the item as a pointer, append an asterisk (*) to the type name.

4. **Click OK.**

The display of the item's value changes to the specified type.

Figure 11.64 Selecting a data type



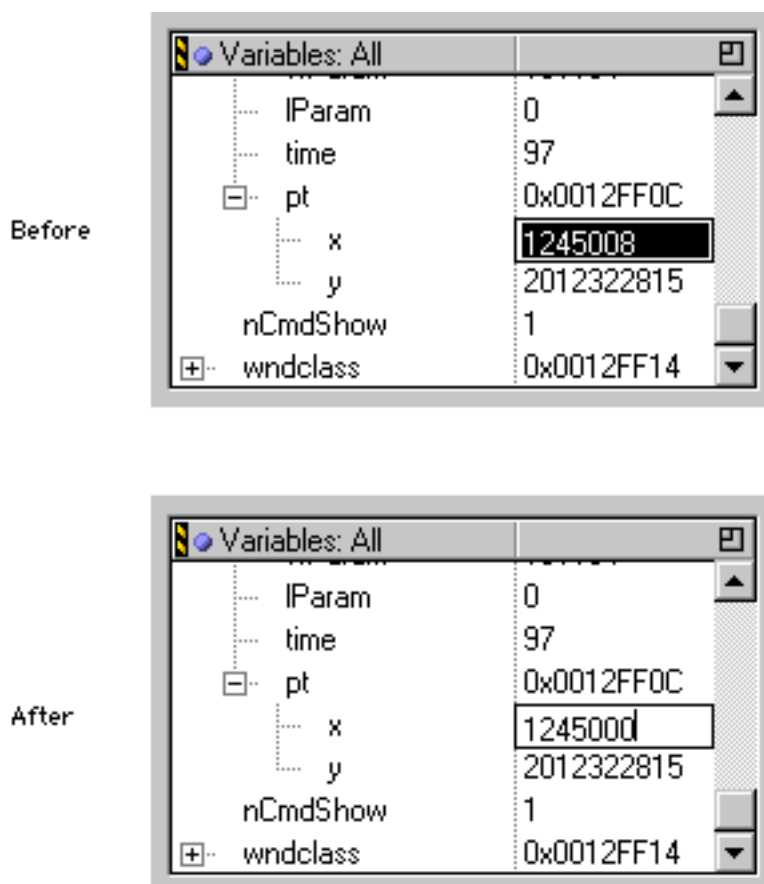
Changing the value of a variable

You can change the value of a variable in any variable view. Variables are displayed in several debugging windows: the Variables pane of the Stack Crawl window or Source Code Browser window, a Variable window, an Array window, or the Expressions window. To change the value, double-click the variable (or select it and press Enter/Return) and type the new value ([Figure 11.65](#)).

You can enter variable values in any of the following formats:

- decimal
- hexadecimal
- floating point
- C string
- Pascal string
- character constant

Figure 11.65 Changing a variable value



To enter a string or character constant, you must include C-style quotation marks (single quotes ' ' for a character constant, double quotes " " for a string). For Pascal strings, include the escape sequence `\p` as the first character in the string.

WARNING! Changing variable values can be dangerous. The debugger allows you to set a variable to any value of the appropriate data type. For example, you could set a pointer to `nil` and crash the machine.

Using the Expressions window

The Expressions window provides a single place to put frequently used local variables, global variables, structure members, array elements, and complex expressions. To open this window, choose

Expressions Window from the Window menu. You can add an item to the Expressions window by dragging and dropping that item from another window, or by selecting the item and choosing **Copy to Expression** from the Data menu or the debugger contextual menu ([Figure 11.16 on page 417](#)).

The contents of the Expressions window are updated whenever execution stops in the debugger. Any variable that is out of scope is left blank. You can take advantage of the Expressions window to perform a number of useful tasks:

- Move a routine's local variables to the Expressions window before expanding them to observe their contents. When the routine exits, its variables remain in the Expressions window and are still expanded when execution returns to the routine. The Expressions window does not automatically collapse local variables when execution leaves a routine, like the Variables pane in the Stack Crawl window.
- Keep multiple copies of the same item displayed as different data types, by using the **Copy to Expression** and **View As** commands in the Data menu and the debugger contextual menu.
- Keep a sorted list of items. You can reorder items by dragging them within the Expressions window.
- View local variables from calling routines. You don't have to navigate back up the calling chain to display a caller's local variables (which hides the local variables of the currently executing routine). Add the caller's local variables to the Expressions window to view them without changing levels in the Stack Crawl pane.

For more information, see [“Expressions Window” on page 420](#).

Viewing raw memory

To examine and change the raw contents of memory:

1. **Select an item or expression representing the base address of the memory you want to examine.**
2. **Choose View Memory from the Data menu.**

A new Memory window opens, displaying the contents of memory in hexadecimal and ASCII format. You can change memory directly from the Memory window by entering hexadecimal values or

characters. You can also change the beginning address of the memory being displayed by changing the expression in the **Display** text field at the top of the window.

Viewing memory at an address

The **View Memory** and **View Memory As** commands in the Data menu allow you to follow any pointer—including an address stored in a register—and view the memory to which it currently points. To display the memory referenced by a pointer:

1. **Select the *value* of the variable or register in a window in which it is displayed.**
2. **Choose **View Memory** or **View Memory As** from the Data menu.**

If you choose **View Memory**, a Memory window opens displaying a raw memory dump starting at the address referenced by the pointer. If you choose **View Memory As**, a dialog box opens and asks you to select a data type ([Figure 11.66](#)); continue with step 3.

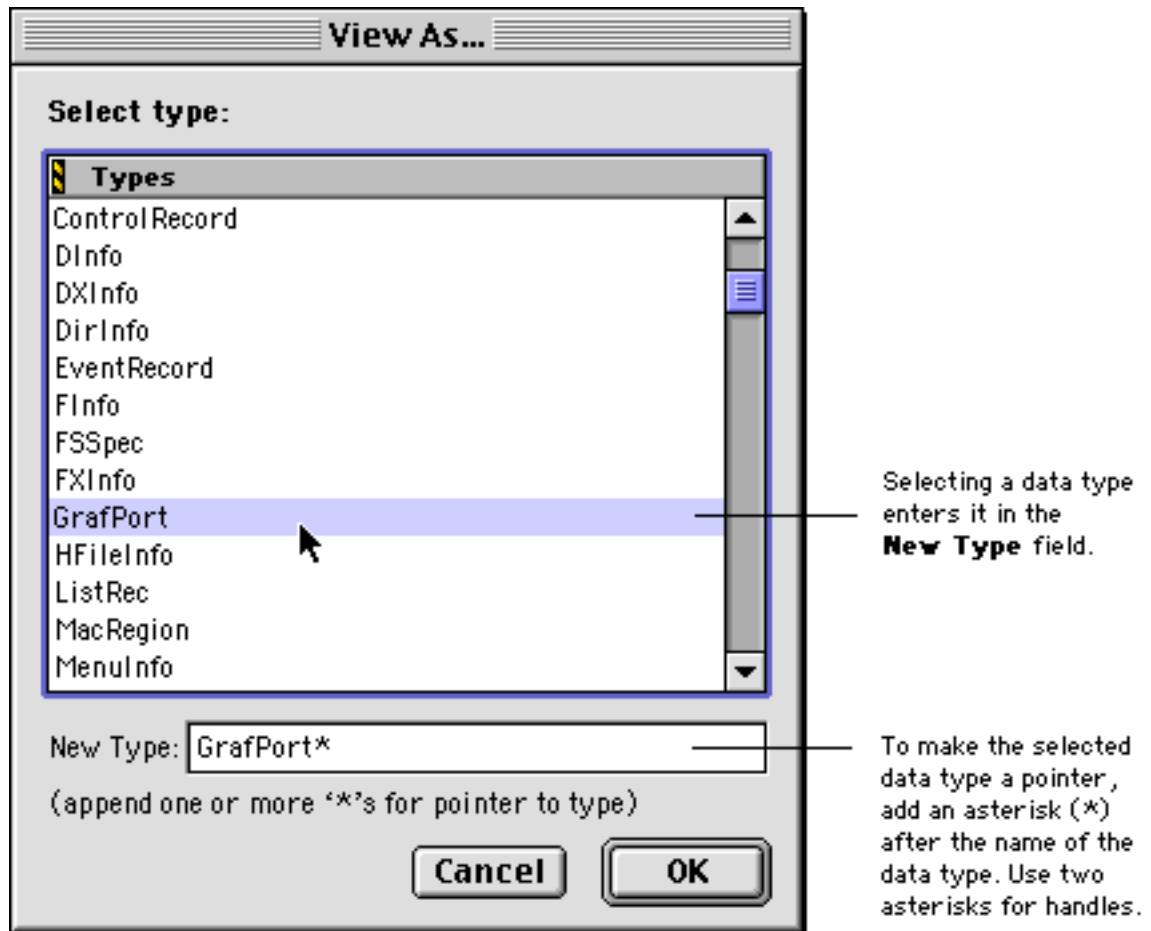
3. **If you chose **View Memory As**, select a data type in the **View As** dialog box.**

The data type you select is displayed in the **New Type** text field near the bottom of the dialog box. To view the memory pointed to by a register, append an asterisk (*) to the type name.

4. **Click OK.**

A new window opens ([Figure 11.67](#)) and displays the contents of memory, starting at the address referenced by the pointer.

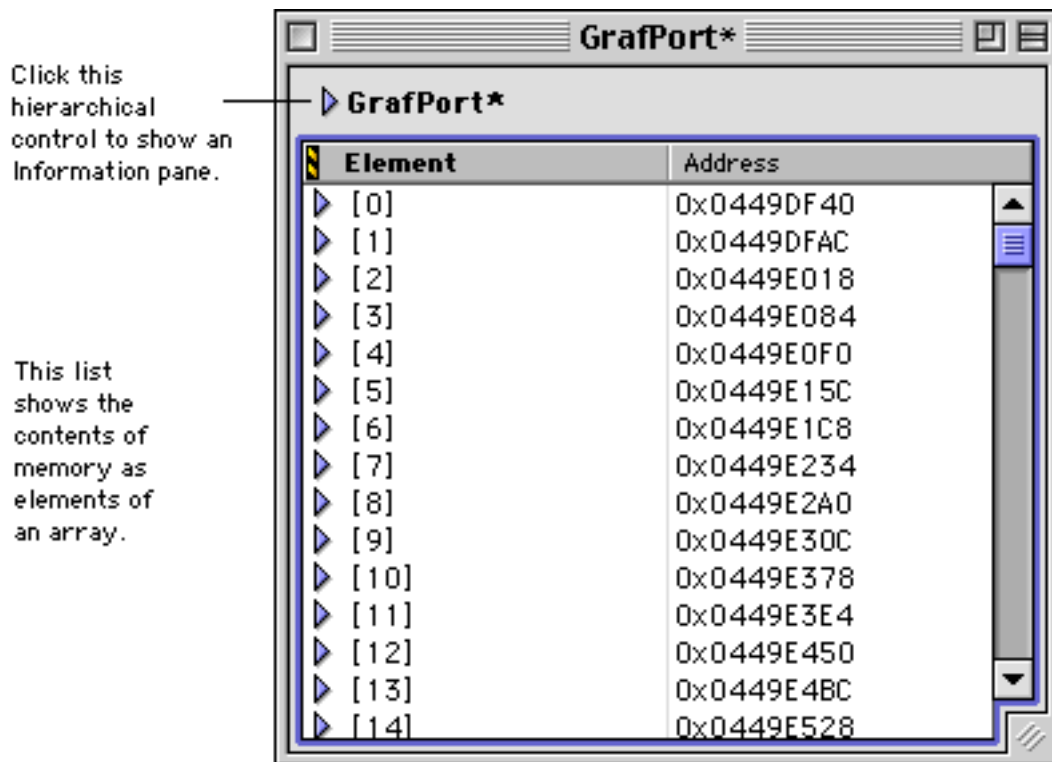
Figure 11.66 Choosing a data type to view memory



NOTE You can use this technique to view the contents of the stack. If your target processor stores the stack pointer in a particular register, select the value of that register. Then follow the previously described steps.

For more information, refer to [“Memory Window” on page 437](#).

Figure 11.67 Viewing memory as a specified data type

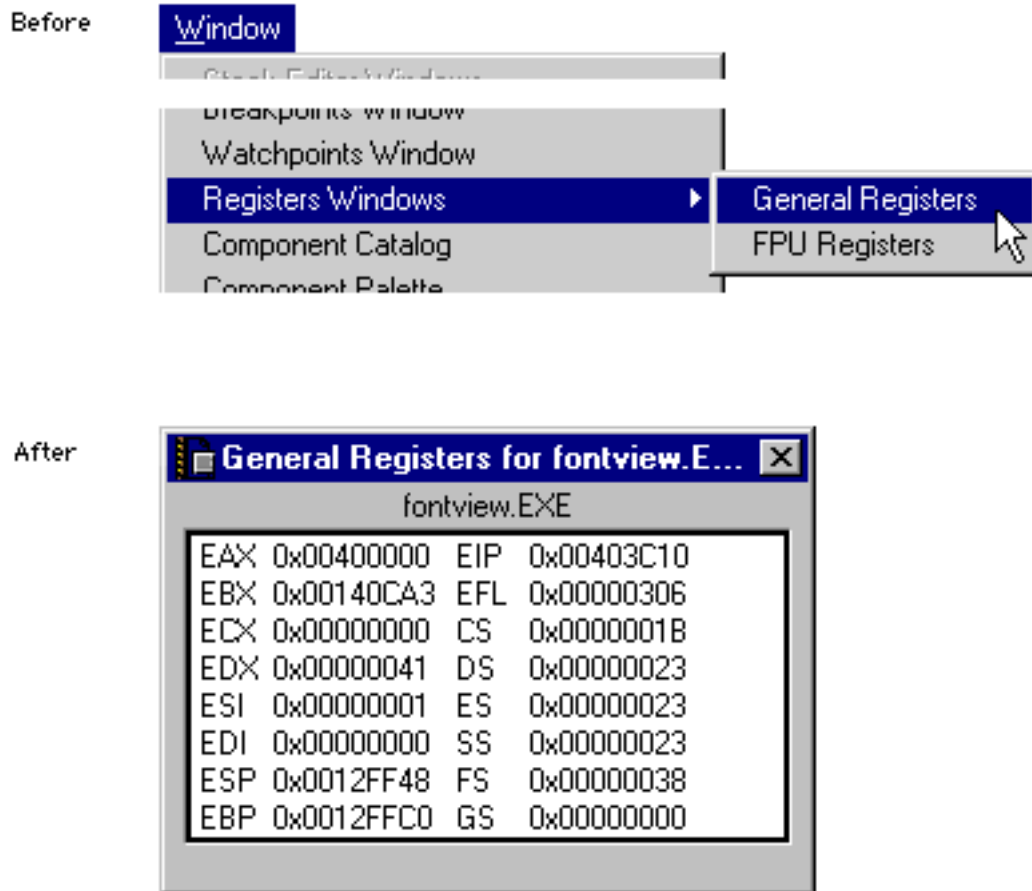


Viewing Processor Registers

To view the contents of the processor's general registers, choose **Window > Registers Windows > General Registers** from the Window menu ([Figure 11.68](#)).

For more information, see [“Register Windows” on page 430](#).

Figure 11.68 Viewing general processor registers



Editing Source Code

You cannot edit source code directly in the debugger. However, you can use the debugger to open the source-code file so that you can modify your code. In the Files pane of the Source Code Browser window, you can double-click a file name to open the file in an editor window.

Windows You can specify that a third-party editor open the file. See [“IDE Extras” on page 259](#) for more information.

Expressions

An expression represents a computation that produces a value, and the debugger displays the value in the Expressions window. You can attach an expression's value to breakpoints and watchpoints. The debugger evaluates all expressions each time it executes a statement.

An expression can combine literal values (numbers and character strings), variables, registers, pointers, and C++ object members with operations such as addition, subtraction, logical and, and equality.

An expression can appear in the Expressions window, the Breakpoints window, or a Memory window. The debugger treats the result of the expression differently, depending on the window in which the expression is displayed.

This section discusses how expressions are treated and used in the debugger. The topics in this section are:

- [How Expressions are Interpreted](#)
- [Using Expressions](#)
- [Example Expressions](#)
- [Expression Syntax](#)

How Expressions are Interpreted

The debugger interprets expressions in various ways, depending on their locations:

- [Expressions in the Expressions Window](#)
- [Expressions in the Breakpoints Window](#)
- [Expressions in a Memory Window](#)

Expressions in the Expressions Window

The Expressions window displays expressions and their values. To see the value of an expression, place it in the Expressions window. To create a new expression:

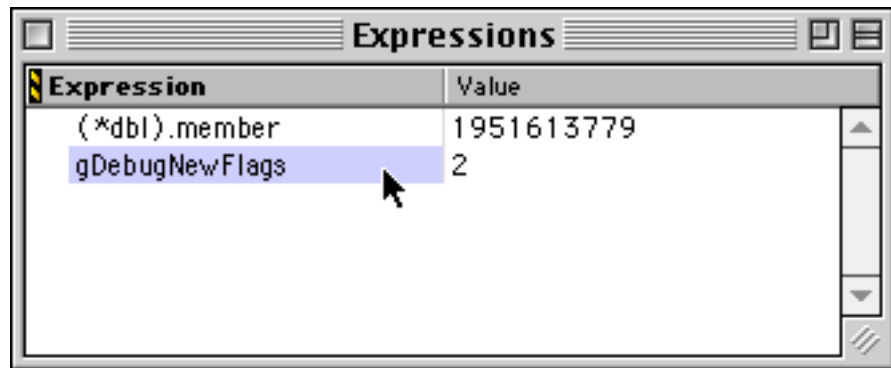
1. **Choose Expressions Window from the Window menu.**

If the Expressions window is already open, click it to make it active.

2. Choose New Expression from the Data menu.
3. Type a new expression and press Enter/Return.

The expression's value is displayed in the Value column next to that expression ([Figure 11.69](#)). You can also create a new expression by dragging a variable or expression from another window to the Expressions window.

Figure 11.69 An expression in the Expressions window



The Expressions window treats all expressions arithmetically: the debugger does not interpret the expression's result as a logical value, as it does in the Breakpoints window.

For more information, see ["Expressions Window" on page 420](#).

Expressions in the Breakpoints Window

You can attach an expression to a breakpoint in the Breakpoints window. The Breakpoints window treats expressions logically rather than arithmetically.

If the expression yields a false (zero) result, the debugger ignores the breakpoint and continues execution. If the result is true (nonzero), the debugger stops at the breakpoint if the breakpoint is active. To learn how to set a breakpoint, see ["Setting and clearing breakpoints" on page 458](#).

You can attach an expression to a breakpoint to make the breakpoint conditional on that expression:

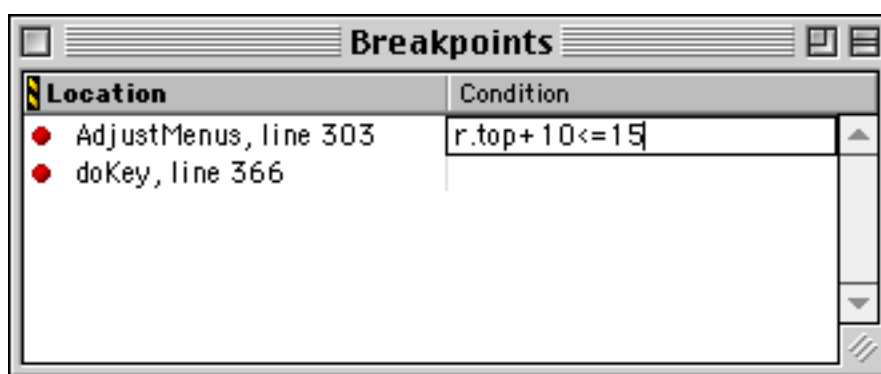
1. **Choose Breakpoints Window from the Window menu.**

If the Breakpoints window is already open, click it to make it active.

2. **Set a condition.**

Double-click the condition field for the desired breakpoint and type an expression ([Figure 11.70](#)). You can also add or change a breakpoint's condition by dragging an expression from another window and dropping it on the breakpoint's condition.

Figure 11.70 A conditional expression in the Breakpoints window



A conditional breakpoint stops the program if the expression yields a true (nonzero) result when execution reaches the breakpoint. If the expression produces a false (zero) result, execution continues without stopping.

For more information, refer to [“Breakpoints Window” on page 423](#) and [“Conditional breakpoints” on page 461](#).

Expressions in a Memory Window

In a Memory window, expressions are treated as addresses. The expression in the **Display** text field at the top of the window defines the base address for the memory displayed in the window. To change a Memory window's base address, follow these steps:

1. **Choose View Memory from the Data menu.**

If a Memory window is already open, click it to make it active.

2. **Enter a new expression.**

Double-click the **Display** field and type an expression. You can also drag an expression from another window and drop it in the Memory window's **Display** field.

The Memory window displays the contents of memory beginning at the address obtained by evaluating the new expression.

Refer to [“Memory Window” on page 437](#) for more information.

Using Expressions

The debugger's expression syntax is similar to that of C/C++, with a few additions and limitations. Pascal style expressions are also supported. The topics discussed include:

- [Special Expression Features](#)
- [Expression Limitations](#)

Special Expression Features

Expressions can refer to specific items:

- The debugger considers integer values to be 4 bytes long. Use the `short` data type to denote a 2-byte integer value.
- The debugger treats `double` values as objects 10 bytes (80 bits) in length, rather than 8 bytes (64 bits).
- To compare character strings, use the `==` (equal) and `!=` (not equal) operators. Note that the debugger distinguishes between Pascal- and C-format strings. Use the prefix `\p` when comparing Pascal string literals. The expression

```
"Nov shmoz ka pop" == "\pNov shmoz ka pop"
```

 yields a false result, because it compares a C string and a Pascal string.
- (Mac OS) To refer to register values, use the `@` symbol and a register name. (Type Option-r to get the `@` symbol.)

Expression Limitations

Expressions have the following limitations:

- Do not use C/C++ preprocessor definitions and macros (defined with the `#define` directive). They are not available to the

expression evaluator, even though they are defined in the source code.

- Do not use operations involving side effects. The increment (`i++`) and decrement (`i--`) operators, as well as assignments (`i = j`), are not allowed.
- Do not call functions.
- Do not use function names or pointers to functions.
- Do not use expression lists.
- Do not use pointers to C++ class members.
- The debugger cannot distinguish between identical variable names used in nested blocks to represent different variables (see [Listing 11.4](#)).

Listing 11.4 Identical variable names in nested blocks (C++)

```
// The debugger cannot distinguish between x the
// int variable and x the double variable. If x is
// used in an expression, the debugger will not
// know which one to use.
```

```
void f(void)
{
    int x = 0;
    .
    .
    .
    {
        double x = 1.0;
        .
        .
        .
    }
}
```

- Type definitions that are not available to the debugger cannot be used in expressions (see [Listing 11.5](#)).

Listing 11.5 Type definitions in expressions (C/C++)

```
// Use long in expressions; Int32 not available
typedef long Int32;

// Use Rect* in expressions; RectPtr not available
typedef Rect* RectPtr;
```

- Nested type information is not available. In [Listing 11.6](#), use `Inner`, not `Outer::Inner` in a debugger expression.

Listing 11.6 Nested type information (C/C++)

```
// To refer to the i member, use Inner.i,
// not Outer::Inner.i

struct Outer
{
    struct Inner
    {
        int i;
    };
};
```

Example Expressions

The list below provides example expressions that you can use in any window that uses expressions.

- A literal decimal value:
`160`
- A literal hexadecimal value:
`0xA0`
- The value of a variable:
`myVariable`
- The value of a variable shifted 4 bits to the left:
`myVariable << 4`
- The difference of two variables:
`myRect.bottom - myRect.top`
- The maximum of two variables:

`(foo > bar) ? foo : bar`

- The value of the item pointed to by a pointer variable:
`*MyTablePtr`
- The size of a data structure (determined at compile time):
`sizeof(myRect)`
- The value of a member variable in a structure pointed to by a variable:
`myRectPtr->bottom`
or
`(*myRectPtr).bottom`
- The value of a class member in an object:
`myDrawing::theRect`

Below are examples of logical expressions: the result is considered true if non-zero, false if zero.

- Is the value of a variable false?
`!isDone`
or
`isDone == 0`
- Is the value of a variable true?
`isReady`
or
`isReady != 0`
- Is one variable greater than or equal to another?
`foo >= bar`
- Is one variable less than both of two others?
`(foo < bar) && (foo < car)`
- Is the fourth bit in a character value set to 1?
`((char)foo >> 3) & 0x01`
- Is a C string variable equal to a literal string?
`cstr == "Nov shmoz ka pop"`
- Is a Pascal string variable equal to a literal string?
`pstr == "\pScram gravy ain't wavy"`
- Always true:
`1`
- Always false:
`0`

Expression Syntax

This section defines the debugger's expression syntax. Each listing follows several conventions to make it easier to read:

- The first line in a definition identifies the item being defined.
- Each indented line represents a definition for that item.
- An item with more than one definition has each definition listed on a new line.
- Items enclosed in angle brackets (<>) are defined elsewhere.
- Items in *italic* typeface are to be replaced by a value or symbol.
- All other items are literals to be used exactly as they appear.

For example,

```
<name>  
    identifier  
    <qualified-name>
```

defines the syntax of a name. A name can be either an identifier or a qualified name; the latter is a syntactic category described in another of the definitions listed in this section.

The following list describes expression syntax:

```
<name>  
    identifier  
    <qualified-name>  
  
<typedef-name>  
    identifier  
  
<class-name>  
    identifier  
  
<qualified-name>  
    <qualified-class-name>::<name>  
  
<qualified-class-name>  
    <class-name>  
    <class-name>::<qualified-class-name>  
  
<complete-class-name>  
    <qualified-class-name>  
    :: <qualified-class-name>
```

```
<qualified-type-name>
    <typedef-name>
    <class-name>::<qualified-type-name>

<simple-type-name>
    <complete-class-name>
    <qualified-type-name>
    char
    short
    int
    long
    signed
    unsigned
    float
    double
    void

<ptr-operator>
    *
    &

<type-specifier>
    <simple-type-name>

<type-specifier-list>
    <type-specifier> <type-specifier-list>(opt)

<abstract-declarator>
    <ptr-operator> <abstract-declarator>(opt)
    (<abstract-declarator>)

<type-name>
    <type-specifier-list> <abstract-
declarator>(opt)

<literal>
    integer-constant
    character-constant
    floating-constant
    string-literal

<register-name>
    @PC
    @SP
    @Dnumber
    @Anumber
```

```
<register-name>
  @Rnumber
  @FPRnumber
  @RTOC

<register-name>
  $PC
  $SP
  $RTOC
  $Anumber
```

NOTE Registers not targeted by the processor do not display random values for unknown register expressions.

NOTE For specifying a register, the range for *number* depends on the number of registers available on the target processor.

```
<primary-expression>
  <literal>
  this
  ::identifier
  ::<qualified-name>
  (<expression>)
  <name>
  <register-name>

<postfix-expression>
  <primary-expression>
  <postfix-expression>[<expression>]
  <postfix-expression>.<name>
  <postfix-expression>-><name>

<unary-operator>
  *
  &
  +
  -
  !
  ~

<unary-expression>
  <postfix-expression>
  <unary-operator> <cast-expression>
```

```
    sizeof <unary-expression>
    sizeof(<type-name>)
<cast-expression>
    <unary-expression>
    (<type-name>)<cast-expression>
<multiplicative-expression>
    <cast-expression>
    <multiplicative-expression> * <cast-
expression>
    <multiplicative-expression> / <cast-
expression>
    <multiplicative-expression> % <cast-
expression>
<additive-expression>
    <multiplicative-expression>
    <additive-expression> + <multiplicative-
expression>
    <additive-expression> - <multiplicative-
expression>
<shift-expression>
    <additive-expression>
    <shift-expression> << <additive-expression>
    <shift-expression> >> <additive-expression>
<relational-expression>
    <shift-expression>
    <relational-expression> < <shift-
expression>
    <relational-expression> > <shift-
expression>
    <relational-expression> <= <shift-
expression>
    <relational-expression> >= <shift-
expression>
<equality-expression>
    <relational-expression>
    <equality-expression> == <relational-
expression>
    <equality-expression> != <relational-
expression>
```

```
<and-expression>
    <equality-expression>
    <and-expression> & <equality-expression>

<exclusive-or-expression>
    <and-expression>
    <exclusive-or-expression> ^ <and-
expression>

<inclusive-or-expression>
    <exclusive-or-expression>
    <inclusive-or-expression> | <exclusive-or-
expression>

<logical-and-expression>
    <inclusive-or-expression>
    <logical-and-expression> && <inclusive-or-
expression>

<logical-or-expression>
    <logical-and-expression>
    <logical-or-expression> || <logical-and-
expression>

<conditional-expression>
    <logical-or-expression>
    <logical-or-expression> ? <expression> :
<conditional-expression>

<expression>
    <conditional-expression>
```

Troubleshooting

This section discusses various problems people have encountered while debugging their programs. There are suggested solutions for each problem.

The general topics include:

- [General Problems](#)
- [Problems Launching the Debugger](#)
- [Problems Running/Crashing the Debugger](#)
- [Problems with Breakpoints](#)

- [Problems with Variables](#)
- [Problems with Source Files](#)

General Problems

There may come a time when one of the solutions in this section does not seem to work, or your specific problem is not discussed. The following list details additional steps you can take to try resolving the problem:

- Remove easily regenerated files and data, including `.(x)sym` files, `.dbg` files, preferences, binaries in your project (by choosing **Remove Object Code** from the Project menu. See [“Removing Object Code” on page 369](#) for more information).
- Copy the newest version of the IDE to your hard drive.
- (Mac OS) Check for extension conflicts.
- (Mac OS) Try a few sample sessions with all possible extensions disabled.

Problems Launching the Debugger

This section lists questions and problems with launching the debugger.

The debugger does not launch

Problem

Even if **Enable Debugger** is selected from the Project menu, when I debug my application the debugger does not launch.

The **Run** or **Debug** command in the CodeWarrior **Project** menu is dimmed.

Background

You can launch the debugger automatically from the CodeWarrior IDE only if the project is an application project, the debugger is enabled, and the project is properly configured to generate debugging information.

Solutions

- Make sure your project generates an application. The **Run** command is only available when creating an application.
- Make sure that the debugger is enabled. Choose **Enable Debugger** command from the Project menu to enable the debugger. If you see the **Disable Debugger** command instead, the debugger is already enabled.
- Set up your project to generate symbolics information for the debugger. For more information, see [“Preparing for Debugging” on page 397](#).

The Debug command does nothing

Problem

The **Debug** command does nothing.

Background

You must have everything configured properly for the debugger to work correctly. In addition, make sure your code actually does something!

Solutions

- Make sure debugging is enabled for the project. See [“Preparing for Debugging” on page 397](#) for more information.
- Consult the IDE release notes for the latest information on incompatibilities with third-party software.

Errors reported on launch (Mac OS)

Problem

I get error -27 when I start debugging and -619 when I quit.

Background

You may be using an older version of Ram Doubler that is not compatible with the debugger.

Solution

- Upgrade Ram Doubler to version 1.5.2 or later.

Problems Running/Crashing the Debugger

This section covers problems that occur while you are running the debugger.

Project works in the debugger, crashes without

Problem

My project works fine when running under the debugger's control. When I run the project without the debugger, my program crashes.

Background

Running a program under the debugger changes the operating environment in which the program runs. This can have the strange effect of making an otherwise flawed program work correctly. It is difficult to pinpoint the cause of this problem in your code, but there are two likely factors that can cause the odd behavior.

- The debugger slows down the program's execution. If you have code that is time-sensitive, things may happen too quickly when the debugger is not present. Keep this in mind when studying the problem.
- The presence of the debugger can also modify memory management of the project. A block of memory cannot move if the debugger is running, for example. With the debugger absent, the block moves and a memory-related bug results.

Solutions

- Look for time-sensitive problems, race conditions, and similar factors in your code.
- Look for memory-related problems, such as accessing null pointers or handles, improperly disposing of resource handles, or disposing of handles more than once.

Problems with Breakpoints

This section covers problems related to setting and clearing breakpoints.

Statements don't have breakpoints

Problem

Some statements don't have dashes in the Breakpoint column, making it impossible to set them.

Background

The CodeWarrior linkers do not generate symbolics information for source code that is not linked into the final output file. If a statement is never actually used, the linker does not include it in the final object code. You cannot set a breakpoint on such a statement, because the object code does not exist.

Code optimization may also reorganize the object code extensively, affecting the correspondence between object code and source code and making it difficult or impossible to set breakpoints accurately.

Solutions

- Make sure all your code is used. Change the source code if necessary.
- Check your source code to see if statements were ignored by the compiler because of compiler directives.
- Disable all compiler optimizations and rebuild your project. Optimization may make changes in object code that do not correspond to your source code. You should get a breakpoint marker at every available statement after disabling optimizations. For more information, see [“Impact of optimizing code on breakpoints” on page 463](#).
- Use a coding style wherein you place only one statement on a source line. The compiler outputs breakpoint information for multiple statements on a line, but the debugger only displays the current source line. You might step multiple times on the same source line.

Breakpoints do not respond

Problem

I set a breakpoint, but it does not work.

Background

A breakpoint stops execution only if it is reached, it is active, and its condition (if it has one) is true.

Solutions

- Step through your code to verify whether you reach the statement at which you placed the breakpoint. To learn more, refer to [“Setting and clearing breakpoints” on page 458](#).
- Look in the Breakpoints window to see if the breakpoint is inactive.
- If the breakpoint has a condition, make sure it evaluates to true. The debugger ignores breakpoints with false conditions. See [“Conditional breakpoints” on page 461](#) for more information.

Problems with Variables

This section covers problems related to variables.

A variable does not change

Problem

I have a variable and I assign it a value, but the value does not change in the debugger.

Background

You are not using the variable in the code. As a result, the compiler optimizes the final output file so that the variable is not used.

Solutions

- Remove the unused variable from your code.
- Modify your code to use the variable.

Variables are assigned incorrect values

Problem

I notice that values seem to be changing incorrectly. I encounter one of these two problems:

- Two or more variables are set to the same value simultaneously.

- One variable receives a value that is supposed to be assigned to another.

Background

The compiler has recognized that the variables are not used concurrently, and has given the variables the same storage location. What you are seeing is a kind of automatic compiler optimization called “register coloring.” Register coloring checks to see how variables are used in a routine. If two or more variables are in the same scope but are not used at the same time, the compiler may use the same processor register for both variables. Using registers instead of memory to store and manipulate variables improves a program’s performance.

[Listing 11.7](#) is a good example of the kind of code that results in register coloring. Because four different variables are set but never used simultaneously, the compiler has arranged for all four to use the same register. The debugger, however, has no way of knowing that all four variables share the same register, so it shows all four variables changing with each assignment.

Listing 11.7 Variables changing with register coloring

```
void main(void)
{
    long a = 0, b = 0, c = 0, d = 0;

    a = 1; /* a is set to 1 */
    a = 2; /* a is set to 2, b remains unchanged */
    a = 3; /* a is set to 3, c remains unchanged */
    a = 4; /* a is set to 4, d remains unchanged */
}
```

Solutions

- Do nothing. Register coloring is not a problem.
- To prevent register coloring in C/C++, declare your variables with the `volatile` keyword. Do this with a preprocessor directive so that you can easily remove the `volatile` storage class specifiers after debugging. See the *C Compilers Reference* and the *Assembler Guide* for more information.

Strange variable names

Problem

The debugger shows variables in the Variables pane or Global Variables window that are not declared in the source code.

Background

The compiler often creates its own temporary variables in the object code as it translates source code. These temporary variables have a dollar sign (\$) in their names. The debugger also displays C++ virtual base class types with a \$ prefix.

The compiler and linker often add variables from libraries and run-time routines that help initialize and terminate your program.

Solution

- None. This is not a problem that needs correction.

Strange data types

Problem

When I select **Show Types** from the Data menu, some enumerated values are displayed as having type “?anonx,” where x is an arbitrary number.

Background

The debugger cannot display the names of enumerated types if the names are not defined in the source code. At compile time, the compiler assigns a generic type name to such enumerated types. It is this generic name that the debugger displays.

For example, in [Listing 11.8](#), with **Show Types** selected, variable `myMarx` is displayed as having the anonymous type `?anonx`, because its enumerated type has no name. On the other hand, variable `myBeatle` is shown with type `Beatle`, because its enumerated type is defined with that name.

Listing 11.8 Unnamed enumerated types (C/C++)

```
// Debugger displays as anonymous type
enum {Groucho,
```

```
Harpo,  
Chico,  
Zeppo } myMarx = Harpo;  
  
// Debugger displays as type Beatle  
typedef enum Beatle {John,  
                    Paul,  
                    George,  
                    Ringo} myBeatle = John;
```

Solution

- None. This is not a problem that needs correction.

Unrecognized data types

Problem

I declared my own data type. Why can I not view a variable as that type?

Background

The symbolics file includes information only about types that are used in the program. Types defined in `typedef` statements are not stored in the symbolics file, so you need to view the variable as the type from which it is derived. For example, if you declare a type `MyLong` based on the `long` data type, you can view it as a `long`, but not as a `MyLong`. For more information, refer to [“Viewing data as different types” on page 474](#).

Solution

- Use the base data type.
- Choose the **Show Types** item from the Data menu to see how the debugger labels the type.

“undefined identifier” in the Expressions window

Problem

A user-defined type in an expression in the Expressions window gives an “• undefined identifier •” value.

Background

The debugger does not recognize data types that are simply aliases of another type, because such alias types are not included in the symbolics file.

For example, given the Pascal type declaration

```
TYPE  
    MYBIGINT = LONGINT;
```

the expression

```
MYBIGINT(thePtr)
```

in the Expressions window will display its value as
“• undefined identifier •.” To get the correct result, use the following expression instead:

```
LONGINT(thePtr)
```

To learn more, see [“Expression Limitations” on page 485](#).

Solution

- Use the original data type instead of the defined data type.

Problems with Source Files

This section covers problems related to source-code files.

No source-code view

Problem

All I see in the source pane is assembly-language code. The Source pop-up menu does not let me display source code.

Background

There is no symbolics information available for that code. You may not have enabled debugging for a file, or you may be stepping through some ancillary code added by the linker that has no corresponding source code (for example, glue code). Without symbolics information, the debugger can only display the code in assembly language.

Solutions

- If the code is from your own source file, make sure to generate symbolics information for the file. For more information, refer to [“Setting Up a File for Debugging” on page 398](#).
- If the code is from some other source (such as a compiled library), step out of the function to return to the caller. There is no source code to view.

Outdated source files

Problem

When I run my project, I get a warning that says the modification dates do not match.

Background

The symbolics file keeps track of when the source file on which it is based was last changed. If the date and time stored in the symbolics file do not match those of the original file, the debugger warns you that the symbolics information may no longer be current.

Solution

- Touch the source file (or make a do-nothing change and save the file), then rebuild your project or bring it up to date.

Spurious ANSI C code in Pascal projects

Problem

I am working on a Pascal program, and when I step through code I find ANSI C routines. I have not included any ANSI C libraries.

Background

The Pascal runtime library was written in C and uses ANSI C routines. These are the routines that are displayed when debugging a Pascal program.

Solution

- None. This is not a problem that needs correction.

Debugger Error Messages

The following list explains error messages that you might receive from the debugger, with some hints about the possible causes or circumstances of the error. Messages listed without comment are self-explanatory.

An unknown error occurred while trying to target an existing process.

Bad type code

Internal error.

Bus Error

Attempt to read or write to an invalid address.

can't display value -- type information not supported

The symbolics file contains a data type that the debugger does not support.

Can't use this source file, it was not saved before running, or was edited after linking.

The debugger does not have access to the same text available to the compiler. The debugger just issues a warning unless the debugging information references nonexistent text, in which case it gives you this error message.

class name expected

Unexpectedly encountered something other than a class name while evaluating an expression.

Could not complete your request because the process is not suspended.

The command you issued cannot be performed while the program is running.

Could not set a watch point because the page containing that memory location overlaps low memory or the system heap.

Watchpoints cannot be set in low memory or in the system heap.

Could not set a watch point because the page containing that memory location overlaps the stack.

Watchpoints are implemented via the memory write-protection mechanism, which operates at the page level. You cannot write-protect a page of memory containing part of the stack.

Couldn't locate the program entry point, program will not stop on launch.

When launching a program, the debugger normally sets an implicit breakpoint at the beginning of the function named `main()` (in C/C++) or the main program (in Pascal). If the debugger cannot find such a routine, it just launches the program without suspending that program's execution.

identified or qualified name expected

Unexpectedly encountered something other than an identifier or qualified name while evaluating an expression.

illegal character constant

Invalid character constant encountered while evaluating an expression.

illegal string constant

Invalid string constant encountered while evaluating an expression.

illegal token

Invalid token encountered while evaluating an expression.

Invalid C or Pascal string.

An ill-formed string was encountered in evaluating an expression.

Invalid character constant.

An invalid character constant was encountered in evaluating an expression.

Invalid escape sequence inside string or character constant.

C/C++ escape sequence in a string was not valid syntax.

invalid pointer or reference expression

Invalid pointer or reference expression encountered while evaluating an expression.

invalid type declaration

Invalid type encountered while evaluating an expression.

invalid type information in SYM file

The debugger is unable to display a variable because of bad data in the symbolics file.

New variable value is too large for the destination variable.

For example, you have attempted to assign a 20-byte string to a 10-byte string variable.

No type with that name exists.

The debugger does not recognize a type name you have entered in the **View As** dialog box.

Register not available

The debugger is unable to get a valid register value to display a register variable. For example, when looking at routines up the stack from the current routine, the debugger cannot retrieve the saved register values unless all routines below it on the stack have debugging information.

string too long

String exceeds maximum permissible length.

The new variable value is the wrong type for the destination variable.

You have attempted to assign a value of the wrong type to a variable, such as a string to an integer variable.

typedef name expected

Unexpectedly encountered something other than a typedef name while evaluating an expression.

Unable to step from here.

The debugger cannot step execution from this point.

Unable to step out from here.

The debugger cannot step out from this point.

undefined identifier

Undefined identifier encountered while evaluating an expression.

unexpected token

Unexpected token encountered while evaluating an expression.

unknown error "^0"

An internal error that was not expected to reach the user.

unterminated comment

A closing comment bracket is missing.

Variable or expression cannot be used as an address.

For example, if `r` is a `Rect`, `*(char*)r` is invalid.

**Warning - this SYM file has some invalid or inconsistent data.
The debugger may show incorrect information.**

Your symbolics file may have been corrupted.

'*' or '&' expected

Unexpectedly encountered something other than a pointer or reference operator while evaluating an expression.

RAD Designs and Layouts



This chapter helps you understand the major features of CodeWarrior's rapid application development (RAD) tools. You can also learn about the motivations for RAD development, as well as the concepts of designs and layouts in CodeWarrior projects.

The sections in this chapter are:

- [About RAD](#)
- [CodeWarrior RAD Tools](#)
- [Creating RAD Projects](#)
- [RAD Wizards](#)
- [Working with Designs](#)
- [Working with Layouts](#)

About RAD

Traditional application programming requires tedious and manual work. When you create a new application, you are responsible for coding every aspect of the program. You must specify all initialization routines, the interactions between the various pieces of the program, and the user interface.

RAD tools automate some of these manual processes, so you can concentrate your efforts on the application itself. The tools automatically generate code that handles generic initialization routines and the drawing of user interface elements on the screen. In addition, you can modify the generated code to suit your application's specific needs.

CodeWarrior includes RAD tools as part of the development environment. These tools support the development of applications, applets, and components through a plug-in architecture. You can

use existing RAD plug-ins with the IDE, or you can create your own. This flexibility supports multiple languages, platforms, and application frameworks.

CodeWarrior RAD Tools

CodeWarrior RAD tools let you visually construct an application. The tools included with the IDE are for use with Java. They extend the graphical capabilities of the IDE for use with RAD. Additional programming frameworks will be supported in the future.

This section discusses the following topics:

- [Class Authoring](#)
- [Component Model](#)

Class Authoring

RAD tools let you create and modify classes through a high-level visual interface. You can use this interface to create, rename, and delete the various classes, member functions, and data members in your application. The RAD tools automatically synchronize declarations and definitions as you modify the application.

Because CodeWarrior uses a plug-in architecture for RAD, you can easily extend the IDE's capabilities. The RAD tools automatically generate source code via scripts. You can modify existing scripts to customize the generated code for your application.

Component Model

CodeWarrior RAD tools use a component model to create applications. A component in RAD context is a self-describing object. The component represents an object in the programming framework you use to create an application. All of the properties, methods, and events that describe an object are included in the component.

Components enable a powerful and intuitive user interface for applications across a wide variety of platforms. RAD tools use these

components to provide several services, including persistence and automation.

CodeWarrior uses these tools to implement the component model:

- [Layout Editor](#)
- [Catalogs](#)

Layout Editor

One major part of the RAD tools is the Layout editor. This application builder uses a visual development environment. User interface elements are visually represented as components. You can drag and drop components to form your application. As you make changes, the IDE updates the Layout editor display and modifies the generated source code accordingly.

Catalogs

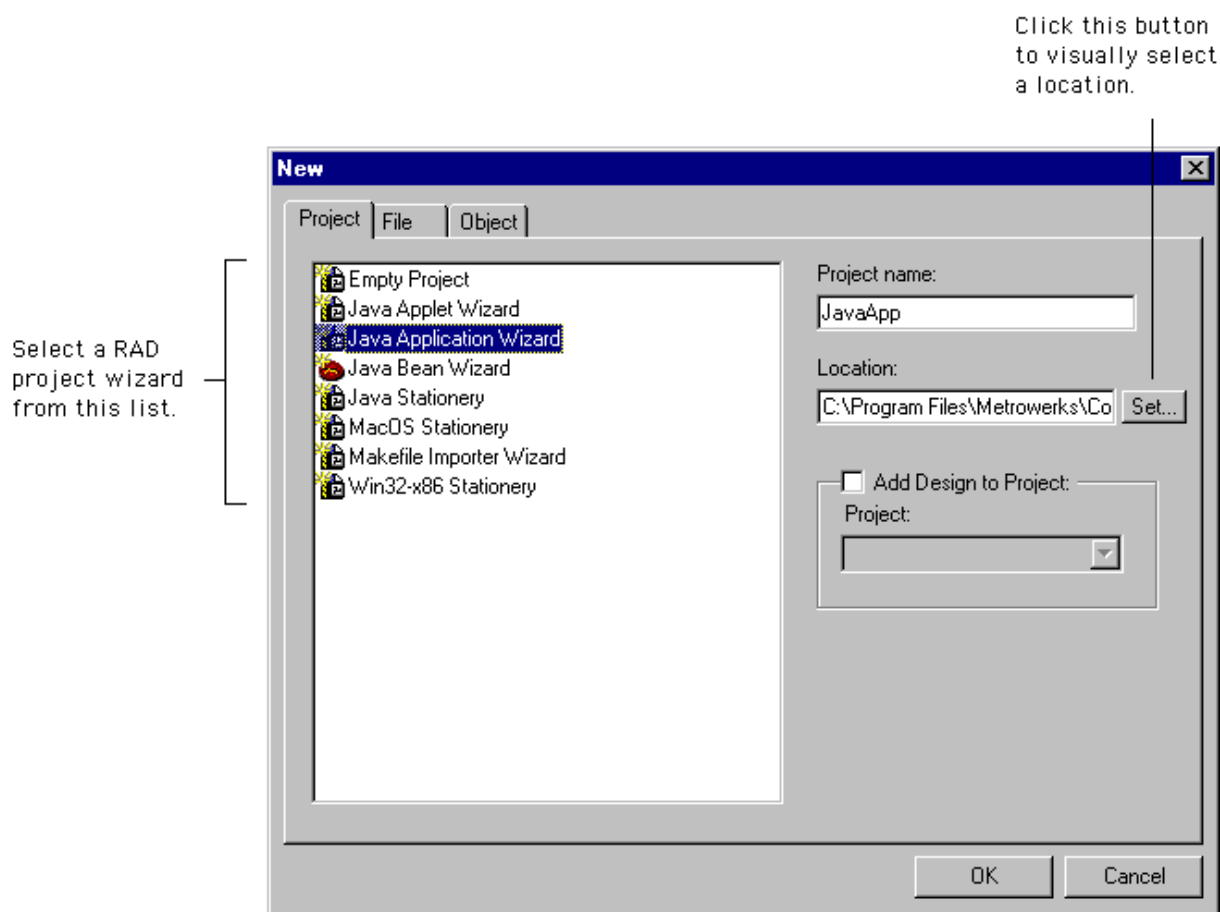
Catalogs comprise the second major part of the RAD tools. Catalogs are repositories of available components for the application. The IDE provides catalogs based on the programming framework you use. For example, when you create a Java application, you can choose components from Java-specific catalogs. The properties and events of the components in a catalog are fully customizable.

Creating RAD Projects

To create a CodeWarrior RAD project, choose **New** from the File menu. The IDE displays the New window, as shown in [Figure 12.1](#).

Click the **Project** tab at the top of the window to display the Project panel. This panel shows the available stationery and wizards that you can use to create new CodeWarrior projects. When you select an item in the list, various options become available along the right side of the Project panel.

Figure 12.1 New dialog box

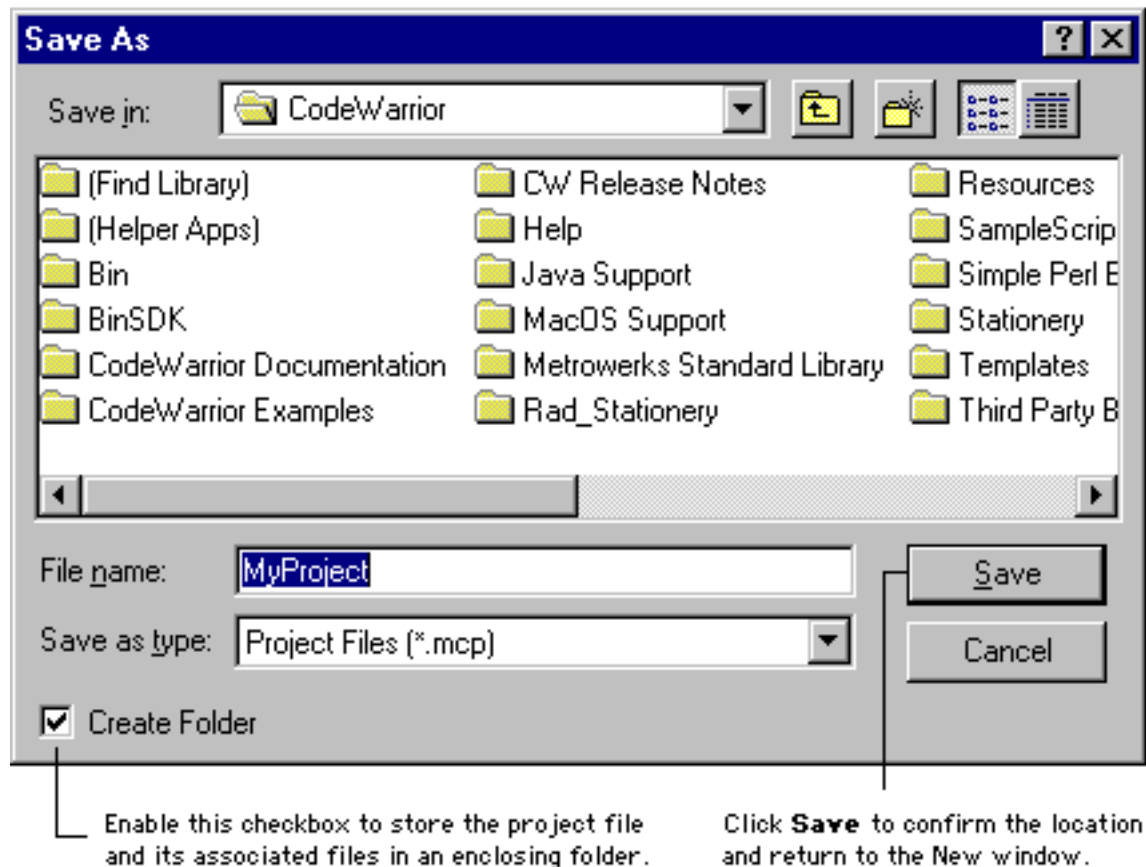


To create a new RAD project, select one of the following wizards in the project list:

- **Java Applet Wizard**—helps you create a Java applet. The wizard asks for information about the applet's class, HTML page, parameters, and description.
- **Java Application Wizard**—helps you create a Java application. The wizard asks for information about the application's class, frame class, and description.
- **Java Bean Wizard**—helps you create a Java bean. The wizard asks you for information about the bean's class, location, package, and modifiers.

After you select a project wizard, type a name for your new project in the **Project name** field.

Figure 12.2 Save dialog box (Windows)



TIP We suggest naming your project file with a `.mcp` file name extension, like this: `MyProject.mcp`. This naming convention helps you quickly identify the project file on your hard disk. In addition, the Windows-hosted version of the CodeWarrior IDE uses this extension to identify the project file.

The **Location** field displays the full path to the folder in which the project is saved. To change the current path, type a new path directly into the field. Alternatively, click the **Set** button to the right of the field to display a dialog box, which is shown in [Figure 12.2](#) (Windows) and [Figure 12.3](#) (Mac OS). Use the dialog box controls to navigate to a location on your hard disk where you want to save the project. To create a new folder to contain the project file and all of its associated files, make sure the **Create Folder** checkbox is enabled.

Then click **Save** to confirm the location and return to the New window.

WARNING! If you try to save the new project, and you already have an existing project with the same name in the same location on the hard disk, the IDE displays an error message. Be sure to use a unique name for your new project.

Figure 12.3 Save dialog box (Mac OS)



When you finish typing a name for the RAD project and choosing a location in which to save that project, click **OK** in the New window. The IDE opens the wizard you selected in the Project panel. The selected wizard guides you through the remaining steps of the project-creation process. For detailed information about each wizard, see [“RAD Wizards” on page 515](#).

RAD Wizards

When creating a new RAD project or design, as described in [“Creating RAD Projects” on page 511](#) and [“Adding Designs to a Project” on page 531](#), you select a RAD wizard. The wizard helps you complete the item-creation process. This section describes the following RAD wizards:

- [Java Applet Wizard](#)
- [Java Application Wizard](#)
- [Java Bean Wizard](#)

For beginners

Each wizard assumes you have a basic understanding of the associated programming language. For example, if you use the Java Application wizard, you should understand the difference between an Abstract Window Toolkit (AWT) frame and a Swing frame. If the items displayed in the wizard are unfamiliar, you should consult additional programming language references for more information.

The RAD wizards include the following navigation buttons:

- **Back**—return to the previous step
- **Next**—proceed to the subsequent step
- **Finish**—display a summary of current information
- **Cancel**—discard all changes

Each wizard is divided into a series of steps. You progress through these steps in sequence, and each step builds on the information provided in previous steps. When you supply enough information in a particular step, click **Next** to continue.

You can modify the default settings in each step. To accept all current settings in the wizard, click **Finish**. The wizard displays a summary of all the current settings for the new project. Click **Generate** to accept the current settings and create the new RAD item, or click **Cancel** to return to the wizard and continue modifying settings.

Java Applet Wizard

The Java Applet wizard is divided into the following steps:

1. [Describe the applet class for the Java applet.](#)
2. [Specify an HTML page for the Java applet. \(optional\)](#)
3. [Create parameters for the Java applet.](#)
4. [Provide additional information for the Java applet.](#)

To use the Java Applet wizard, follow these steps:

1. **Describe the applet class for the Java applet.**

This section of the wizard, shown in [Figure 12.4](#), lets you specify the class name, package name, and location for the new applet. Other options are available to further describe the applet.

This section includes the following parts:

- [Class Name](#)
- [Package Name](#)
- [Location](#)
- [Can run as a standalone application](#)
- [Generate standard methods](#)
- [Generate HTML Page](#)
- [AWT applet](#)
- [Swing applet](#)

Figure 12.4 Java Applet wizard - Applet Class

New Java Applet: Page 1 of 4

Applet Class

Class name:

Package name:

Location: ...

☐ Can run as a standalone application

☐ Generate standard methods

☒ Generate HTML Page

☒ AWT applet

☐ Swing applet

< Back Next > Finish Cancel

Class Name

Enter a name for the applet's class in this field. An example name is provided.

Package Name

If you wish, you can enter in this field a package for the applet class. A sample package name is provided.

Location

In this field, type a location in which to save the applet class for the new applet. Alternatively, click the button next to the field. A standard window displays. Use the window controls to select a location. Note that a valid location depends on your settings for strict filenames and strict package roots.

Can run as a standalone application

Enable this checkbox if you want the new applet to operate without needing to be called from another application. Disable this checkbox otherwise.

Generate standard methods

Enable this checkbox if you want the new applet to generate standard methods for its interface, such as `show()` and `dispose()`. Disable this checkbox otherwise.

Generate HTML Page

Enable this checkbox if you want to create an HTML page to help you test the new applet. Disable this checkbox otherwise. Note that the Java Applet wizard displays the HTML Page section only when you enable the **Generate HTML Page** checkbox. This checkbox is enabled by default.

AWT applet

Click this radio button to designate the new applet as an AWT applet. This radio button is enabled by default.

Swing applet

Click this radio button to designate the new applet as a Swing applet.

2. Specify an HTML page for the Java applet. (optional)

This section of the wizard, shown in [Figure 12.5](#), is displayed only when you enable the [Generate HTML Page](#) checkbox from the previous section. The HTML Page section lets you define a Hypertext Markup Language (HTML) page to test the new applet.

This section includes the following parts:

- [Title](#)
- [Applet Name](#)
- [Codebase/Archive](#)
- [Use Archive \(JAR\)](#)
- [Width](#)
- [Height](#)

- [HSpace](#)
- [VSpace](#)
- [Align](#)

Title

Enter in this field a title for the HTML page you wish to generate. An example title is provided.

Applet Name

Your HTML browser displays messages about the applet by using the name you type in this field. A sample applet name is shown.

Codebase/Archive

If desired, use this field to specify a codebase or archive for use with the new applet. An example is shown.

Figure 12.5 Java Applet wizard - HTML Page

The screenshot shows a dialog box titled "New Java Applet: Page 2 of 4". It contains the following fields and controls:

- Title:** A text field containing "HTML Test Page".
- Applet Name:** A text field containing "TestApplet".
- Codebase/Archive:** A text field containing "Java Classes".
- Use Archive (JAR):** An unchecked checkbox.
- Width:** A text field containing "400".
- Height:** A text field containing "300".
- Align:** A dropdown menu with "middle" selected.
- HSpace:** A text field containing "0".
- VSpace:** A text field containing "0".
- Navigation buttons:** "< Back", "Next >", "Finish", and "Cancel".

Use Archive (JAR)

Enable this checkbox if you want to use a Java archive (JAR) with the new applet. If this checkbox is enabled, a “. jar” extension is appended to the name in the [Codebase/Archive](#) field.

Width

The number you type in this field represents the number of pixels that span the width of the applet within the HTML page.

Height

The number typed in this field specifies the number of pixels that span the height of the applet within the HTML page.

HSpace

The number in this field defines the number of pixels placed on the left side and on the right side of the applet within the HTML page.

VSpace

The number in this field defines the number of pixels placed above and below the applet within the HTML page.

Align

Use this pop-up menu to determine the alignment of the applet within the HTML page:

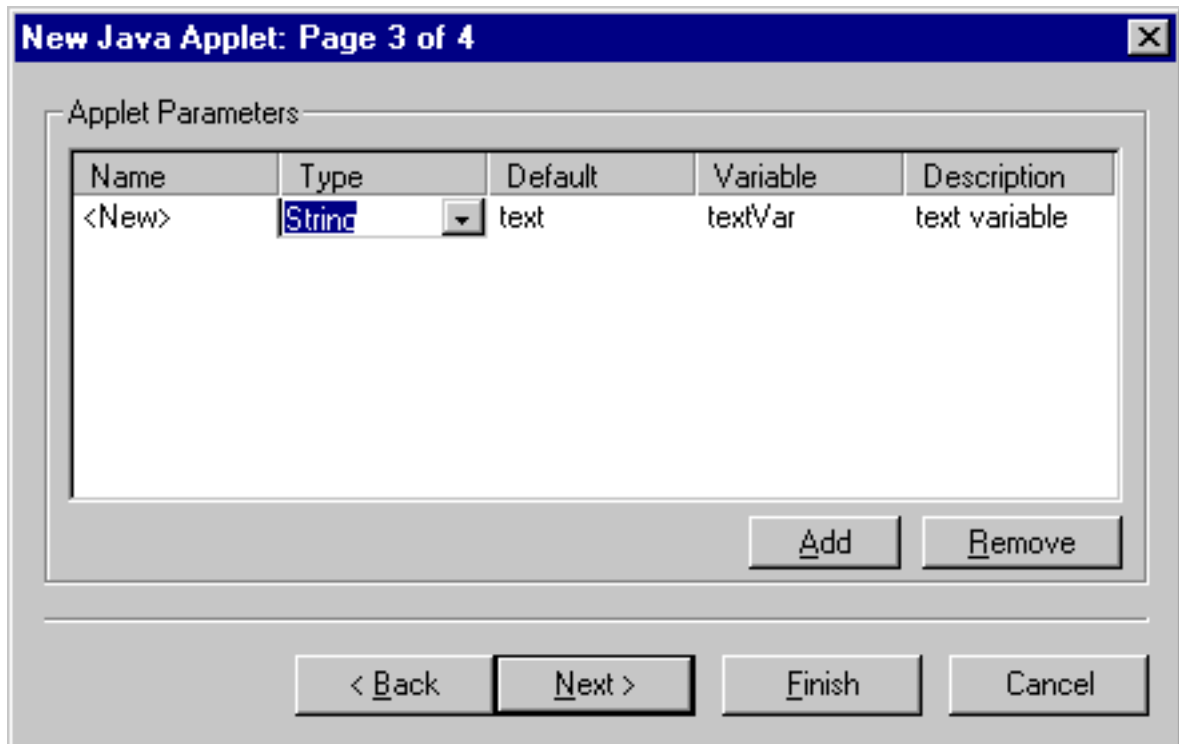
- **absbottom**—position at the absolute bottom of the page
- **absmiddle**—position at the absolute middle of the page
- **baseline**—position at the baseline of the page
- **bottom**—position at the bottom of the page
- **left**—position at the left side of the page
- **middle**—position at the middle of the page. This is the default option.
- **right**—position at the right side of the page
- **texttop**—position at the top of the text on the page
- **top**—position at the top of the page

3. Create parameters for the Java applet.

This section of the wizard, shown in [Figure 12.6](#), lets you create a set of parameters for the new applet. Each parameter has the following fields:

- **Name**—the name of the parameter
- **Type**—the data type of the parameter
- **Default**—the default value for the parameter
- **Variable**—the variable that represents the parameter
- **Description**—a short description of the parameter

Figure 12.6 Java Applet wizard - Applet Parameters



A sample **<New>** parameter is ready for you to modify. To change a particular field, double-click it. Type a new value in the field and press Enter/Return to confirm your changes.

If you double-click the **Type** field, choose a new value from the pop-up menu that displays inside the field. Alternatively, if you know which data type you would like to choose beforehand, you can use a

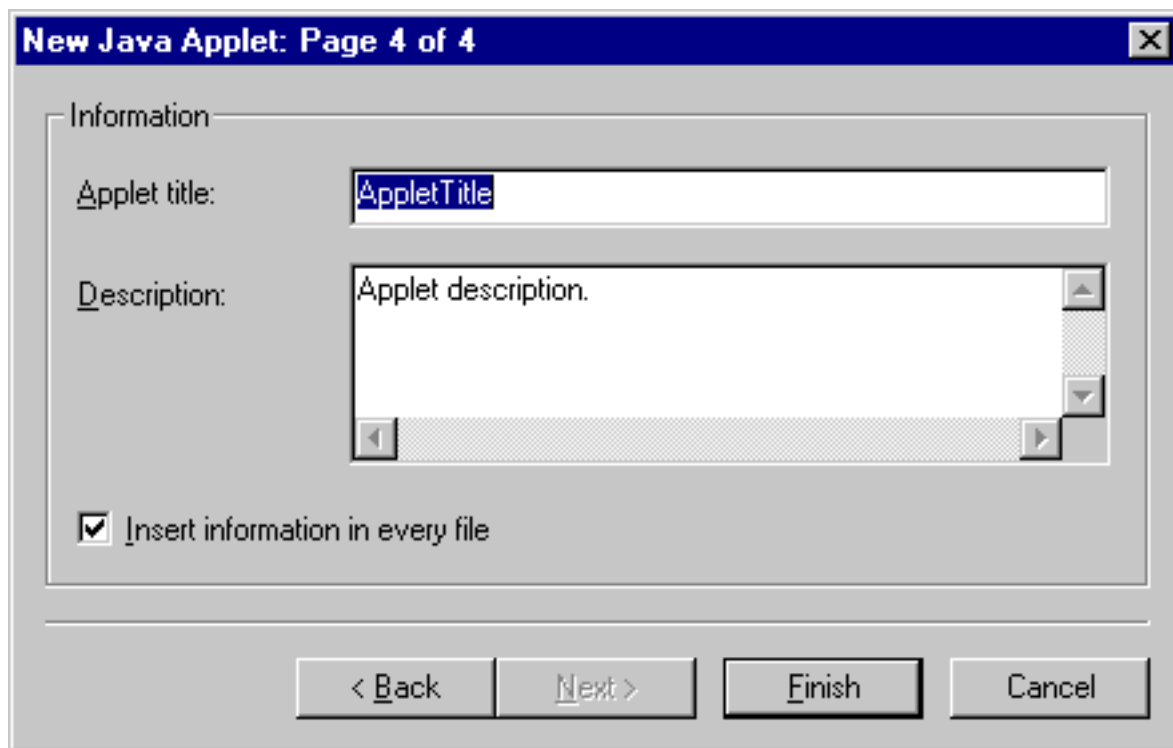
shortcut. Simply double-click the **Type** field and type the first letter of your desired data type. The field displays a new value. If an incorrect value displays, type the same letter again to choose the next closest match. For example, if you double-click the field and type the letter *s*, the field changes to **String**. Typing the letter *s* again changes the field to **short**.

To add another parameter to the applet, click **Add**. The wizard creates a new parameter for you to modify. To remove a particular parameter, select one of its fields and click **Remove**.

4. **Provide additional information for the Java applet.**

This section of the wizard, shown in [Figure 12.7](#), lets you provide additional information to describe the applet.

Figure 12.7 Java Applet wizard - Information



This section includes the following parts:

- [Applet title](#)
- [Description](#)

- [Insert information in every file](#)

Applet title

If desired, enter a title for the new applet in this field.

Description

If you wish, type a description for the new applet in this field.

Insert information in every file

To add the information contained in the [Applet title](#) and [Description](#) fields to the beginning of every applet file, enable this checkbox. Disable the checkbox otherwise. The checkbox is enabled by default.

Java Application Wizard

The Java Application wizard is divided into the following steps:

1. [Describe the application class for the Java application.](#)
2. [Describe the frame class for the Java application.](#)
3. [Provide additional information for the Java application.](#)

To use the Java Application wizard, follow these steps:

1. **Describe the application class for the Java application.**

This section of the wizard, shown in [Figure 12.8](#), lets you specify application class information for the Java application.

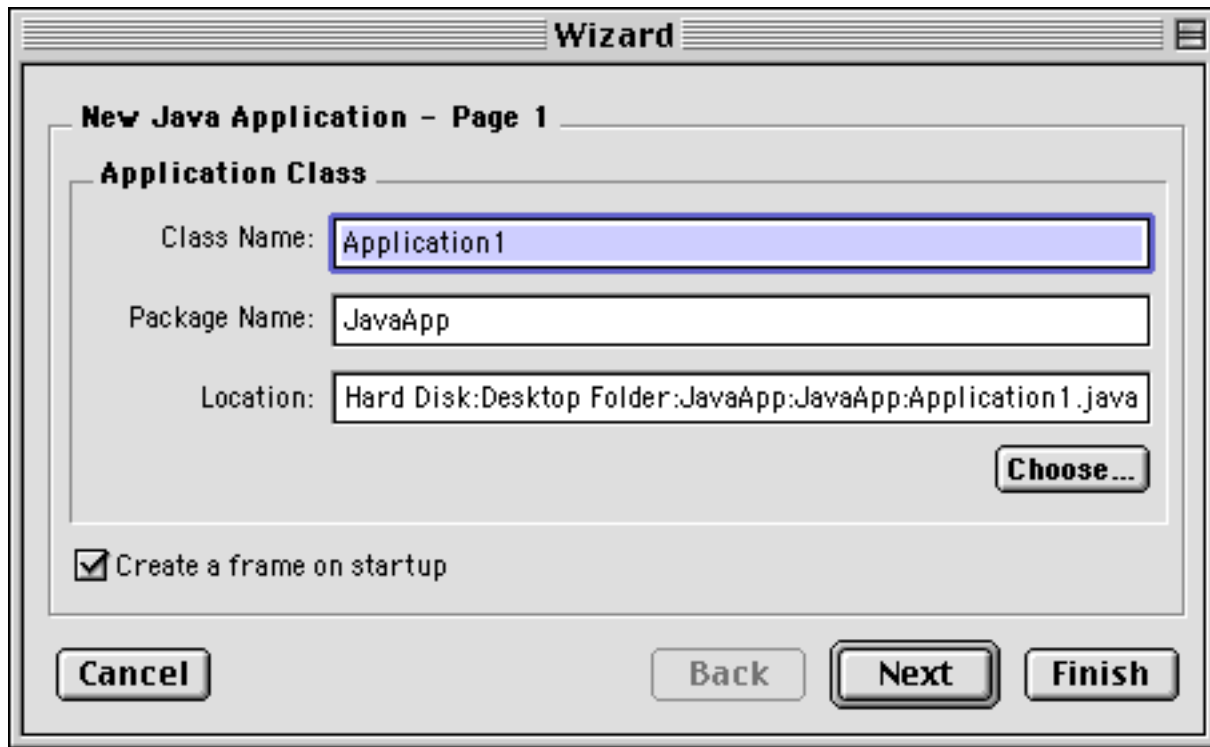
This section includes the following parts:

- [Class Name](#)
- [Package Name](#)
- [Location](#)
- [Create a frame on startup](#)

Class Name

Enter a name for the application's class in this field. An example name is provided.

Figure 12.8 Java Application wizard - Application Class



Package Name

If you wish, you can enter in this field a package for the application class. A sample package name is provided.

Location

In this field, type a location in which to save the application class for the new application. Alternatively, click the button next to the field. A standard window displays. Use the window controls to select a location.

Create a frame on startup

Enable this checkbox if you want the IDE to automatically generate a new layout frame. This option automatically displays the [Layout Editor](#), [Component Palette](#), and [Object Inspector](#). Disable the checkbox if you do not want to automatically display these tools.

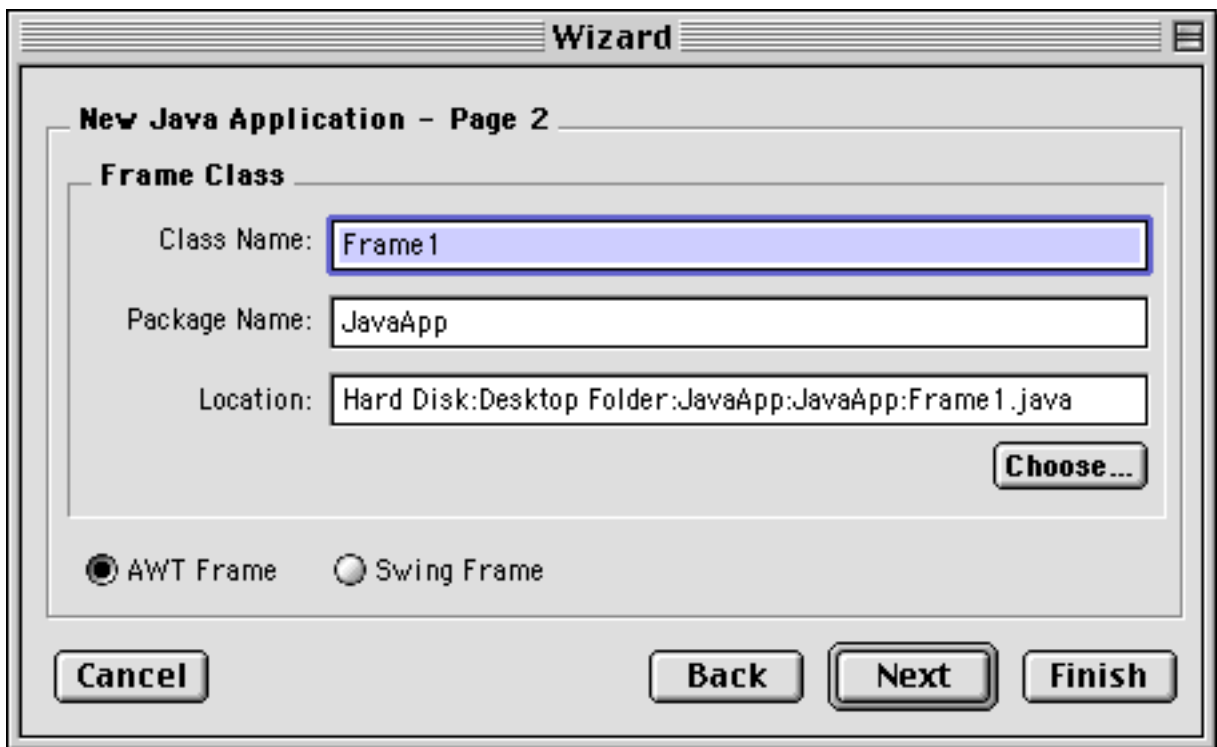
2. Describe the frame class for the Java application.

This section of the wizard, shown in [Figure 12.9](#), lets you specify frame class information for the Java application.

This section includes the following parts:

- [Class Name](#)
- [Package Name](#)
- [Location](#)
- [AWT frame](#)
- [Swing frame](#)

Figure 12.9 Java Application wizard - Frame Class



Class Name

Enter a name for the frame's class in this field. An example name is provided.

Package Name

If you wish, you can enter in this field a package for the frame class. A sample package name is provided.

Location

In this field, type a location in which to save the frame class for the new application. Alternatively, click the button next to the field. A standard window displays. Use the window controls to select a location. Note that a valid location depends on your settings for strict filenames and strict package roots.

AWT frame

Click this radio button to designate the new application as an AWT frame. This radio button enabled by default.

Swing frame

Click this radio button to designate the new application as a Swing frame.

3. Provide additional information for the Java application.

This section of the wizard, shown in [Figure 12.10](#), lets you provide additional information to describe the application.

This section includes the following parts:

- [Application title](#)
- [Description](#)
- [Insert information in every file](#)

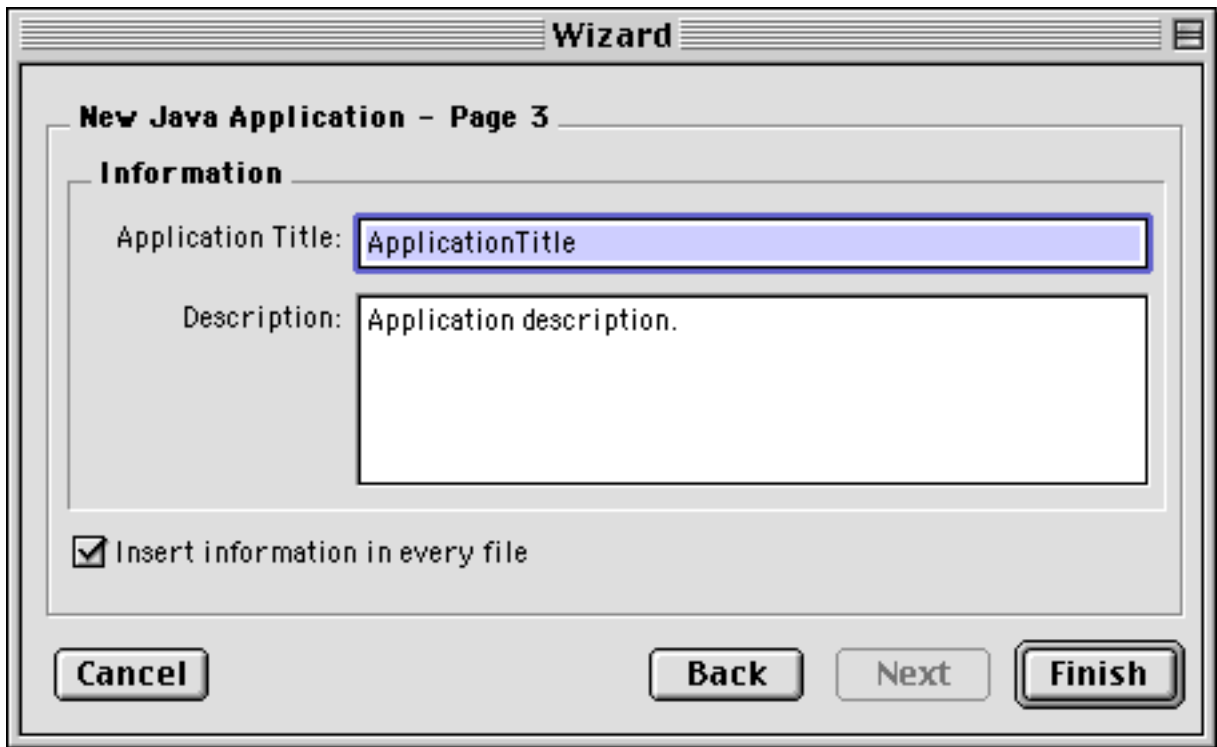
Application title

If desired, enter a title for the new application in this field.

Description

If you wish, type a description for the new application in this field.

Figure 12.10 Java Application wizard - Information



Insert information in every file

To add the information contained in the [Application title](#) and [Description](#) fields to the beginning of every application file, enable this checkbox. Disable the checkbox otherwise. The checkbox is enabled by default.

Java Bean Wizard

The Java Bean wizard is divided into the following steps:

1. [Choose a name and location for the Java bean.](#)
2. [Specify a base class and a list of implementations.](#)

To use the Java Bean wizard, follow these steps:

1. **Choose a name and location for the Java bean.**

This section of the wizard, shown in [Figure 12.11](#), lets you specify the name and location of the new bean, as well as the modifiers for the bean.

Figure 12.11 Java Bean wizard - Name and Location

New Java Bean

Name and Location

Class Name:

☒ Class is a Bean

File: ▼

▼ class

Package:

Modifiers

Access: ▼ Specifier: ▼

< Back Next > Finish Cancel

This section includes the following parts:

- [Class Name](#)
- [File](#)
- [Package](#)
- [Modifiers](#)

Class Name

Enter a name for the bean's class in this field. An example is provided below the field. The **Class is a Bean** checkbox below the field remains enabled. This status indicates that you are creating a Java bean.

NOTE If you want the applet to be compatible with other development tools, the class name should match the applet's file name.

File

This pop-up menu indicates that you are creating a **New File**. No additional options are available.

Type the path to which you want to save the file in the field above the **Package** field, as shown in [Figure 12.11 on page 528](#). Alternatively, click **Choose** next to the field. A window opens that lets you browse your computer's contents. Use the window controls to select the location to which you want to save the file.

Package

If you wish, you can enter in this field a package for the bean.

NOTE If you want the applet to be compatible with other development tools, the file path and package path should match.

Modifiers

Use the **Specifier** pop-up menu to choose the method specifier for the new bean. Possible specifiers include **None**, **Abstract**, and **Final**. The **Access** pop-up menu in this section indicates that you are creating a bean with **Public** access. No additional options are available for this pop-up menu.

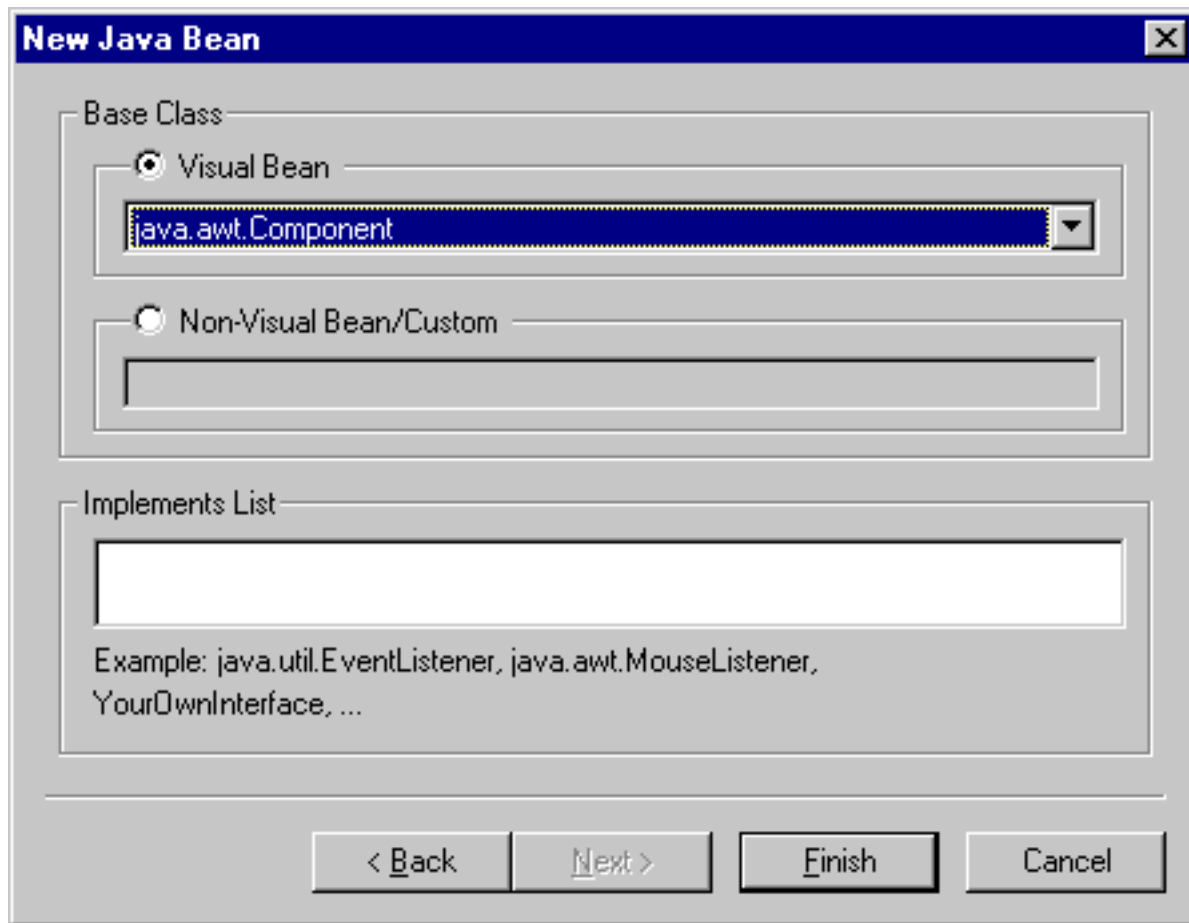
2. Specify a base class and a list of implementations.

This section of the wizard, shown in [Figure 12.12](#), lets you specify the base class and implementations for the new bean.

This section includes the following parts:

- [Base Class](#)
- [Implements List](#)

Figure 12.12 Java Bean wizard - Base Class and Interfaces



Base Class

The settings in this part determine the base-class type of the new bean. There are two possibilities:

- **Visual Bean**—Enable this option if the base class is a visual bean. The pop-up menu below this option lists common AWT and Swing base classes. Choose a base class from this pop-up menu.
- **Non-Visual Bean/Custom**—Enable this option if the base class of the event set is a non-visual bean or a custom class. Type the name of the base class in the field below this option.

Implements List

In this part, type a list of interfaces to be implemented by the base class. Example interfaces are provided. Separate multiple interfaces with commas.

Working with Designs

The IDE uses designs to organize and manage RAD projects. A design includes a particular set of user interface layouts and components that are part of the application. A CodeWarrior project can include several designs. Each design can include several build targets.

This section discusses the following topics:

- [Adding Designs to a Project](#)
- [Design View in the Project Window](#)
- [Designs in the Targets View](#)

Adding Designs to a Project

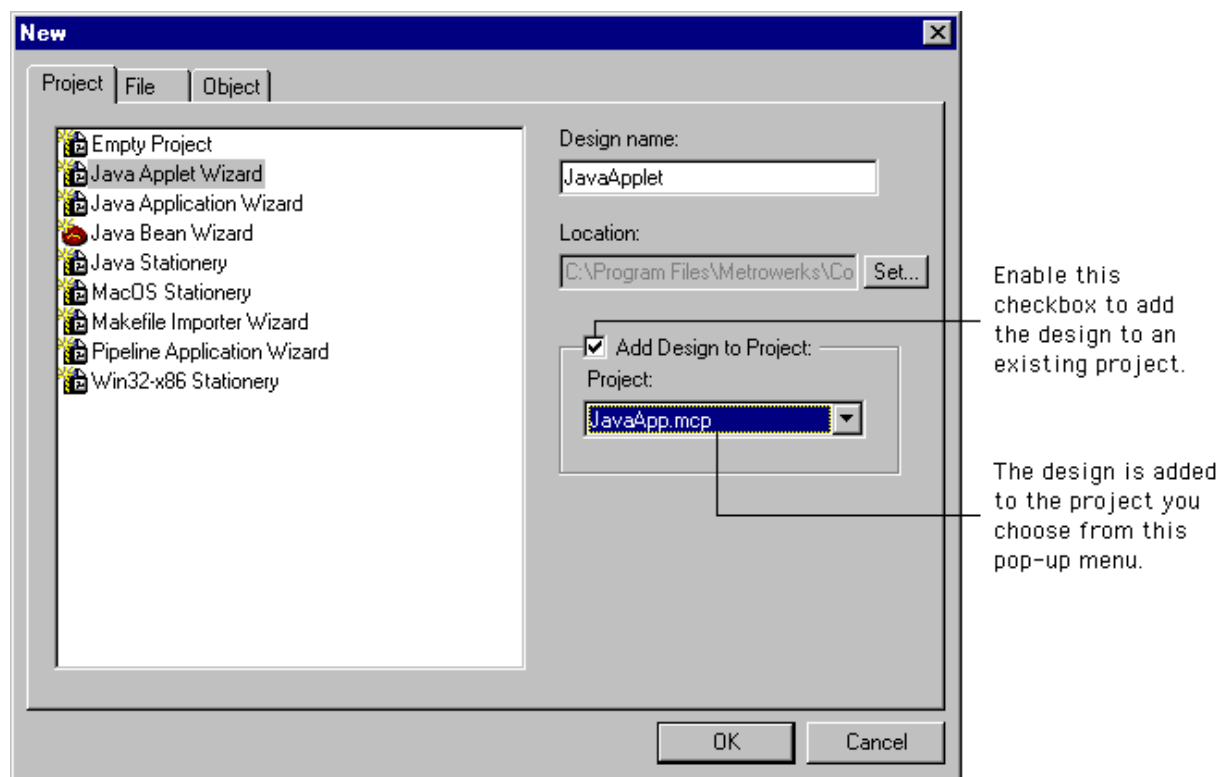
To add a design to an open project in the IDE, choose **New** from the File menu. The IDE displays the New window, shown in [Figure 12.13](#). Click the **Project** tab at the top of the window to display a list of available wizards.

The wizards you use to create designs are the same wizards you use to create projects:

- [Java Applet Wizard](#)—helps you create a Java applet. The wizard asks for information about the applet's class, HTML page, parameters, and description.
- [Java Application Wizard](#)—helps you create a Java application. The wizard asks for information about the application's class, frame class, and description.
- [Java Bean Wizard](#)—helps you create a Java bean. The wizard asks you for information about the bean's class, location, package, and modifiers.

For example, if you want to create a Java applet design, select the **Java Applet Wizard**.

Figure 12.13 Adding a design to a project



To add the new design to the project, enable the **Add Design to Project** checkbox. Because you are adding the design to an existing project rather than creating a new project, the **Location** field grays out, as shown in [Figure 12.13](#). Type a name for the new design in the **Design Name** field.

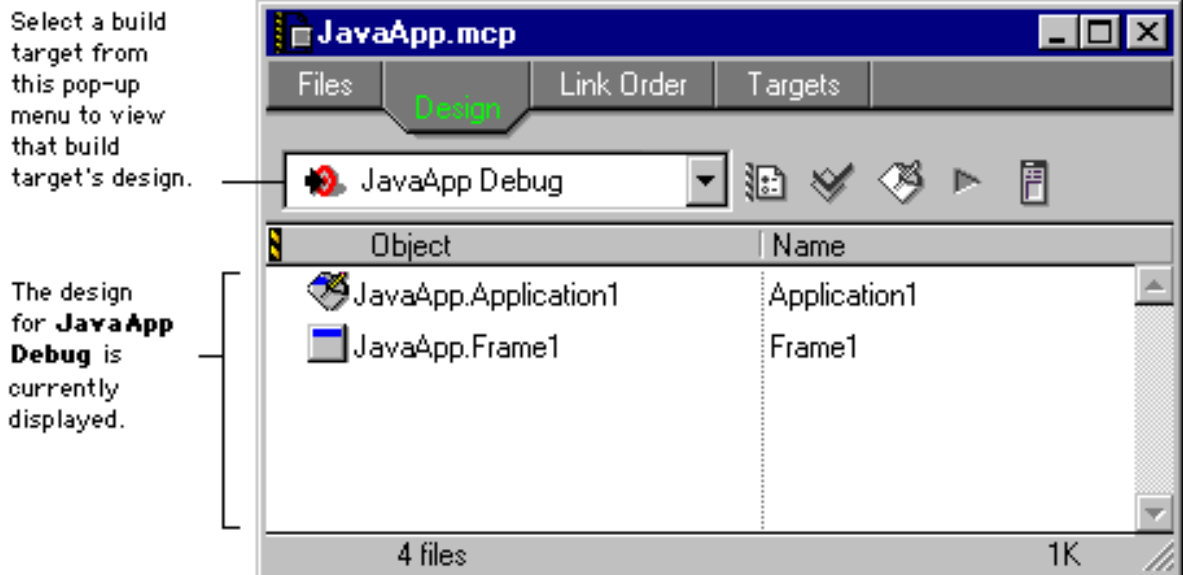
Select the project's name from the **Project** pop-up menu, as shown in [Figure 12.13](#). The design is added to the project you choose from this pop-up menu.

After you click **OK**, the selected wizard opens and guides you through the design-creation process. See [“RAD Wizards” on page 515](#) for more information about each wizard. When you complete the wizard, the IDE adds the design to the project.

Each RAD project can have multiple designs. However, a single project file can have at most 255 build targets. Because each design can have several build targets, you should keep this limit in mind as you add designs to your project. Many of the suggestions in

[“Strategy for Creating Complex Projects” on page 94](#) apply when creating multiple designs for a single RAD project.

Figure 12.14 Design view in Project window



Design View in the Project Window

When you create a RAD project, click the **Design** tab in the Project window to display the Design view. This view, shown in [Figure 12.14](#), provides information about the design for the active build target. For example, [Figure 12.14](#) shows the design for the **Java Application** build target. To view another build target's design, select that build target's name from the Current Target pop-up menu.

Some of the objects displayed in the Design view are hierarchical. Click the hierarchical control next to an object to expand that object and view its contents. Click the control again to collapse the view.

Designs in the Targets View

Click the **Targets** tab in the Project window to view the hierarchy of build targets within each design, as shown in [Figure 12.15](#). A build target's icon includes a design badge (🔗) when it is part of the current design. For example, the **JavaApp Debug** build target in

[Figure 12.15](#) is part of the current design because its icon includes the design badge. Since the **JavaApp Release** build target is not part of the current design, its icon does not include the design badge.

Figure 12.15 Designs in the Targets view

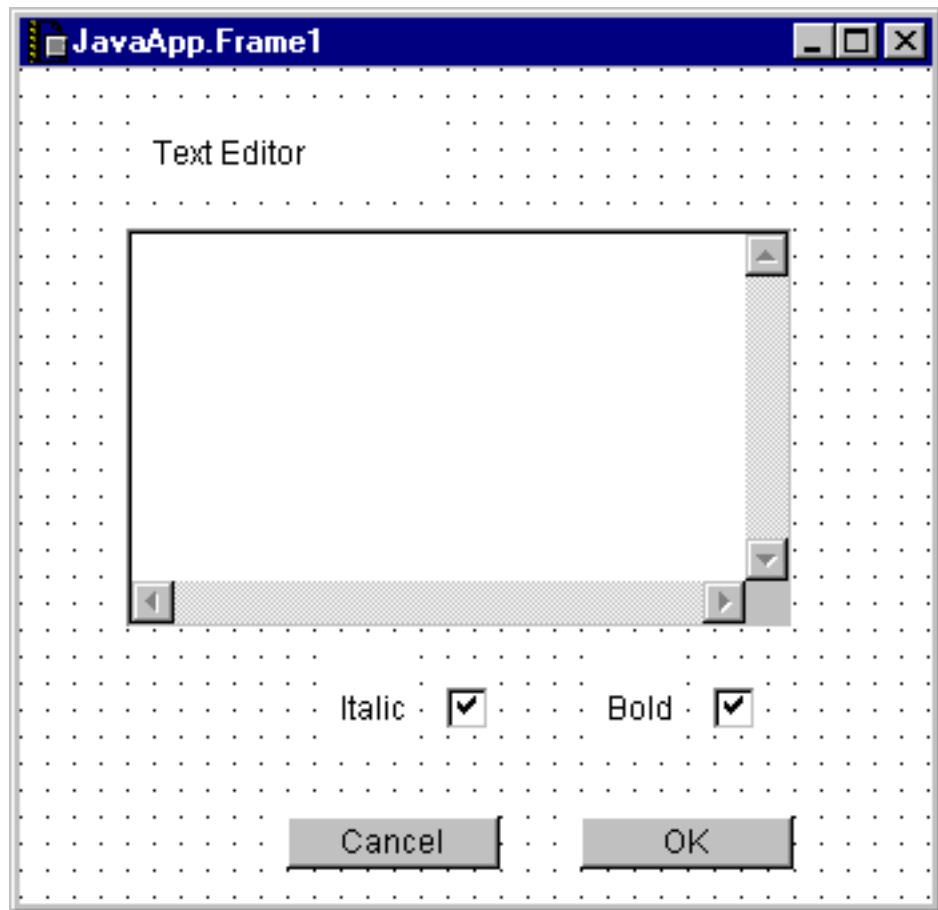


Working with Layouts

Each RAD project can include multiple layouts. A layout is a collection of user interface elements that comprise an application. A specific layout describes the content of a window, dialog box, or alert. Additionally, a layout describes the content of all the possible views in a particular window when you perform various actions.

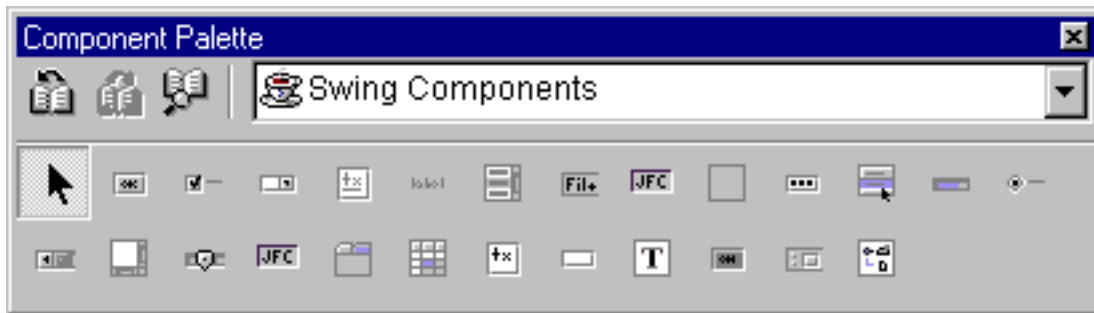
The IDE features a Layout editor that displays each layout in your project. The Layout editor uses the IDE's plug-in architecture to display the user interface. For example, when you create a Java RAD project, the IDE uses the Java RAD plug-in to display the user interface in the Layout editor. [Figure 12.16](#) shows a sample layout.

Figure 12.16 A layout in the Layout editor



The Layout editor works in conjunction with the catalogs to build an application. All of the available components for your RAD project are managed by catalogs. The Component Palette and the Component Catalog window let you view and use the contents of these catalogs. When you add a new interface element to the layout, you can drag a component from the Component Catalog window and drop that component into the layout. You can also select the same component from the Component Palette, which is shown in [Figure 12.17](#).

Figure 12.17 Component Palette



To learn how to create layouts and use the Layout editor, see [“RAD Layout Editing” on page 537](#).

[“Component Palette” on page 546](#) and [“Component Catalog window” on page 550](#) describe the two catalog tools you use with the Layout editor.

[“RAD Components” on page 569](#) describes the individual components inside the catalogs included with CodeWarrior.

RAD Layout Editing



CodeWarrior's rapid application development (RAD) tools include a Layout editor, Component Palette, Component Catalog window, and Object Inspector. These tools help you visually build an application's user interface. This chapter describes the four tools and their use with RAD.

The following topics are discussed:

- [Layout Editor](#)
- [Layout Wizards](#)
- [Component Palette](#)
- [Component Catalog window](#)
- [Object Inspector](#)

Layout Editor

The Layout editor displays the graphical user interface for your RAD project. The IDE uses a RAD plug-in architecture to display the Layout editor's environment. For example, when you develop a Java application with the RAD tools, the IDE uses the Java plug-in to draw the layout on the screen.

This section describes the following:

- [Creating a Layout](#)
- [Modifying a Layout](#)

Creating a Layout

When you create a new layout for your RAD project, the layout's default state does not contain any user interface components. You use the Component Catalog window or the Component Palette to

place user interface components in the layout. As you build the application's interface, the RAD tools automatically generate source code to create that interface at runtime.

Each RAD design can have several layouts. However, to maintain peak performance, you should limit the number of layouts you use in a single project. If your project includes more than 30 layouts, you should consider other ways to organize those layouts. Many of the suggestions in [“Strategy for Creating Complex Projects” on page 94](#) apply when creating multiple layouts for a single RAD project.

If you want to add a new layout to an open project in the IDE, choose **New** from the File menu. In the New window, click the **Object** tab to display a list of object wizards, as shown in [Figure 13.1](#).

Choose a wizard that corresponds to the layout you wish to create:

- [Java Frame Wizard](#)—helps you create a new layout frame for a Java RAD project. The wizard asks for information about the frame's class, package, and location.

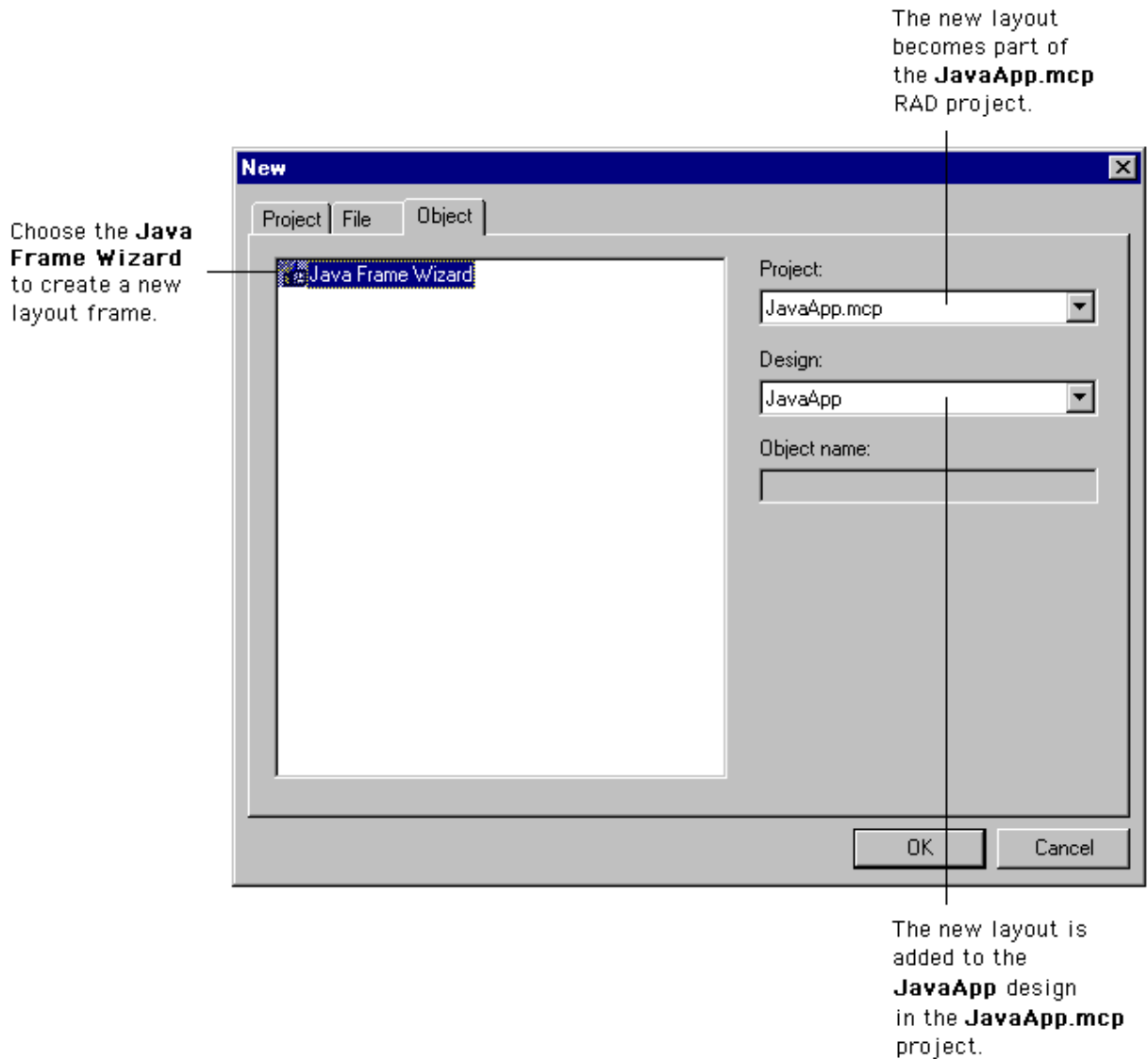
From the **Project** pop-up menu, select the project to which you want to add the new layout. Use the **Design** pop-up menu to select the particular design in the project to which you want to add the new layout.

When you click **OK**, the wizard guides you through the layout-creation process. After you configure the new layout's settings in the wizard, click **Finish** to add the layout to the project and open a new layout window. For detailed information about the available layout wizards, see [“Layout Wizards” on page 544](#).

Modifying a Layout

To modify an existing layout, you use the Layout editor in conjunction with the Component Palette, Component Catalog window, and Object Inspector. You select components from the Component Catalog window or the Component Palette and then place those components in the layout window. You manipulate the components in the layout as you would manipulate graphic objects in a drawing application.

Figure 13.1 **Creating a new layout**



A layout is modified in the following ways:

- [Creating objects](#)
- [Manipulating objects](#)
- [Examining objects](#)
- [Removing objects](#)
- [Layout editor contextual menu](#)

Creating objects

The RAD tools access available project components from the current catalog. The Component Palette and Component Catalog window give you two convenient ways to access the components.

If you use the Component Catalog window, you drag components to the layout window to create new interface objects. If you use the Component Palette, you click the button representing the component you want to add, and then you drag a cursor in the layout window to place that component in the layout.

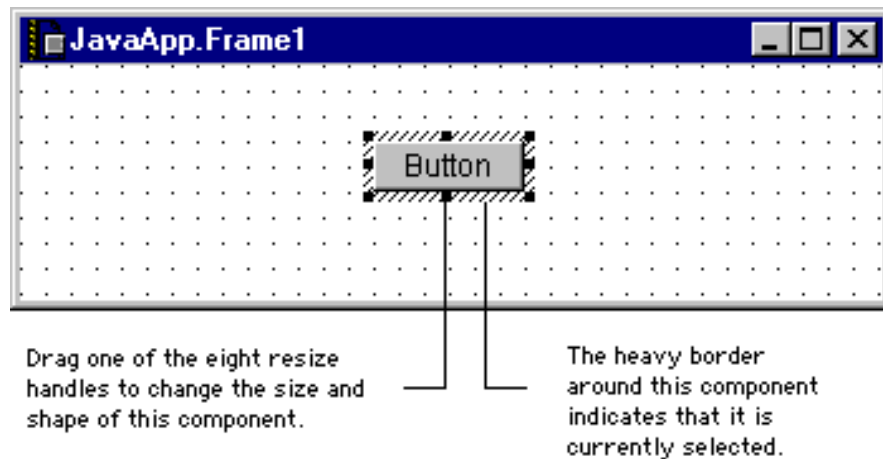
For more information about the Component Palette, see [“Component Palette” on page 546](#). To learn more about the Component Catalog window, see [“Component Catalog window” on page 550](#).

Manipulating objects

To move an object in a layout, select it in the layout window. A selected object is shown in [Figure 13.2](#). The heavy border around the object indicates that it is currently selected. Drag selected objects to move them to a new positions in the layout window. Alternatively, use the arrow keys on your keyboard to move the selected objects. To resize or reshape an object, drag one of its resize handles in the desired direction. A heavy outline is displayed as you drag the resize handle. This outline represents the final shape of the object when you release the resize handle.

You can select multiple objects in two different ways:

- Shift-click each object that you want to add to your selection. A heavy border is displayed around all selected objects. If you decide against an object in the selection, Shift-click that object again.
- Draw a marquee around a group of objects to quickly select them, as shown in [Figure 13.3 on page 542](#). Position the cursor near the group of objects you want to select, then drag the cursor to display a marquee. Release the marquee when it encloses the objects you want to select. To add additional objects to the selection, position the cursor near the additional objects and then press the Shift key while dragging the cursor. This action displays another marquee. Release the marquee when it encloses the additional objects.

Figure 13.2 A selected object

You can also manipulate objects with the Layout editor contextual menu. To learn more, refer to [“Layout editor contextual menu” on page 543](#).

TIP The Tab key on your keyboard is useful for layouts with several objects. After you select a particular object in the layout window, press the Tab key to quickly select a different object. The order in which the Tab key selects objects is based on the order in which those objects were placed in the layout window.

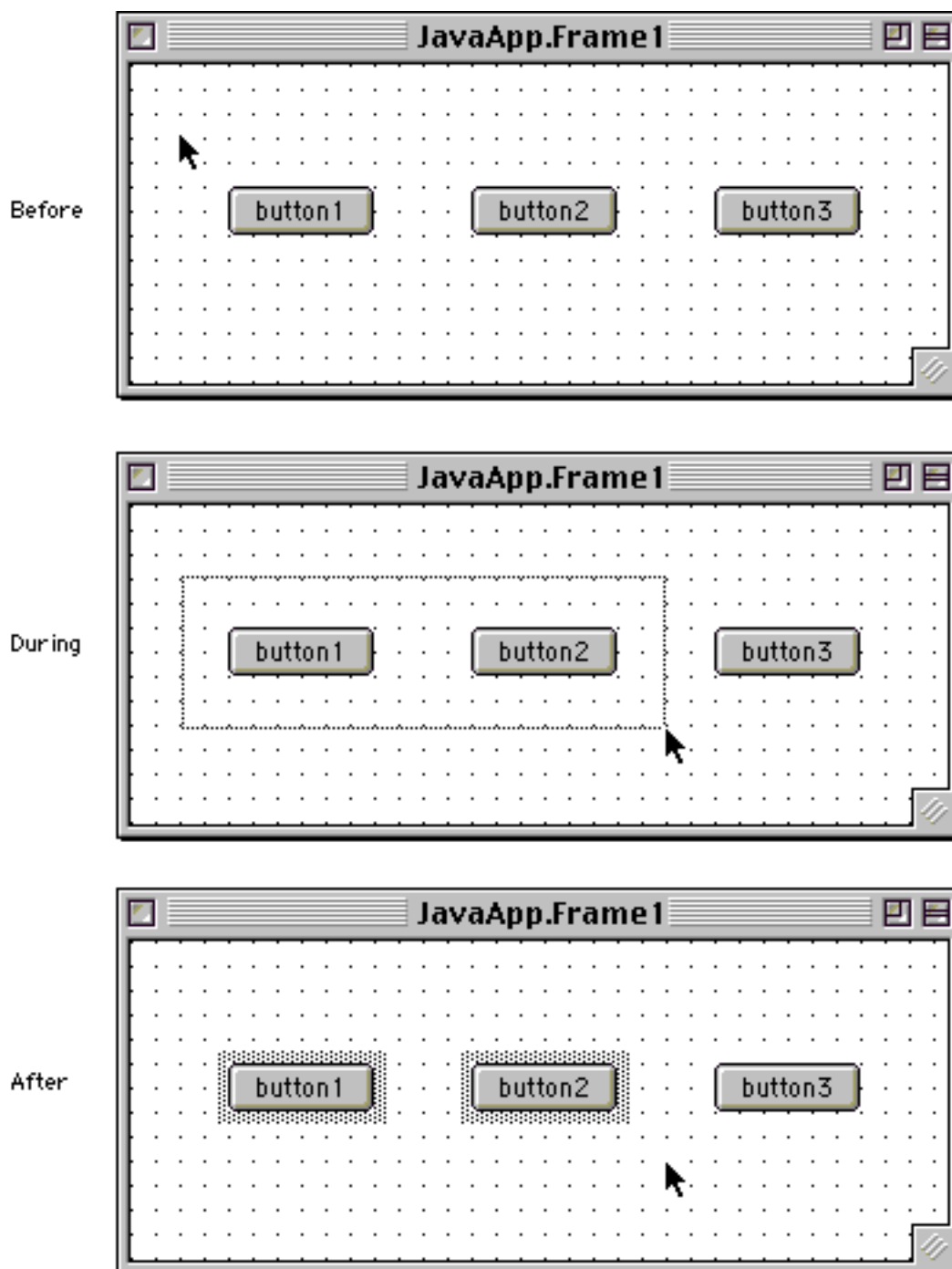
Examining objects

After you place objects in the layout, you can edit the various properties that characterize each object, as well as the events handled by that object. The Object Inspector displays the available properties and events for a selected object. For more information, see [“Object Inspector” on page 562](#).

Removing objects

To remove objects in a layout, select them and press Delete. Alternatively, select the objects and choose **Delete** (Windows) or **Clear** (Mac OS) from the Edit menu. The Layout editor removes the selected objects from the layout.

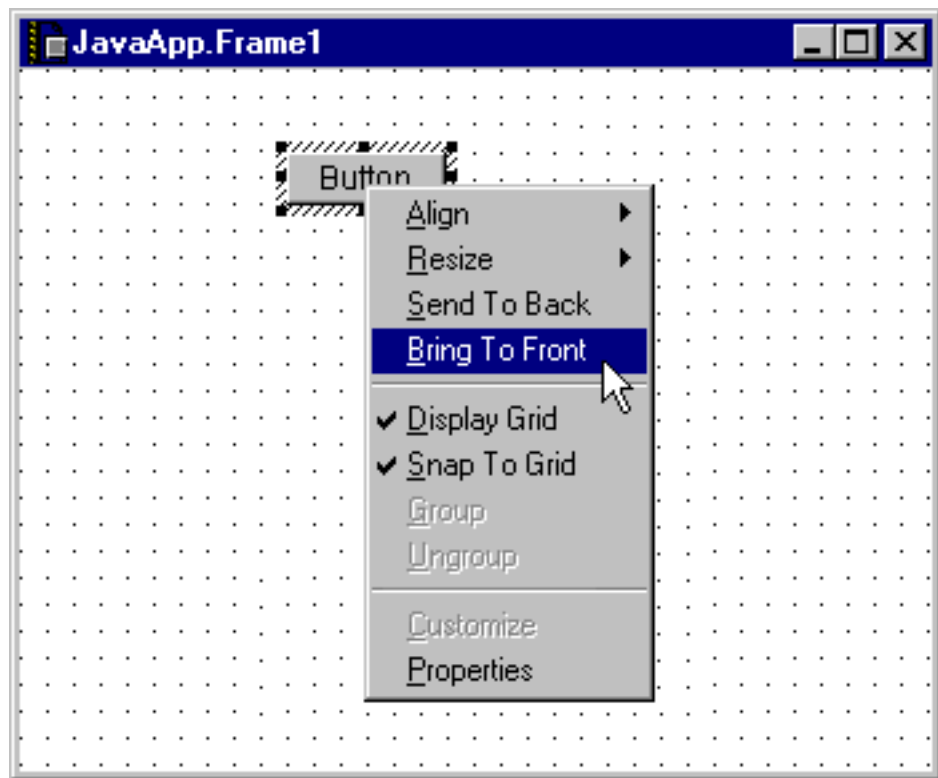
Figure 13.3 Marquee selection of two objects



Layout editor contextual menu

The Layout editor features a contextual menu that conveniently duplicates the commands in the Layout menu. The available commands in the contextual menu depend on the selected item. A sample contextual menu is shown in [Figure 13.4](#).

Figure 13.4 Layout editor contextual menu



To display the Layout editor contextual menu:

Windows Right-click an item.

Mac OS Control-click an item.

The following list describes some common graphical editing commands in the contextual menu:

- **Align**—displays a submenu from which you can choose different ways to position selected objects.
- **Resize**—displays a submenu from which you can choose different ways to change the size of selected objects.

- **Send To Back**—moves the selected objects so that they are displayed behind all other objects.
- **Bring To Front**—moves the selected objects so that they are displayed in front of all other objects.
- **Display Grid**—when checked, displays the graphical grid in the layout window.
- **Snap To Grid**—when checked, the [Layout Editor](#) automatically aligns objects with the graphical grid.
- **Group**—combines selected objects so that they move together as a group.
- **Ungroup**—separates a selected group so you can move each object independently.
- **Properties**—displays the [Object Inspector](#) to examine the properties of the selected object.

Layout Wizards

When you create a new layout for your RAD project, as described in [“Creating a Layout” on page 537](#), you select a layout wizard. The wizard helps you complete the layout-creation process. This section describes the following layout wizards:

- [Java Frame Wizard](#)

For beginners

Each wizard assumes you have a basic understanding of the associated programming language. For example, if you use the Java Frame wizard, you should understand the difference between an Abstract Window Toolkit (AWT) frame and a Swing frame. If the items displayed in the wizard are unfamiliar, you should consult additional programming language references for more information.

The layout wizards include the following navigation buttons:

- **Back**—return to the previous step
- **Next**—proceed to the subsequent step
- **Finish**—display a summary of current information
- **Cancel**—discard all changes

A wizard is divided into a series of steps. You progress through these steps in sequence, and each step builds on the information provided in previous steps. When you supply enough information in a particular step, click **Next** to continue.

You can modify the default settings in each step. To accept all current settings in the wizard, click **Finish**. The wizard displays a summary of all the current settings for the new project. Click **Generate** to accept the current settings and create the new layout, or click **Cancel** to return to the wizard and continue modifying settings.

Java Frame Wizard

The Java Frame wizard consists of one step:

1. **Describe the frame class for the Java frame.**

This section of the wizard, shown in [Figure 13.5](#), lets you specify the class name, package name, and location for the new frame. Other options are available to further describe the frame.

This section includes the following parts:

- [Class Name](#)
- [Package Name](#)
- [Location](#)
- [AWT frame](#)
- [Swing frame](#)

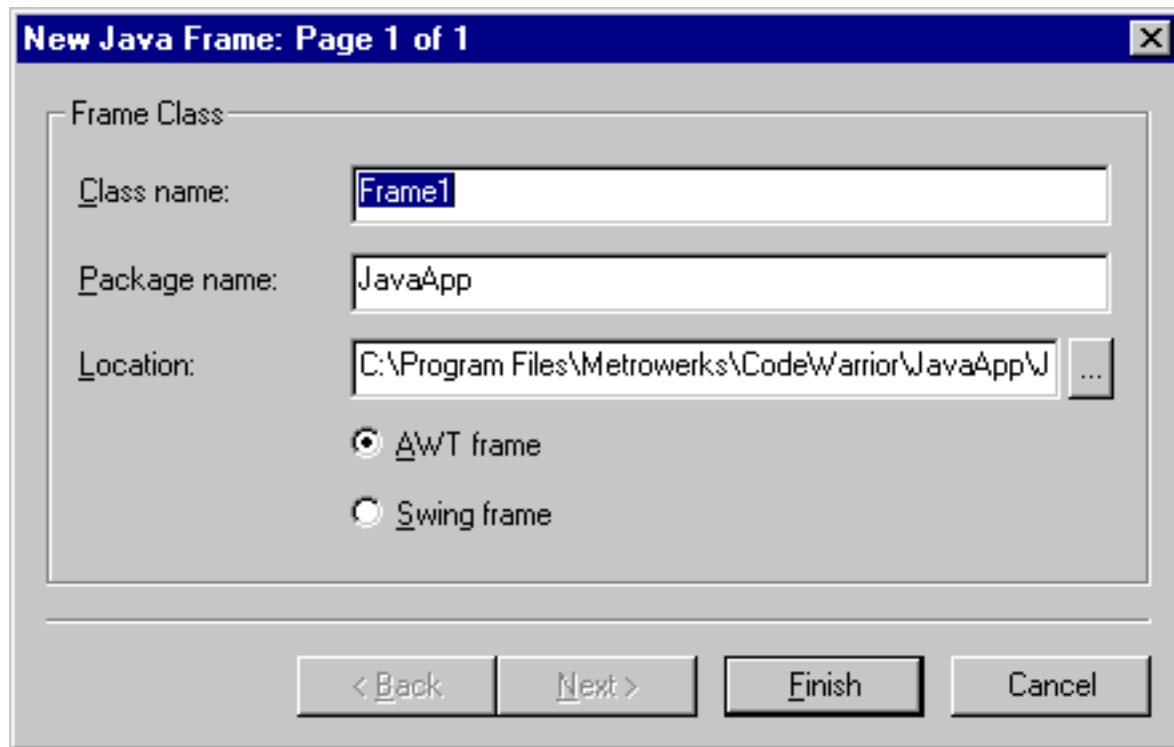
Class Name

Enter a name for the frame's class in this field. An example name is provided.

Package Name

If you wish, you can enter in this field a package for the frame class. A sample package name is provided.

Figure 13.5 Java Frame wizard - Frame Class



Location

In this field, type a location in which to save the frame class for the new layout. Alternatively, click the button next to the field. A standard window displays. Use the window controls to select a location.

AWT frame

Click this radio button to designate the new frame as an AWT frame. This radio button enabled by default.

Swing frame

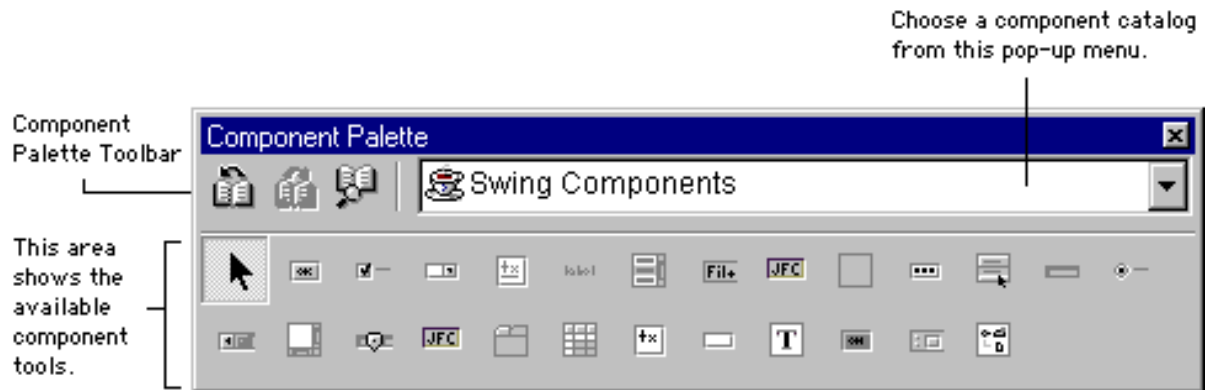
Click this radio button to designate the new frame as a Swing frame.

Component Palette

The Component Palette displays graphical tools that represent the components from the current catalog. These tools help you to

conveniently build the application's user interface. Choose **Component Palette** from the Window menu to display the Component Palette, shown in [Figure 13.6](#) (Windows) and [Figure 13.7](#) (Mac OS).

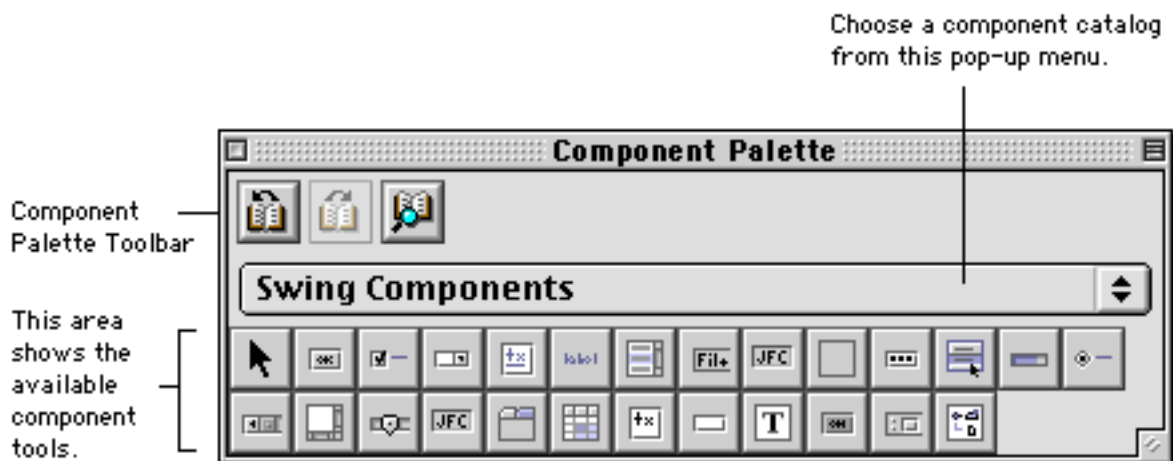
Figure 13.6 Component Palette (Windows)



The Component Palette has three main parts:

- [Component Palette Toolbar](#)
- [Catalog Pop-up Menu](#)
- [Component Tools](#)

Figure 13.7 Component Palette (Mac OS)






Component Palette Toolbar

The Component Palette toolbar provides convenient access to the components detailed in the Component Catalog window. [Table 13.1](#) describes each button in the Component Palette toolbar.

Catalog Pop-up Menu

The Component Palette displays component tools from a particular catalog. To use component tools from another catalog, choose the catalog's name from the Catalog pop-up menu, shown in [Figure 13.6](#) (Windows) and [Figure 13.7](#) (Mac OS).

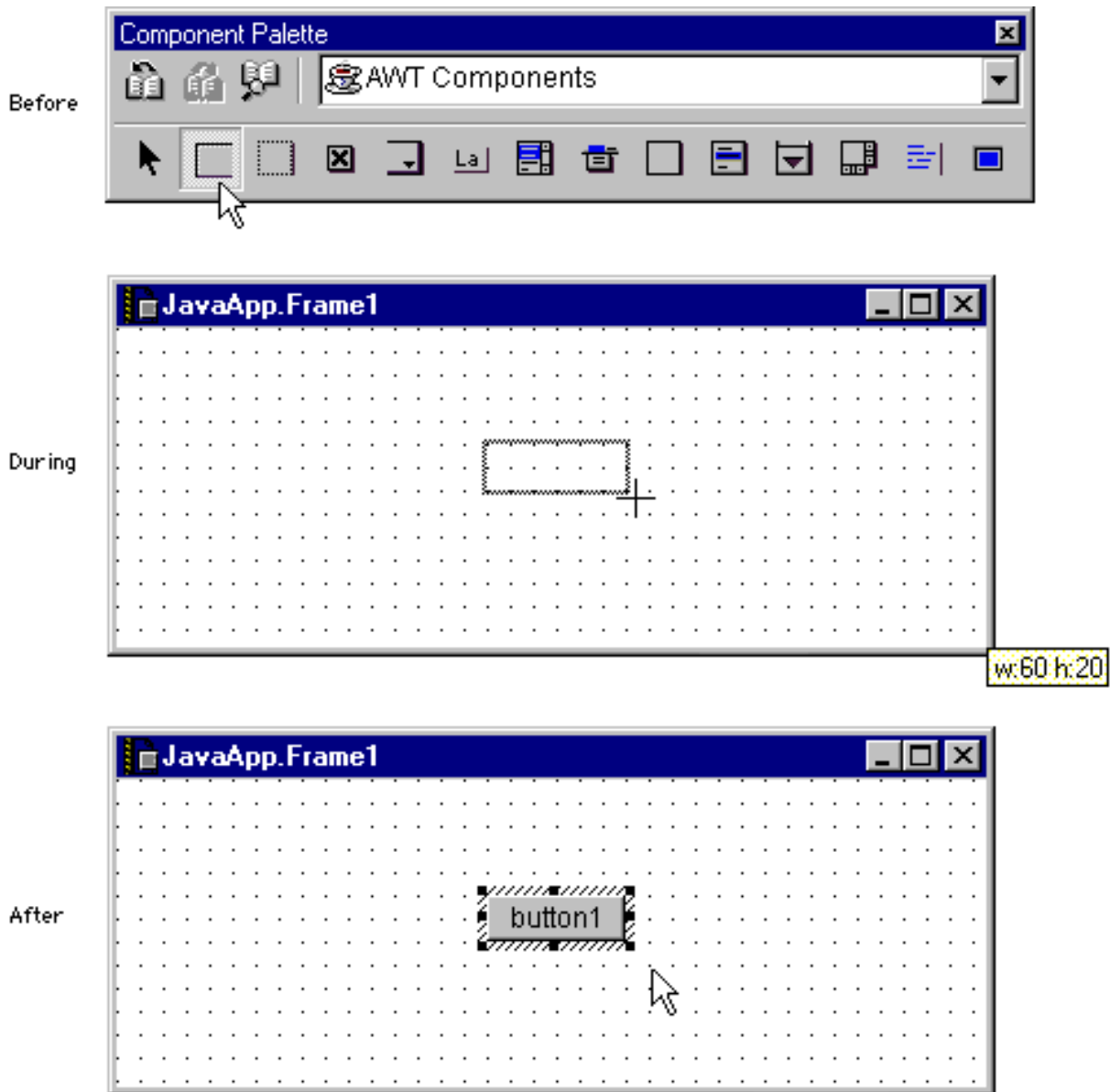
Table 13.1 Component Palette toolbar buttons

Button	Name	Description
	Open Catalog	Lets you open an additional catalog file and add it to the Component Palette.
	Close Catalog	Closes an added catalog file. The closed catalog is removed from the Component Palette and the Component Catalog window.
	Component Catalog	Opens the Component Catalog window.

Component Tools

The bottom of the Component Palette displays tools from the current catalog. You click a particular tool and then “draw” an object in the layout window. When you finish drawing the object, the Layout editor updates the layout window display. The process is illustrated in [Figure 13.8](#).

Figure 13.8 Drawing an object in the layout window



To use the Component Palette, do the following:

1. **Click the tool representing the component you want to place in the layout window.**



If you decide against using the tool you selected, click the Unselect tool. After you click this tool, you can click a different component tool to use in the layout window.

2. Drag the cursor while inside the layout window.

As you drag the cursor, the Layout editor displays a heavy outline to indicate the final position and shape of the component in the layout.

Windows The cursor's icon changes to a crosshair during this step. While you drag the crosshair, a pop-up window displays the layout window's coordinate system. If you decide against using the component tool while dragging the crosshair, press the Esc key. The Layout editor dismisses the component tool and removes the heavy outline from the layout window.

3. Release the outline.

The component is placed in the layout window. You can modify the component's properties and events using the [Object Inspector](#).

For more information about the various component tools in the Component Palette, refer to ["RAD Components" on page 569](#).

TIP (Windows) If you position the cursor over a component tool and wait briefly, the IDE displays the name of that tool. Use this feature to quickly distinguish the various tools.

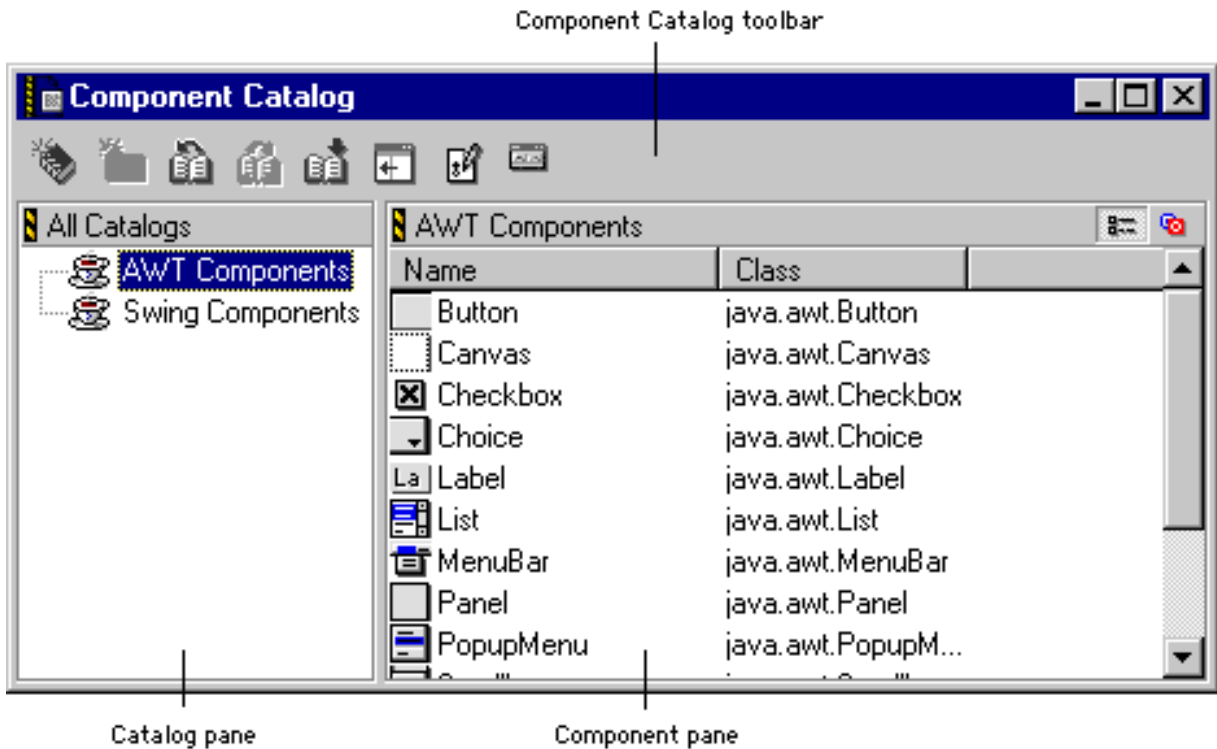
Component Catalog window

The Component Catalog window lets you drag and drop components from the current catalog onto the layout window. You can also view existing catalogs, import additional catalogs, and create new catalogs. Choose **Component Catalog** from the Window menu to display the Component Catalog window, shown in [Figure 13.9](#).

This section describes the following:

- [Component Catalog Toolbar](#)
- [Catalog Pane](#)
- [Component Pane](#)
- [Component Catalog Contextual Menu](#)
- [Creating Component Catalogs](#)



Figure 13.9 Component Catalog window









Component Catalog Toolbar

The Component Catalog toolbar lets you view and manipulate catalogs and their contents. [Table 13.2](#) describes the buttons in the toolbar.

Table 13.2 Component Catalog toolbar buttons

Button	Name	Description
	New Catalog	Creates a new catalog file to store a customized collection of components
	New Folder	Creates a new folder within your own catalog so you can further organize components

Button	Name	Description
	Open Catalog	Opens a catalog file and adds it to the Catalog pane
	Close Catalog	Closes a catalog file that was added to the Catalog pane
	Import Components	Imports components into the active catalog
	Toggle Index View	Displays or hides the Catalog pane in the Component Catalog window
	Edit Item Properties	Opens the Object Inspector for the selected item so you can edit that item's properties
	Component Palette	Displays the Component Palette and makes it frontmost

Catalog Pane

The Catalog pane, shown in [Figure 13.9 on page 551](#), displays information about available catalogs, including the customized catalogs you create or import. Click the hierarchical control next to a catalog to expand it and view its contents. Click the control again to collapse the view. Alternatively, double-click an item to expand or collapse its view in the Catalog pane.

You can create customized catalogs to store the components you use most frequently. These customized catalogs make it easier for you to build your application's layouts. To learn how to create your own catalogs, see ["Creating Component Catalogs" on page 558](#).

TIP If you tend to work with the components in a single catalog, you can hide the Catalog pane by clicking the **Toggle Index View** button in the Component Catalog toolbar.

Component Pane

The Component pane, shown in [Figure 13.9 on page 551](#), displays various information about the items in each catalog. The name of the catalog is displayed in the upper-left corner of the pane. You drag components directly from the Component pane into a layout window to create user interface objects.

The Component pane is discussed in the following topics:

- [Content View buttons](#)
- [Component Information Bar](#)
- [Component list](#)
- [Sorting and resizing component lists](#)

Content View buttons

The Content View buttons toggle the display of the components in the current catalog:



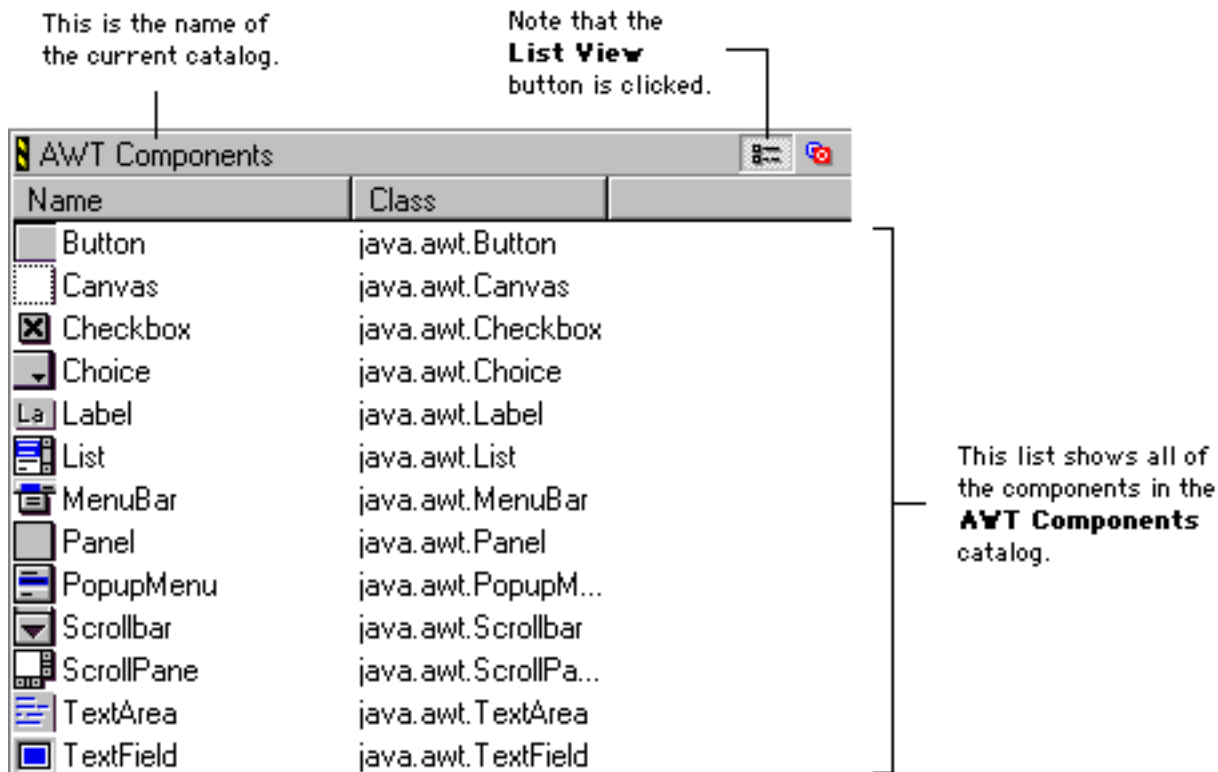
Click the **List View** button to display the contents of the Component pane as a list. The List view, shown in [Figure 13.10](#), displays all the components in the current catalog.



Click the **Live View** button to display the contents of the Component pane as live objects. The Live view, shown in [Figure 13.11](#), is only available for component catalogs that you create. Refer to [“Creating Component Catalogs” on page 558](#) for more information.

NOTE If you try to use the Live view for a catalog that you did not create, the IDE displays the error message “No contents to show for *Catalog*,” where *Catalog* is the name of the selected catalog in the [Catalog Pane](#).

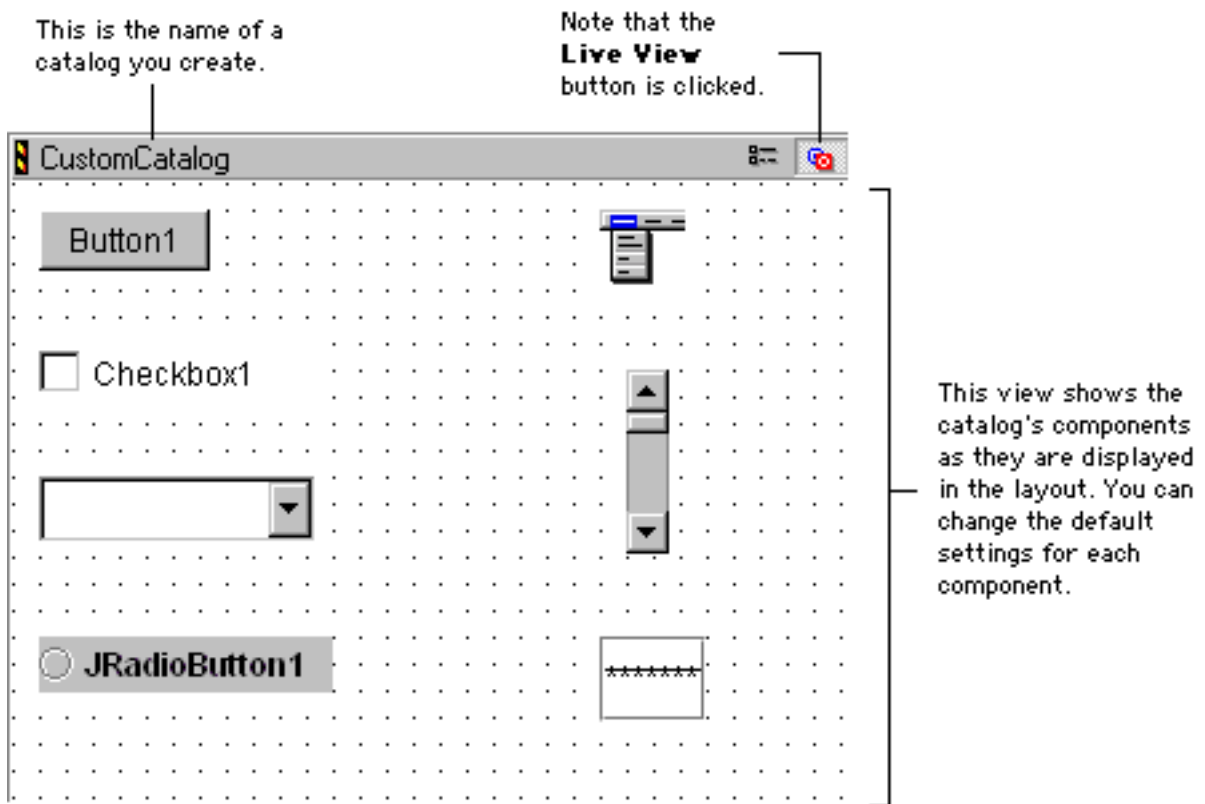
Figure 13.10 Component pane - List view



The two views in the Component pane help you accomplish different tasks. The List view is useful for quickly browsing a catalog's components. The Live view shows graphical representations of your own catalog's components.

The Live view also helps you create a customized set of components for your application's layouts. For example, you can change the default size and shape of a component in the Live view. When you use the component to create a new interface object in the layout window, the new object retains the size and shape of the component.

Figure 13.11 Component pane - Live view

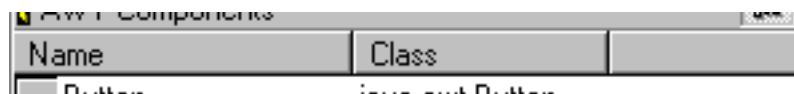


Component Information Bar

The List view of the Component pane includes a Component Information Bar, shown in [Figure 13.12](#). This information bar helps you organize the components in the current catalog.

You can sort the component list according to the columns in the Component Information Bar. You can also resize each column in the list. For more information, see [“Sorting and resizing component lists” on page 557](#).

Figure 13.12 Component Information Bar




[Table 13.3](#) details the columns displayed in the Component Information Bar for various catalog types.

Component list

The component list, shown in [Figure 13.10 on page 554](#), displays all components in the current catalog and their associated Component Palette button icons. To place a particular component in your layout, drag it from the component list in the Component Catalog window and drop it into the layout window. As you drag the component inside the layout window, a heavy outline indicates the final position of the component. After you release the outline, the component is placed in the layout window.

Table 13.3 Component Information Bar items

Catalog Type	Column	Description
Java catalogs (AWT Components , Swing Components)	Name	The name of the component.
	Class	The class of which the component is a member.
Customized catalogs (created or imported)	Name	The name of the component.
		Identifies whether the component is locked.
	Modified	The date the component was last modified.
	Class	The class of which the component is a member.
	Comments	Shows comments for the component.

If you decide against using the component while dragging it in the layout, just drag the component outside the layout window and release it. After releasing the component, you can go back to the component list and drag another component to the layout window.


For more information about removing objects from your layouts, see [“Removing objects” on page 541](#). Refer to [“Layout editor](#)

[contextual menu” on page 543](#) for more information about contextual menu commands. To learn about the components in the component list, see [“RAD Components” on page 569](#).

Sorting and resizing component lists

You can sort the components in the List view according to the columns in the Component Information Bar, shown in [Figure 13.12 on page 555](#).

Click the column for which you want to sort the components. Each time you click the column, you toggle the sort order between ascending order and descending order. For example, if you want to sort the components by name, click the **Name** column in the Component Information Bar. Click the column again to sort the components in descending order.

-  To resize the columns in the Component Information Bar, position the cursor between two column titles. The cursor’s icon changes to the icon shown at left. Drag and release this cursor to resize the columns.

Component Catalog Contextual Menu

The Component Catalog window features a contextual menu that gives you convenient access to several menu commands. The commands displayed in this menu depend on the item you select. A sample contextual menu is shown in [Figure 13.13](#).

To display the Component Catalog contextual menu:

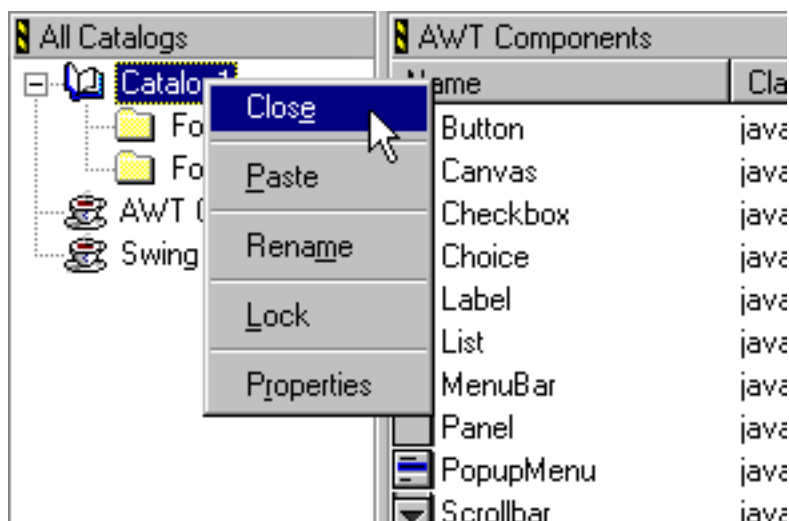
Windows Right-click an item.

Mac OS Control-click an item, or click and hold on an item.

The following list describes the most common commands in this contextual menu:

- **Close**—removes the item from the Catalog pane.
- **View**—changes the current view in the Component pane. You can choose List view or Live view.
- **New Folder**—creates a new folder in the current catalog.

Figure 13.13 Component Catalog contextual menu



- **Rename**—changes the name of the selected item. After you select this command, type a new name for the item and press Enter/Return.
- **Lock**—locks the selected item. When an item is locked, you cannot change any of its properties (except its “locked” status). However, you can add sub-items to a locked container (such as a folder or catalog), remove sub-items from a locked container, and modify sub-items of a locked container (assuming the items to be modified are not locked). A small lock icon indicates that an item is currently locked.
- **Unlock**—unlocks the selected item.
- **Properties**—displays the properties for the selected item in the Object Inspector.

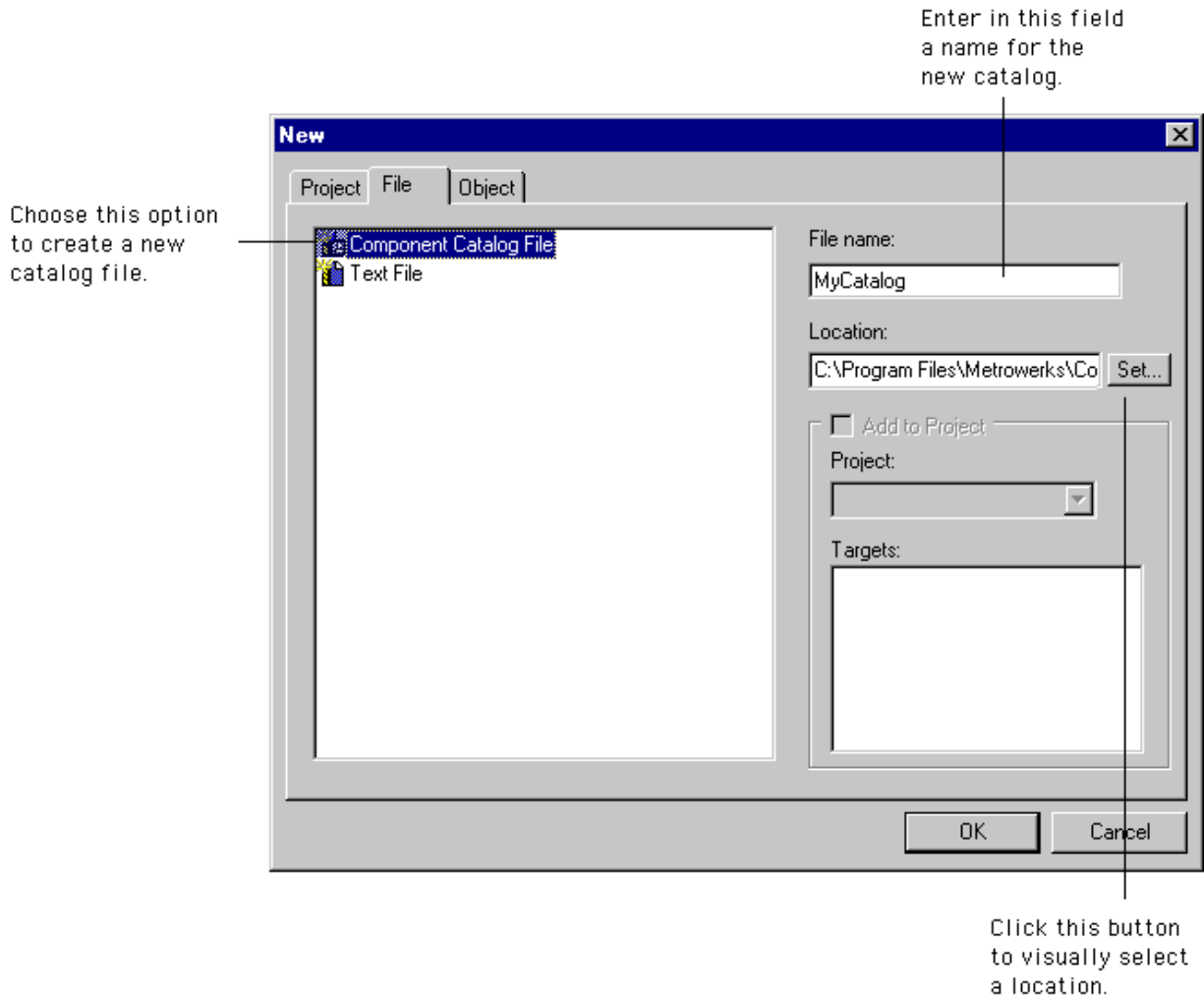
Creating Component Catalogs

You can create your own customized component catalogs to store the RAD components you use most frequently. These catalogs let you combine components from several catalogs into a single catalog file. Whenever you need to create a new layout, your customized catalog file can provide all the components you need.

To create a catalog, click the **New Catalog** button in the toolbar of the [Component Catalog window](#). Alternatively, choose **New** from the File menu, click the **File** tab at the top of the New window, and

then select **Component Catalog File** in the File view. You see the window shown in [Figure 13.14](#).

Figure 13.14 Creating a new catalog file



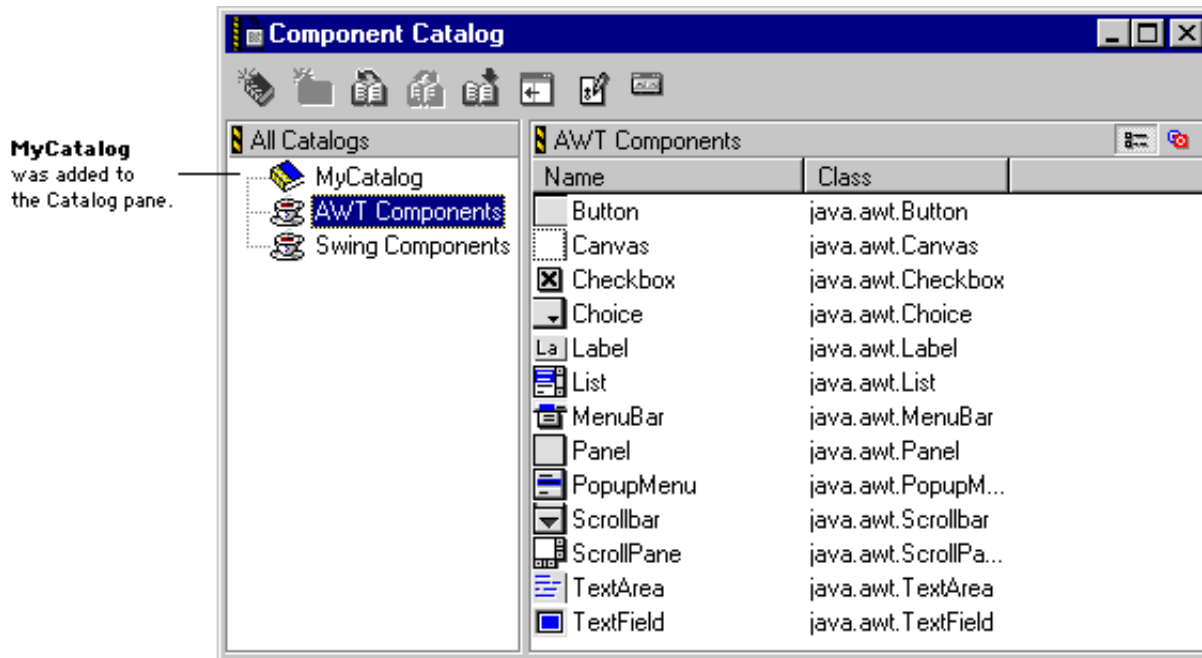
Type a name for your new catalog file in the **File name** field.

TIP We suggest naming your catalog with a `.ctlg` file name extension, like this: `MyCatalog.ctlg`. This naming convention helps you quickly identify the catalog file on your hard disk. In addition, the Windows-hosted version of the CodeWarrior IDE uses this extension to identify the catalog file.

The **Location** field displays the full path to the folder in which the catalog file is saved. To change the current path, type a new path directly into the field. Alternatively, click the **Set** button to the right of the field to display a dialog box. Use the dialog box controls to navigate to a location on your hard disk where you want to save the project.

When you finish typing a name for the catalog and choosing a location in which to save that catalog, click **OK** in the New window. The IDE displays the [Component Catalog window](#) and adds the new catalog to the [Catalog Pane](#), as shown in [Figure 13.15](#).

Figure 13.15 A new catalog added to the Catalog pane



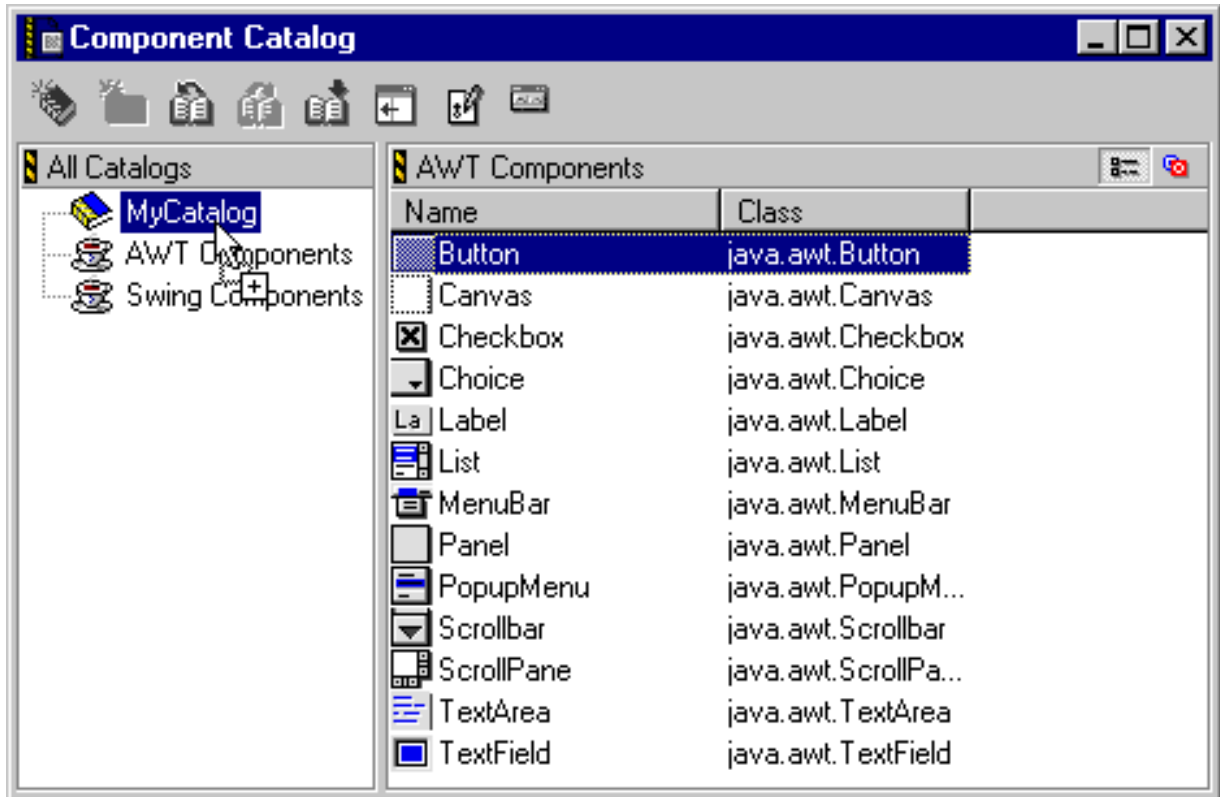
To add a component to your new catalog, follow these steps:

1. **Select an existing catalog with the component you wish to add.**
For example, if you want to add the Java AWT Button component to your catalog, select **AWT Components** in the Catalog pane.
2. **Select the component you wish to add from the Component pane.**
In the example, you would select the **Button** component.

3. Drag and release the selected component onto your catalog.

In the example, you would drag the **Button** component onto **MyCatalog** in the Catalog pane, as shown in [Figure 13.16](#). After you release the **Button** component, it is added to **MyCatalog**.

Figure 13.16 Adding a component to your catalog



TIP To add several components simultaneously, Ctrl/Shift click the components before dragging them onto your catalog. To automatically expand the view of a hierarchical catalog, drag the components onto the catalog and wait briefly.

You can also add components to your catalog using the **Cut**, **Copy**, and **Paste** commands in the Edit menu and the [Layout editor contextual menu](#). To use these commands, perform the following steps:

1. Select an existing catalog with the component you wish to add.
2. Select the component you wish to add from the Component pane.
3. Choose the Cut or Copy command.
4. Select the catalog to which you want to add the component.
5. Choose the Paste command.

Object Inspector

The Object Inspector displays the properties and events for a selected object. You can edit each object characteristic by using pop-up menus, dialog boxes, or by directly typing a new value. Choose **Object Inspector** from the Window menu to display the Object Inspector, shown in [Figure 13.17](#) (Windows) and [Figure 13.18](#) (Mac OS, Solaris, and Linux).

Figure 13.17 Object Inspector (Windows)

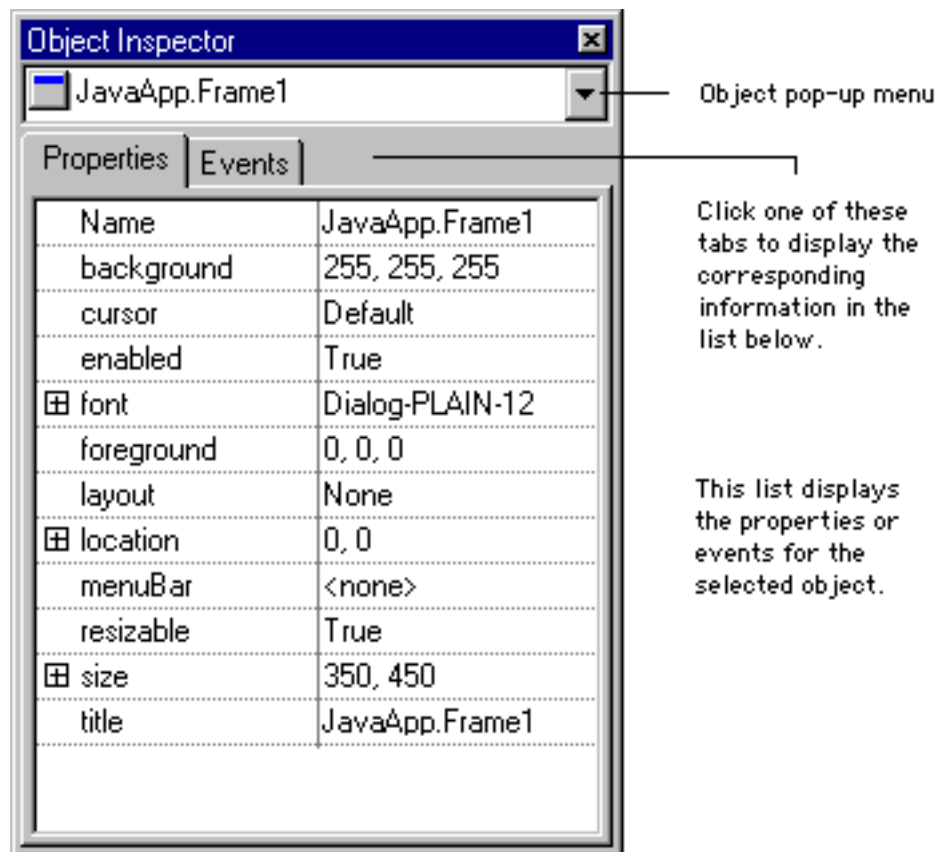
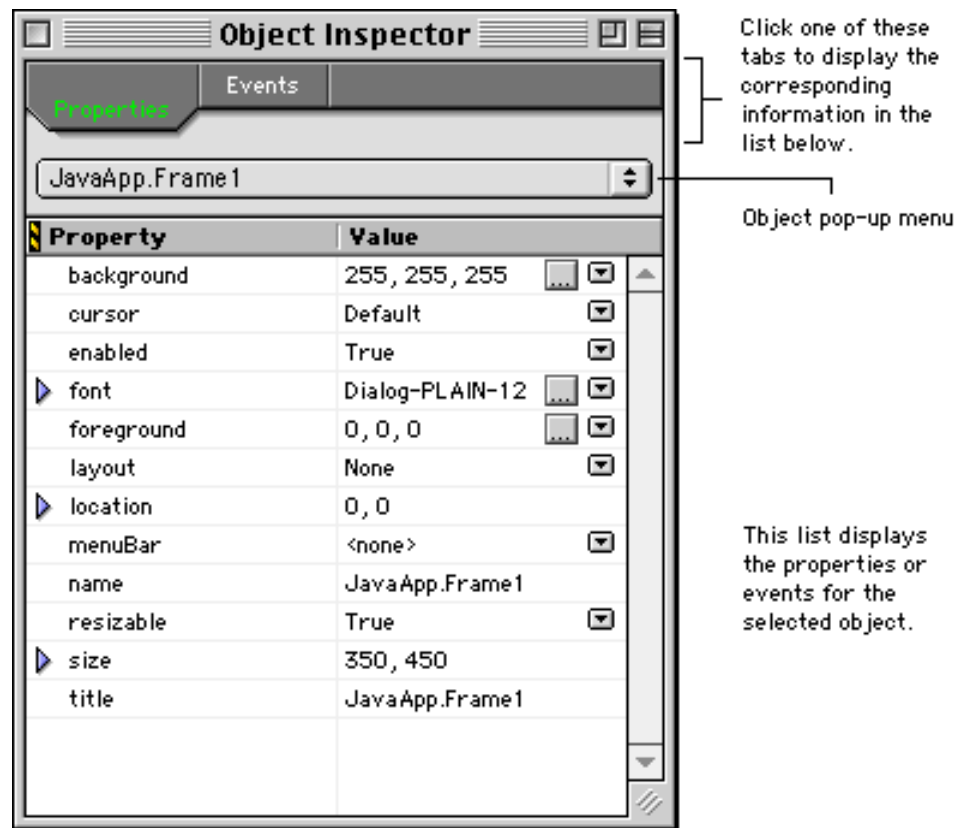


Figure 13.18 Object Inspector (Mac OS)



This section describes the following Object Inspector parts:

- [Object Pop-up Menu](#)
- [Properties Tab](#)
- [Events Tab](#)
- [Object Inspector Contextual Menu](#)

Object Pop-up Menu

The Object Inspector displays information about the object whose name is displayed in the Object pop-up menu. To inspect other items within the selected object's hierarchy, choose the item's name from the Object pop-up menu.

Windows The arrow keys on the keyboard let you quickly access other objects in the current layout. After clicking the Object pop-up menu, press the Up Arrow key to select the object listed above the

current one, or press the Down Arrow key to select the object listed below the current one. Press Enter/Return to confirm the selection and display the object's properties in the Object Inspector.

Properties Tab

To view the properties of the selected object, click the **Properties** tab. The Object Inspector displays a list of applicable properties. Each line in the list displays the name of a property and the current value of that property.

Click the hierarchical control next to a specific property to expand and view its contents. Click the control again to collapse the view. To resize the columns in the list, position the cursor between the columns and drag the cursor. An outline shows the final sizes of the columns. Release the cursor to resize the columns accordingly.

To change the value of a particular property, click its current value. A border is displayed around the line containing the selected property, and the property is highlighted. Type a new value to replace the current one.

Alternatively, you can change the value of a property by clicking buttons displayed next to the current value:



Click this button to display a pop-up menu. Choose pre-defined options from the pop-up menu to change the current property value.



Click this button to display a dialog box. Configure the property value using the options in the dialog box. The current RAD plug-in determines the contents of the dialog box.

If you decide against changing the property's value, press the Esc key on your keyboard. The Object Inspector discards your changes.

The arrow keys on the keyboard provide quick navigation of a component's properties. When a property is currently selected in the Object Inspector, the Up Arrow key selects the property listed above the current one, and the Down Arrow key selects the property listed below the current one.

TIP (Windows) The Left Arrow and Right Arrow keys are useful for toggling between pre-defined property values. Selecting a boolean value, for example, you can use the Left Arrow and Right Arrow keys to toggle the current value between True and False.

Events Tab

The **Events** tab displays information about the selected component's events. By default, all events are disabled. After you modify an event, the RAD tools automatically generate source code. This generated code provides a structured framework to handle the event. You supply the code that determines the way the object behaves and interacts with other objects in your user interface.

To modify a particular event, click its blank text field. The Object Inspector displays and highlights a default name for the event. If you want to change this default name, type a new name.

TIP You can quickly create a new event by double-clicking its blank text field. This double-click accepts the default name for the event.

After you make changes, press Enter/Return to let the RAD tools automatically generate source code. An editor window opens, displays the generated code, and highlights the name you chose for the event. [Listing 13.1](#) shows the generated code for a `componentResized` event in a Java AWT checkbox component.

Listing 13.1 Sample generated code

```
public
void checkbox1componentResized( java.awt.event.ComponentEvent e)
{
}
```

You must supply the code that specifies the component's behavior. For example, the generated code in [Listing 13.1](#) does not include any code inside the braces. You must create the code within these braces in order to control the component's behavior in your application.

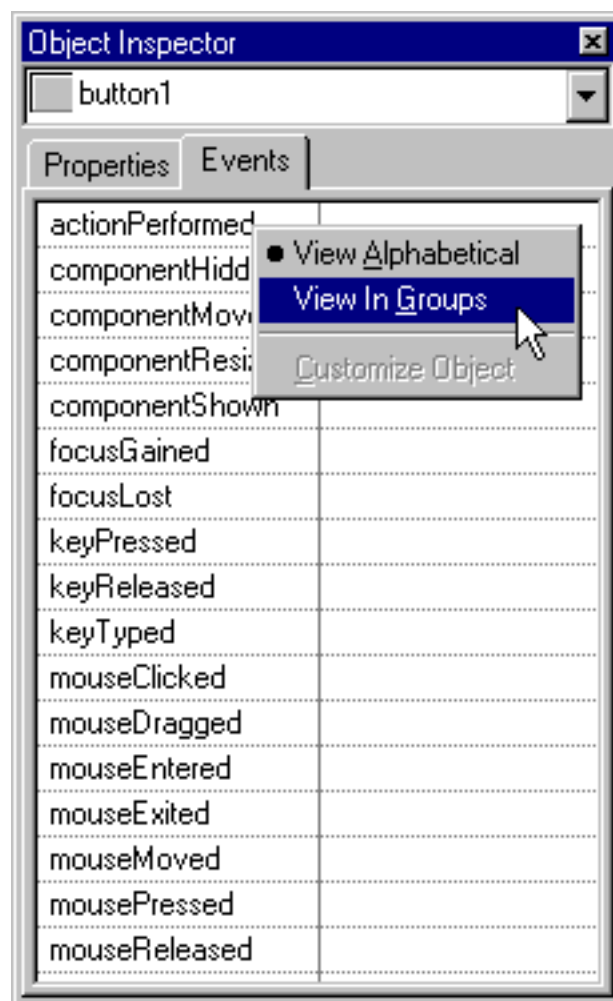
Choose **Save** from the File menu or press Ctrl/Command-S to immediately save your changes.

If you decide against modifying an event, press the Esc key on your keyboard. The Object Inspector discards your changes.

Object Inspector Contextual Menu

The Object Inspector contextual menu provides several convenient commands for manipulating objects. The available commands in this menu depend on the object you select. A sample Object Inspector contextual menu is shown in [Figure 13.19](#).

Figure 13.19 Object Inspector contextual menu



To display the Object Inspector contextual menu:

Windows Right-click an item.

Mac OS Control-click an item.

The following list describes some of the commands in the Object Inspector contextual menu:

- **Select All**—selects the contents of the current text field.
- **Delete**—removes the contents of the current text field.
- **Customize Object**—lets you change object characteristics.
- **View Alphabetical**—changes the Events view so that all object events are listed in alphabetical order, regardless of the groups to which those events belong.
- **View In Groups**—changes the Events view so that object events are displayed in group hierarchies.

RAD Components

CodeWarrior's rapid application development (RAD) tools use components to build an application's user interface. These components address specific elements of the interface, such as buttons, scroll bars, and checkboxes. This chapter describes the components supplied with the CodeWarrior RAD tools.




This chapter discusses the following:











- [Java AWT Components](#)
- [Java Swing Components](#)
- [Component Editors](#)

Java AWT Components

[Table 14.1](#) describes the components that belong to the Java **AWT Components** catalog. The icons shown in the following table are displayed in the [Component Palette](#) and [Component Catalog window](#).

Table 14.1 Java AWT components














Icon	Description
	Button—represents a button in the user interface that you click to yield a result
	Canvas—represents a container for other user interface elements
	Checkbox—represents a checkbox option that you enable or disable













Icon	Description
	Choice—represents a pop-up menu that you use to select an option from a list of possible options
	Label—represents an area of text that identifies or describes other user interface elements
	List—represents a scrollable list from which you choose one or more options
	MenuBar—displays an icon in the layout window, but not in the actual application. Double-click the icon to open the Menu Editor .
	Panel—represents a means by which to organize various arrangements of user interface elements
	PopupMenu—displays an icon in the layout window, but not in the actual application. Double-click the icon to open the Pop-up Menu Editor .
	Scrollbar—represents a scrollbar in the user interface that is used to adjust various options
	ScrollPane—represents a pane that can create scrollbars along its edges when entered text exceeds the pane's boundaries
	TextArea—represents an area in the user interface that lets you enter multiple lines of text
	TextField—represents an area in the user interface that lets you enter a single line of text

Java Swing Components

[Table 14.2](#) describes the components that belong to the Java **Swing Components** catalog. Many of these components are similar to those in the **AWT Components** catalog, but are more extensively configurable. The icons shown in the following table are displayed in the [Component Palette](#) and [Component Catalog window](#).

Table 14.2 Java Swing components

Icon	Description
	JButton—represents a button in the user interface that you click to yield a result
	JCheckBox—represents a checkbox option that you enable or disable
	JComboBox—represents a combination box that lets you select options either by typing the option's name or by choosing the option from a pop-up list
	JEditorPane—represents an editing pane that adapts its characteristics to the required editing task
	JLabel—represents an area of text that identifies or describes other user interface elements
	JList—represents a scrollable list from which you choose one or more options
	JMenuBar—displays an icon in the layout window, but not in the actual application. Double-click the icon to open the Menu Editor .
	JOptionPane—displays a dialog box that alerts the user, asks for additional input, or provides information
	JPanel—represents a means by which to organize various arrangements of user interface elements
	JPasswordField—represents a single line of text that lets you enter passwords without displaying onscreen the actual characters you type
	JPopupMenu—displays an icon in the layout window, but not in the actual application. Double-click the icon to open the Pop-up Menu Editor .
	JProgressBar—represents the visual display of a bar whose size grows in proportion to the percentage of a task being completed
	JRadioButton—represents a radio button that you enable or disable, one at a time

Icon	Description
	JScrollBar—represents a scrollbar in the user interface that is used to adjust various options
	JScrollPane—represents a pane that can create scrollbars along its edges when entered text exceeds the pane's boundaries
	JSlider—represents an onscreen mechanism, in the form of a slider control, for choosing various options
	JSplitPane—represents a means by which to control the size and display of two user interface elements
	JTabbedPane—represents a pane divided into several parts, with each part accessible through a tab control
	JTable—represents a table of rows and columns
	JTextArea—represents an area in the user interface that lets you enter multiple lines of text
	JTextField—represents an area in the user interface that lets you enter a single line of text
	JTextPane—represents an area of text with graphical attributes
	JToggleButton—represents a button that has two possible states
	JToolBar—represents a toolbar in the layout window
	JTree—represents a hierarchical control in the user interface

Component Editors

Certain components display representative icons of the user interface, rather than the actual interface. These representative icons appear in the layout window only, and are not displayed in the

application. Such components require separate editors to modify the way they are displayed in the application.

For example, the Java AWT MenuBar component produces a square icon in the layout window. This icon does not show the menu bar as it would appear in the application. Instead, you double-click the icon to open the Menu editor. The editor shows the true form of the menu bar.

This section describes the various component editors in the catalogs supplied with the CodeWarrior IDE. These editors include:

- [Menu Editor](#)
- [Pop-up Menu Editor](#)

Menu Editor

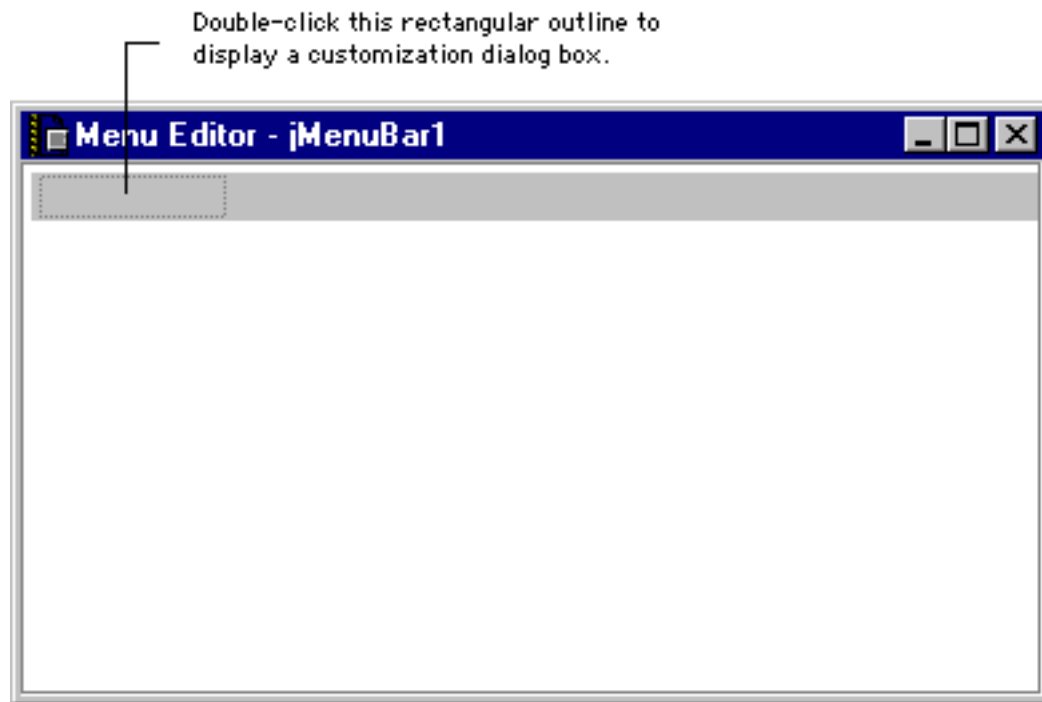
The Menu editor lets you create and modify an application's menu bar and its associated hierarchy of menu commands. This editor displays the menu bar onscreen so that you can see the menus as they appear in the application.

Double-click the representative icon in the layout window to display the Menu editor. Initially, the display is blank, as shown in [Figure 14.1](#). As you build the menu hierarchy, the Menu editor updates the display to reflect your changes.

This section discusses the following topics:

- [Creating menu items](#)
- [Modifying menu items](#)
- [Removing menu items](#)

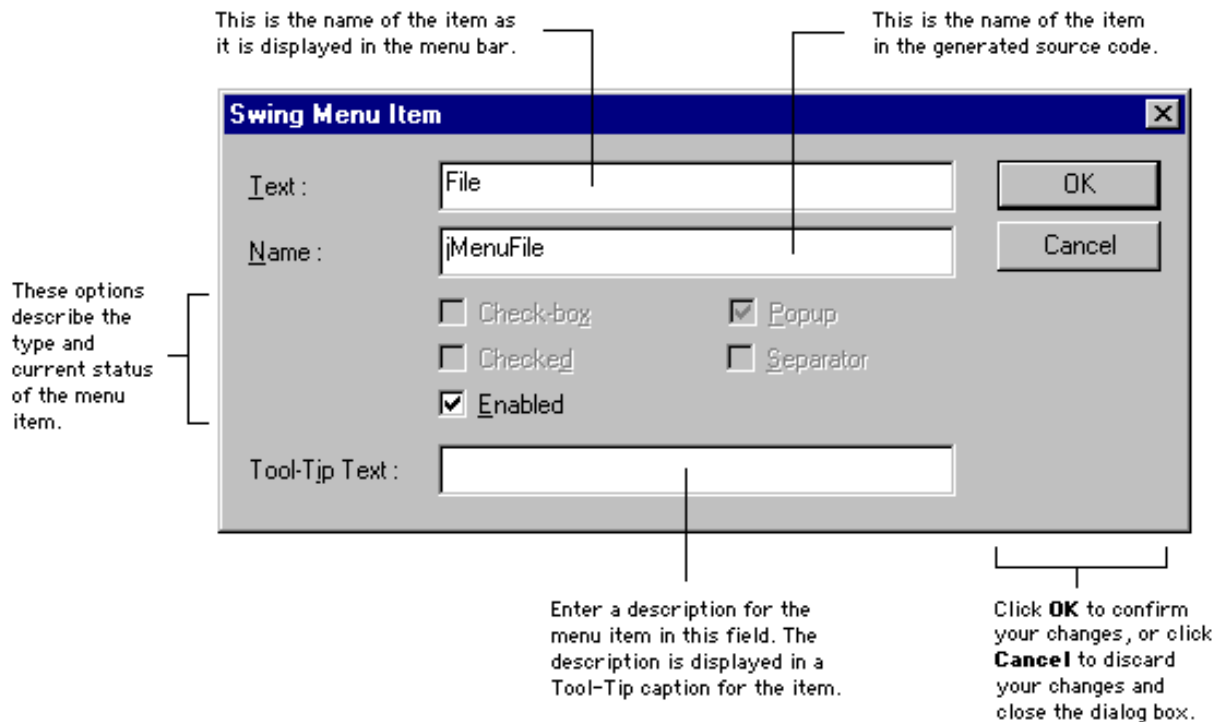
Figure 14.1 **A Menu editor**



Creating menu items

To create a menu item in the application's menu bar, double-click the rectangular outline shown in the Menu editor. A dialog box displays to help you customize a new menu item. The contents of the dialog box depend on the catalog that contains the component. For example, when you use the Menu editor to customize a Java Swing menu bar, the dialog box has the items shown in [Figure 14.2](#).

Figure 14.2 Menu editor dialog box for Java Swing menu items

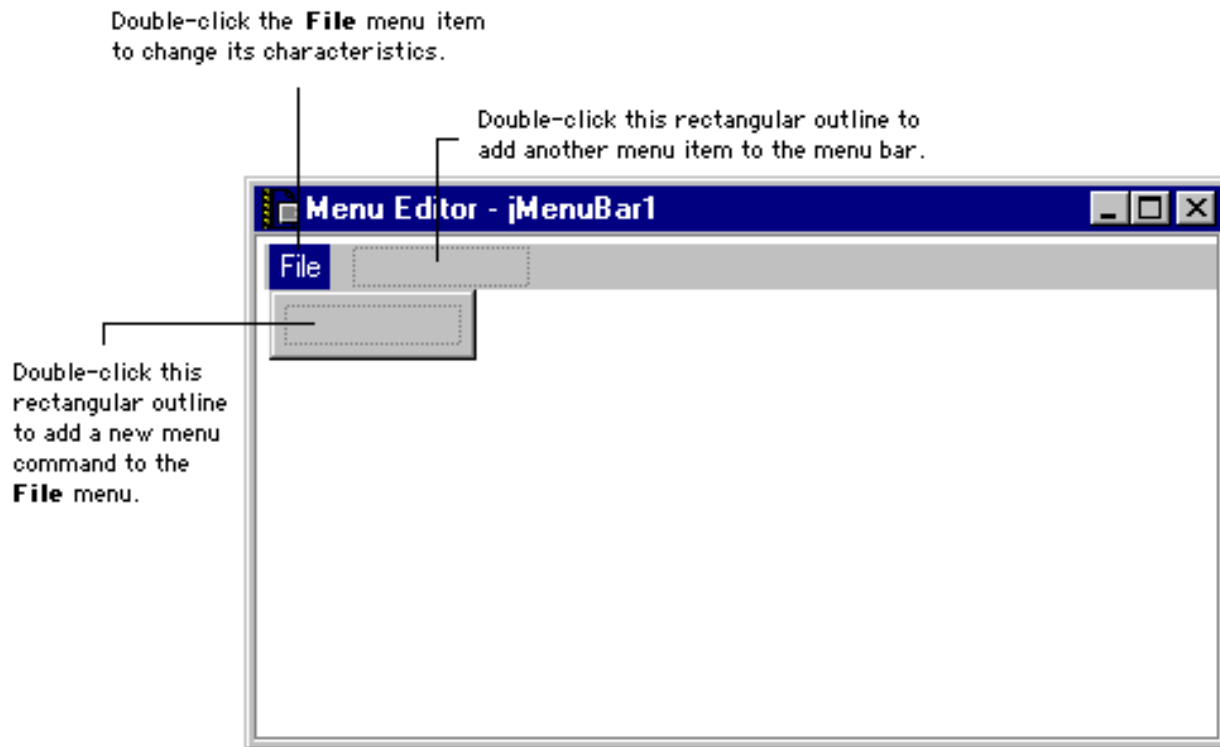


After you enter some information in the dialog box and click **OK**, the Menu editor updates the menu-bar display. For example, if you enter **File** in the **Text** field of the dialog box and click **OK**, the Menu editor display shows the new **File** menu item in the menu bar. The rectangular outline moves to the right of the **File** menu item.

Modifying menu items

You can continue modifying your application's menu bar by double-clicking existing menu items or by clicking rectangular outlines. For example, if you click the **File** menu item shown in [Figure 14.3](#), the Menu editor displays a rectangular outline beneath the **File** menu. This way, you modify the commands that are displayed for each menu item in the menu bar.

Figure 14.3 A Menu editor with modified menu items



When you finish modifying the menu bar, close the Menu editor window. Although the original representative icon appears in the layout window, your changes to the menu bar remain in place. To immediately save changes to the menu bar, choose **Save** from the File menu after closing the Menu editor window.

Removing menu items

To remove a particular menu item, click its name in the Menu editor window and press Delete. The Menu Editor removes the menu item and updates the display. Removing a menu item from the menu bar also removes that menu item's associated commands.

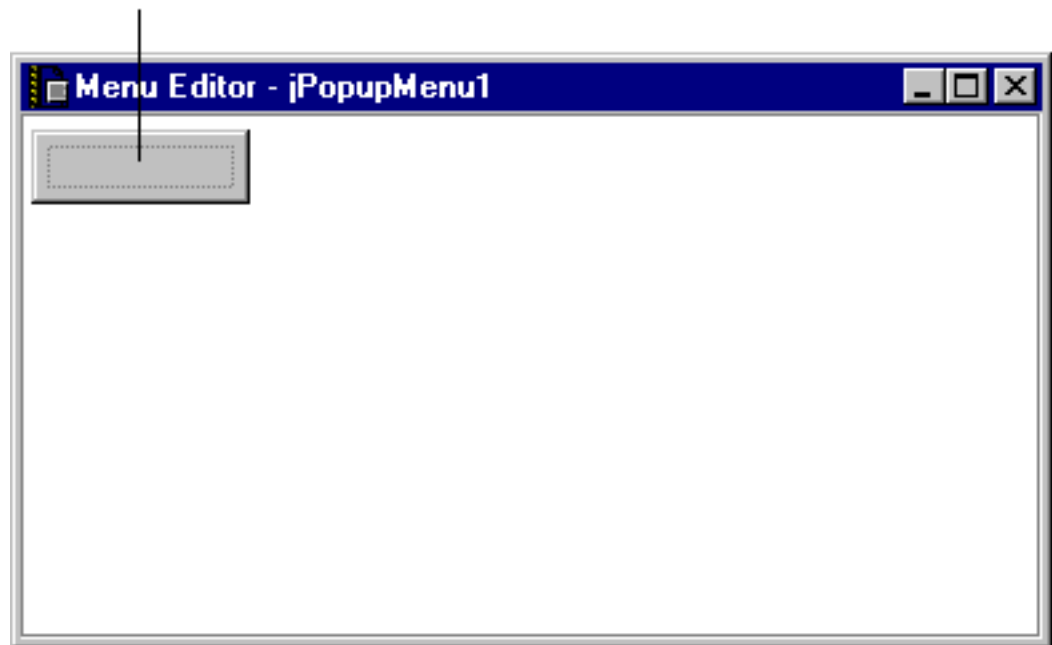
NOTE When removing menu items from the menu bar, you might need to close and then re-open the Menu editor to accurately reflect your changes.

Pop-up Menu Editor

The Pop-up Menu editor is very similar to the Menu editor in its behavior. The only difference between the two editors is that the Pop-up menu editor handles one pop-up menu at a time. [Figure 14.4](#) shows the initial display of a Pop-up menu editor.

Figure 14.4 A Pop-up menu editor

Double-click this rectangular outline to
add menu commands to the pop-up menu.



For more information about creating, modifying, and removing menu items in the Pop-up Menu editor, consult [“Menu Editor” on page 573](#). The same techniques you use in the Menu Editor can be applied to the Pop-up Menu editor.

RAD Browsing



The CodeWarrior browser has additional capabilities for browsing rapid application development (RAD) projects. When you configure your project to generate a browser database, the IDE automatically includes additional RAD-specific information in the database. The browser displays a tab control in the Browser window to help you explore the additional information.

This chapter assumes that you are experienced with the general features of the browser, and that you want to learn how to use the browser's RAD capabilities. See [“Browsing Source Code” on page 207](#) for more information about activating and configuring the browser for your project.

The topics in this chapter include:

- [Browser Window RAD Features](#)
- [RAD Windows](#)

Browser Window RAD Features

When exploring the browser database for RAD projects, the CodeWarrior browser enables additional functionality. This added functionality lets you use the following RAD capabilities:

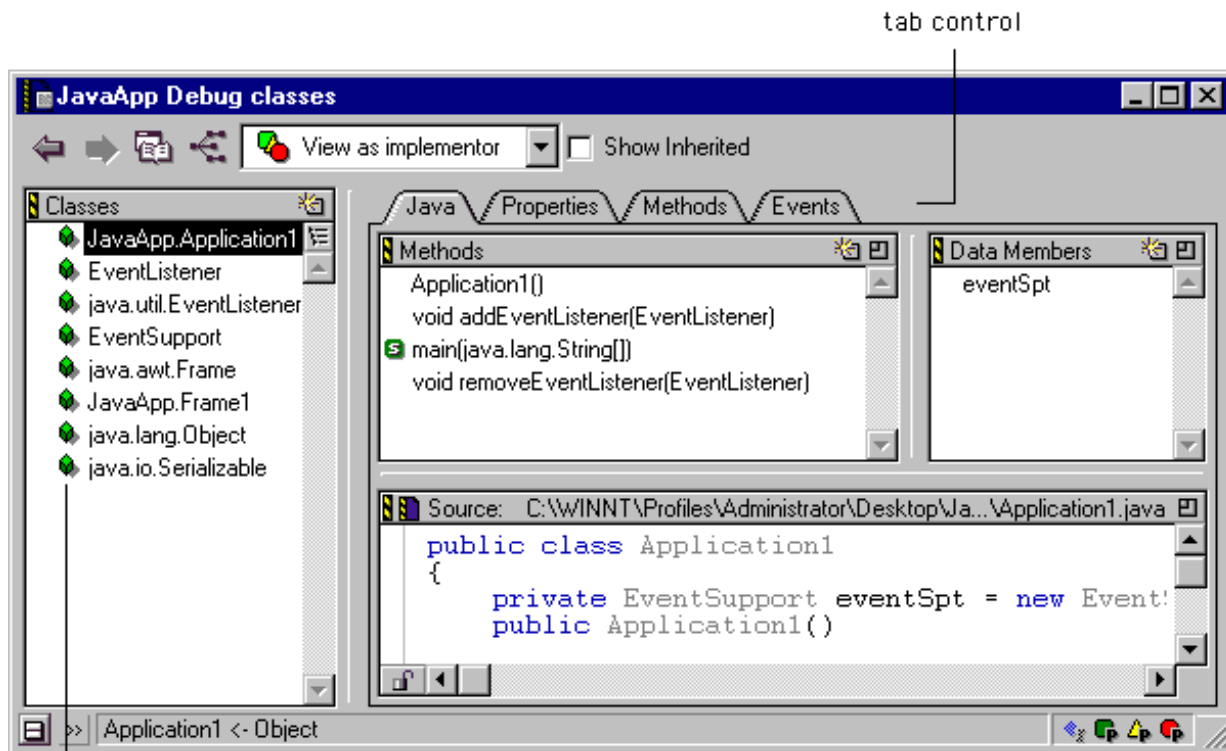
- [Tab Control](#)
- [Properties View](#)
- [Methods View](#)
- [Events View](#)

Tab Control

The tab control is useful for browsing a RAD component's supporting items, such as properties, methods, and events.

The tab control displays one view at a time. Click a tab name to display its associated view. For example, click the **Java** tab in [Figure 15.1](#) to display the Java view. This particular view shows Java RAD information for the selected component in the Classes pane.

Figure 15.1 The Browser window for a Java RAD project



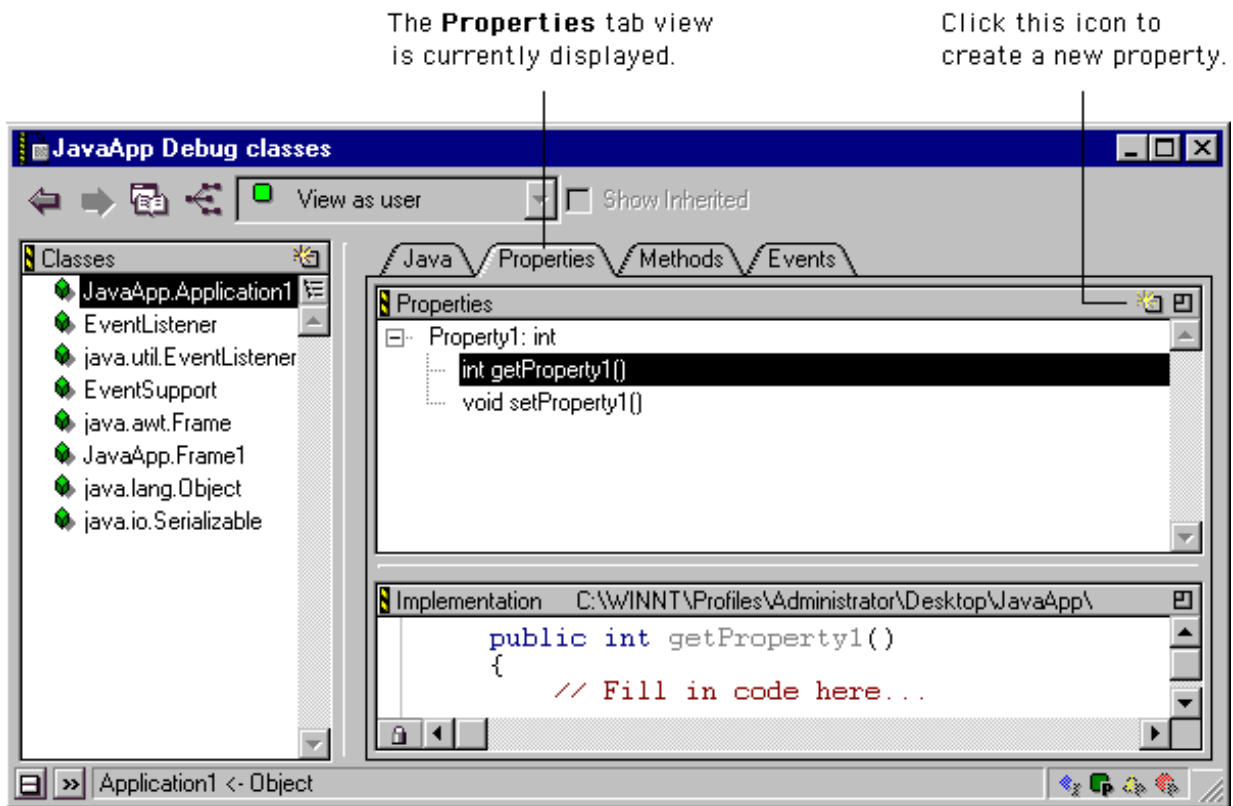
This icon indicates that the item is part of a RAD design.

When you click a **Properties**, **Methods**, or **Events** tab, the corresponding view only displays information for public data not inherited from other member functions. In such views, both the [Browser Access Filters pop-up menu](#) and the **Show Inherited** checkbox are grayed out, indicating that you cannot change these settings. [Figure 15.2](#), [Figure 15.3](#), and [Figure 15.4](#) show the resulting Browser window.

Properties View

The **Properties** view, shown in [Figure 15.2](#), displays information about the properties of the selected component in the Classes pane. This view is divided into the Properties pane and the Implementation pane.

Figure 15.2 Browsing properties



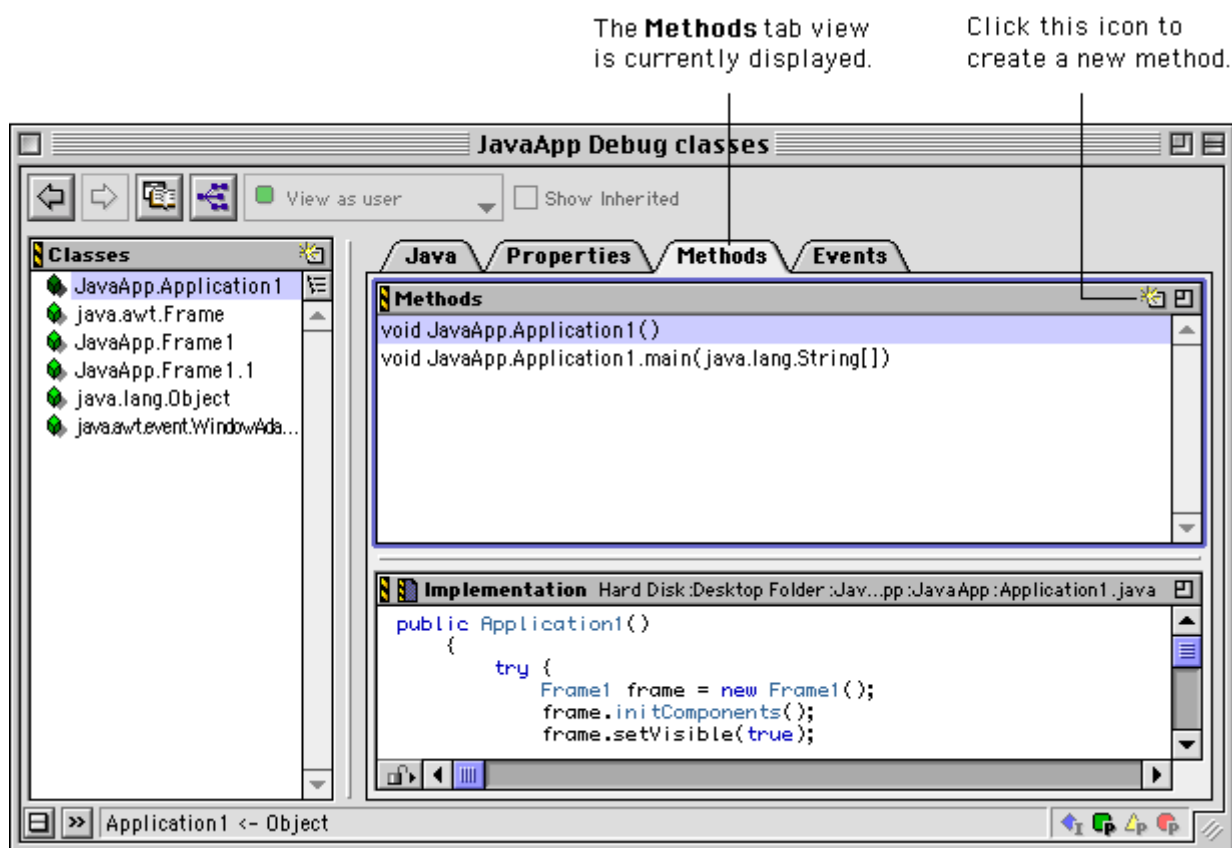
The Properties pane lists the properties for the selected component. When you select a property in this list, the implementation for that property is shown in the Implementation pane. To learn how to create new properties, see [“New Property Window” on page 584](#).

The Implementation pane is similar to the [Source pane](#). This pane displays the selected property’s implementation in your source code. The source code in the pane is fully editable. The top of the pane shows the path to the file containing the implementation.

Methods View

The Methods view, shown in [Figure 15.3](#), displays information about the methods of the selected component in the Classes pane. This tab view is divided into the Methods pane and the Implementation pane.

Figure 15.3 Browsing methods



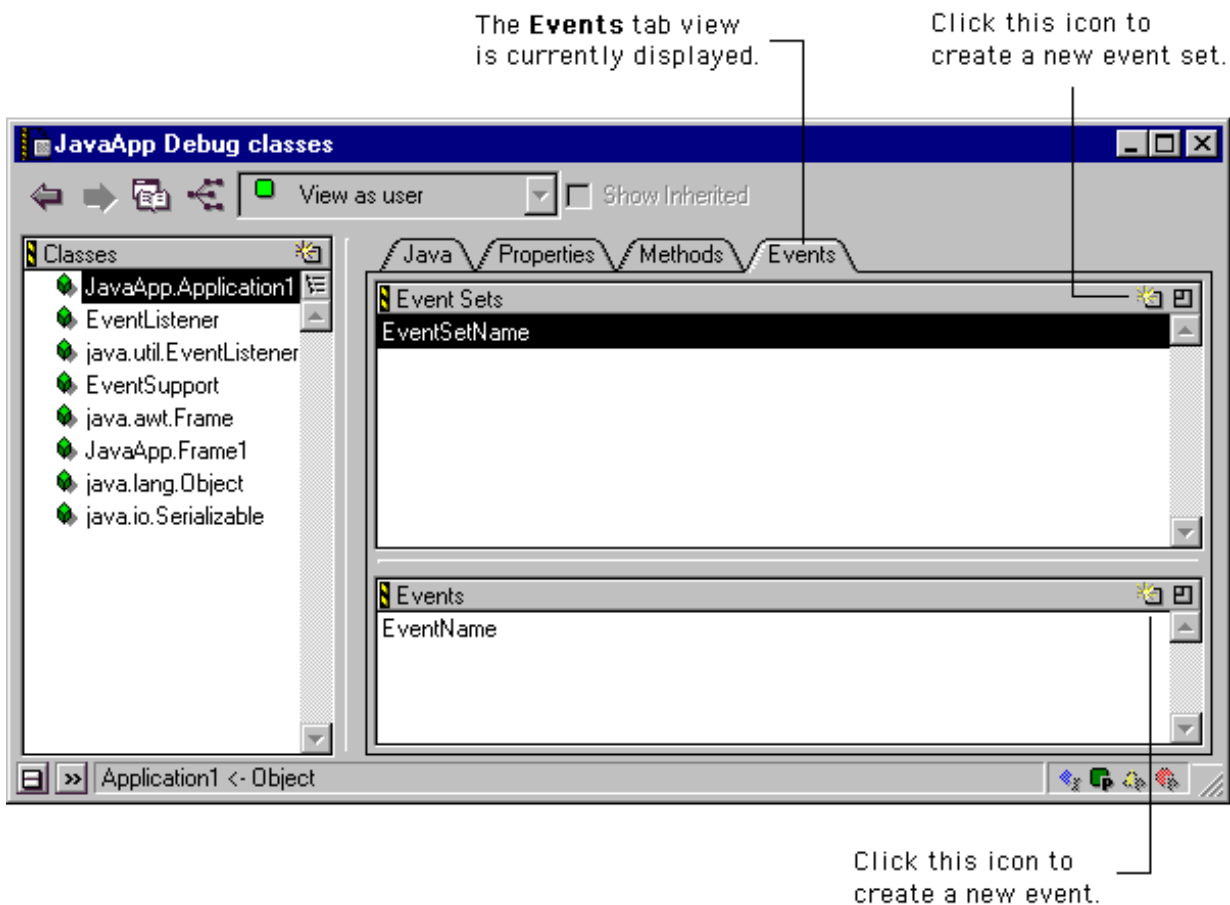
The Methods pane lists the methods for the selected component. When you select a method in this list, the implementation for that method is shown in the Implementation pane. To learn how to create new methods, see [“New Method Window” on page 586](#).

The Implementation pane is similar to the [Source pane](#). This pane displays the selected method’s implementation in your source code. The source code in the pane is fully editable. The top of the pane shows the path to the file containing the implementation.

Events View

The Events view, shown in [Figure 15.4](#), displays information about the events of the selected component in the Classes pane. This tab view is divided into the Event Sets pane and the Events pane.

Figure 15.4 Browsing event sets and events




The Event Sets pane lists groupings of events for the selected component. The Events pane shows the individual events for a selected event set. To learn how to create new event sets, see [“New Event Set Window” on page 589](#). To learn how to create new events, refer to [“New Event Window” on page 597](#).

RAD Windows

When you create a new property, method, event set, or event for a component in your RAD project, the IDE displays a window to assist you. Different options are available for each window type. This section describes the following windows:

- [New Property Window](#)
- [New Method Window](#)
- [New Event Set Window](#)
- [New Event Window](#)

New Property Window

The New Property window helps you create a new property for a selected component in the Browser window. To display this window, make sure that the Browser window is frontmost, then choose [New Property](#) from the Browser menu. Alternatively, click the new item icon () as shown in [Figure 15.2 on page 581](#). The **New Property** command changes slightly, based on the framework you use to develop your code. For example, when developing a Java RAD project, the command changes to **New Bean Property**.

The New Property window is shown in [Figure 15.5](#). This window consists of the following parts:

- [Name](#)
- [Type](#)
- [Package required for type](#)
- [Accessors](#)
- [Has Data Member](#)
- [Summary](#)
- [Add and Cancel buttons](#)

Name

Enter a name for the new property in this field. The name you enter must comply with naming conventions in the framework programming language. For example, you cannot use spaces in names for some languages.

Figure 15.5 New Property window

New Java Property

Adding property to class:

Name:

Type:

Package required for type (optional):

Accessors: ☒ Getter ☒ Setter

☒ Has Data Member:

Initializer:

☐ Transient ☐ Volatile

Summary:

Getter: getProperty1
Setter: setProperty1
Data Member: public void DataMbr1 = "text";

Type

Enter an appropriate property type in this field. The type you specify must be a valid property type in the framework language.

Package required for type

If you wish, you can enter in this field a required package for the property type.

Accessors

These checkboxes determine the accessor characteristics for the property.

Has Data Member

Enable this checkbox if you want the property to have a data member. If you enable the checkbox, the **Name** and **Initializer** fields become available, as well as the **Transient** and **Volatile** checkboxes. Enter the data member name in the **Name** field. You can supply the remaining information as desired.


Summary

This section shows you a synopsis of the information currently specified in the window. As you change a particular value or checkbox, the **Summary** field updates to reflect your changes. This field displays information only after you complete the **Name** and **Type** fields and enable at least one of the **Accessors**.

Add and Cancel buttons

Click **Add** to confirm the current window information and add the new property, or click **Cancel** to discard your changes.

New Method Window

The New Method window helps you create a new method for a selected component in the Browser window. To display this window, make sure that the Browser window is frontmost, then choose [New Method](#) from the Browser menu. Alternatively, click the new item icon () as shown in [Figure 15.3 on page 582](#). The **New Method** command changes slightly, based on the framework you use to develop your code. For example, when developing a Java RAD project, the command changes to **New Bean Method**.

The New Method window is shown in [Figure 15.6](#). This window consists of the following parts:

- [Name](#)
- [Return type](#)
- [Parameters](#)
- [Optional throws](#)
- [Package required for parameters](#)
- [Modifiers](#)
- [Declaration](#)
- [Add and Cancel buttons](#)

Name

Enter a name for the new method in this field. The name you enter must comply with naming conventions in the framework programming language. For example, you cannot use spaces in names for some languages.

Return type

Enter an appropriate method return type in this field. The type you specify must be a valid return type in the framework language.

Parameters

If desired, enter method parameters in this field. Some example parameters are listed above this field.

Optional throws

If desired, enter exception-handling information (*throws*) in this field. Some example throws are listed above this field.

Package required for parameters

If you wish, you can enter in this field a required package for the parameter. Some example packages are listed above this field.

Figure 15.6 **New Method window**

New Java Method

Adding method to class:

Name:

Return type:

Parameters: Example: int a, java.util.Vector b, ...

Optional throws: Example: java.io.IOException, YourPkg.YourExcpt, ...

Package required for parameters (optional): Ex: java.util.*, ...

Modifiers:

Access: Specifier:

☐ Native ☐ Synchronized

Declaration:

Modifiers

Use the **Access** and **Specifier** pop-up menus to choose the access level and method specifier for the new method. Possible access levels include **Public**, **Protected**, and **Private**. Possible specifiers

include **None**, **Abstract**, **Final**, and **Static**. You can enable the **Native** or **Synchronized** checkboxes as desired to further describe the method's modifiers.


Declaration

This section shows you the current form of the declaration as specified in the window. After you change a particular value or checkbox, the **Declaration** field updates to reflect your changes. This field displays information only after you complete the **Name** and **Type** fields and specify an **Access** level.

Add and Cancel buttons

Click **Add** to confirm the current window information and add the new method, or click **Cancel** to discard your changes.

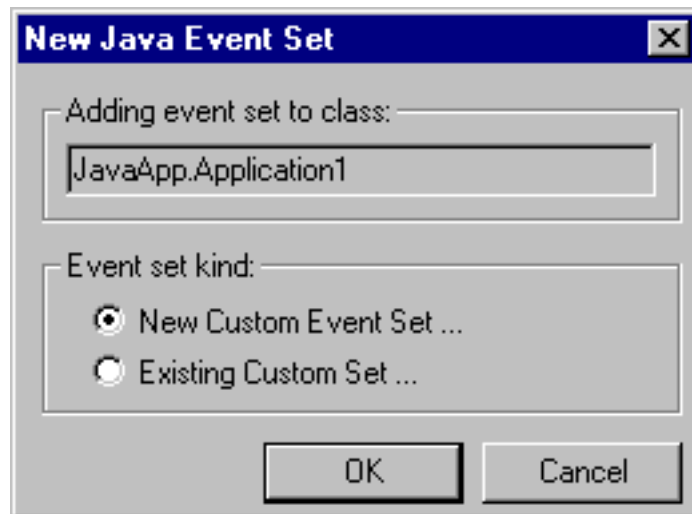
New Event Set Window

The New Event Set window helps you create a new event set for a selected component in the Browser window. To display this window, make sure that the Browser window is frontmost, then choose [New Event Set](#) from the Browser menu. Alternatively, click the new item icon () as shown in [Figure 15.4 on page 583](#). The **New Event Set** command changes slightly, based on the framework you use to develop your code. For example, when developing a Java RAD project, the command changes to **New Bean Event Set**. The New Event Set window is shown in [Figure 15.7](#).

This window lets you specify the type of event set you want to create. There are two possibilities:

- [New Custom Event Set](#)
- [Existing Custom Set](#)

Figure 15.7 **New Event Set window**

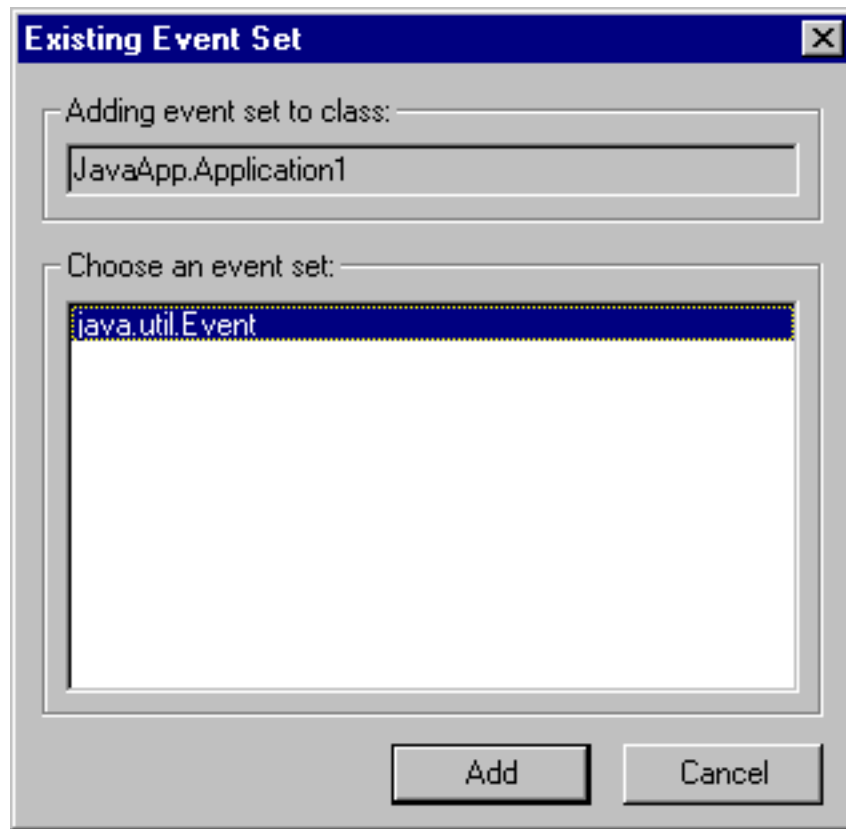


New Custom Event Set

When you enable this option and click **OK**, a wizard opens to help you create a custom event set. See [“New Event Set wizard” on page 591](#) for more information.

Existing Custom Set

When you enable this option and click **OK**, the Existing Event Set window, shown in [Figure 15.8](#), displays. Select from the list the event set you wish to add and click **Add**. This existing event set is then added to the selected component in the Browser window.

Figure 15.8 Adding an existing event set

New Event Set wizard

The [New Event Set Window](#) contains an option to create a [New Custom Event Set](#). When you enable this option and click **OK**, the New Event Set wizard displays.

The wizard includes the following navigation buttons:

- **Back**—return to the previous section
- **Next**—proceed to the subsequent section
- **Finish**—display a summary of current information
- **Cancel**—discard all changes

The New Event Set wizard is divided into the following steps:

1. [Choose a name and location for the new event set.](#)
2. [Specify a base class and a list of implementations.](#)

3. [Assign the new event set to the project's build targets.](#)

You progress through these steps in sequence. Each step builds on the information provided in previous steps. When you supply enough information in a particular step, click **Next** to continue.

To accept all current settings in the wizard and quickly create a new event set, click **Finish**. The wizard displays a summary of all the current settings for the new event set. You can always click **Cancel** while viewing the summary to continue modifying settings.

To use the New Event Set wizard, follow these steps:

1. **Choose a name and location for the new event set.**

This section of the wizard, shown in [Figure 15.9](#), lets you specify the name and location of the new event set, as well as the modifiers for the set.

This section includes the following parts:

- [Class Name](#)
- [File](#)
- [Package](#)
- [Modifiers](#)

Class Name

Enter a name for the event set's class in this field. An example is provided below the field. The **Class is a Bean** checkbox below the field remains disabled. This status indicates that you are not creating a Java bean.

File

This pop-up menu lets you specify the type of event set:

- **New File**—an event set file
- **Relative to class**—an event set that depends on an existing class in the project

Figure 15.9 New Event Set wizard - Name and Location

New Java Event Set

Name and Location

Class Name:

☐ Class is a Bean Example: <EventSetName> Listener

File:

class

Package:

Modifiers

Access: Specifier:

Depending on the option you choose, different fields become enabled below the **File** pop-up menu. The enabled fields for the **New File** option are shown in [Figure 15.9](#), and the enabled fields for the **Relative to class** option are shown in [Figure 15.10](#).

If you choose the **New File** option, type in the enabled field the path to which you want to save the file. Alternatively, click **Set** next to the field. A standard Save window opens. Use the window controls to select the location to which you want to save the file and click **Save**.

Figure 15.10 Choosing the Relative to class option

New Java Event Set

Name and Location

Class Name:

☐ Class is a Bean Example: <EventSetName> Listener

File: **Relative to class** ▼

class

Package:

Modifiers

Access: Specifier:

If you choose the **Relative to class** option, the **class** field and the pop-up menu beside it become enabled, as shown in [Figure 15.10](#). In the **class** field, type the name of the class you want to relate to the event set. Alternatively, click **Set** next to the **class** field, select a class in the window that opens, and then click **Select**. Next, use the pop-up menu to place the event set event set **Before**, **After**, or **Inside** the specified class.

Package

If you wish, you can enter in this field a package for the event set. This field is not available if you choose the **Relative to class** option from the [File](#) pop-up menu.

Modifiers

Use the **Access** and **Specifier** pop-up menus to choose the access level and method specifier for the new event set. Possible access levels include **Public**, **Protected**, and **Private**. Possible specifiers include **None**, **Abstract**, **Final**, and **Static**.

2. Specify a base class and a list of implementations.

This section of the wizard, shown in [Figure 15.11](#), lets you specify the base class and implementations for the new event set.

Figure 15.11 New Event Set wizard - Base Class and Interfaces

New Java Event Set

Base Class

☐ Visual Component

java.awt.Component

☒ Custom

java.util.EventListener

Implements List

Example: java.util.EventListener, java.awt.MouseListener, YourOwnInterface, ...

< Back Next > Finish Cancel

This section includes the following parts:

- [Base Class](#)
- [Implements List](#)

Base Class

The settings in this part determine the base-class type of the new event set. There are two possibilities:

- **Visual Component**—Enable this option if the base class is a visual component. The pop-up menu below this option lists classes recognized by the installed RAD plug-ins for the IDE. Choose a component from this pop-up menu.
- **Custom**—Enable this option if the base class of the event set is a non-visual component or a custom class. Type the name of the base class in the field below this option. A default name is provided.

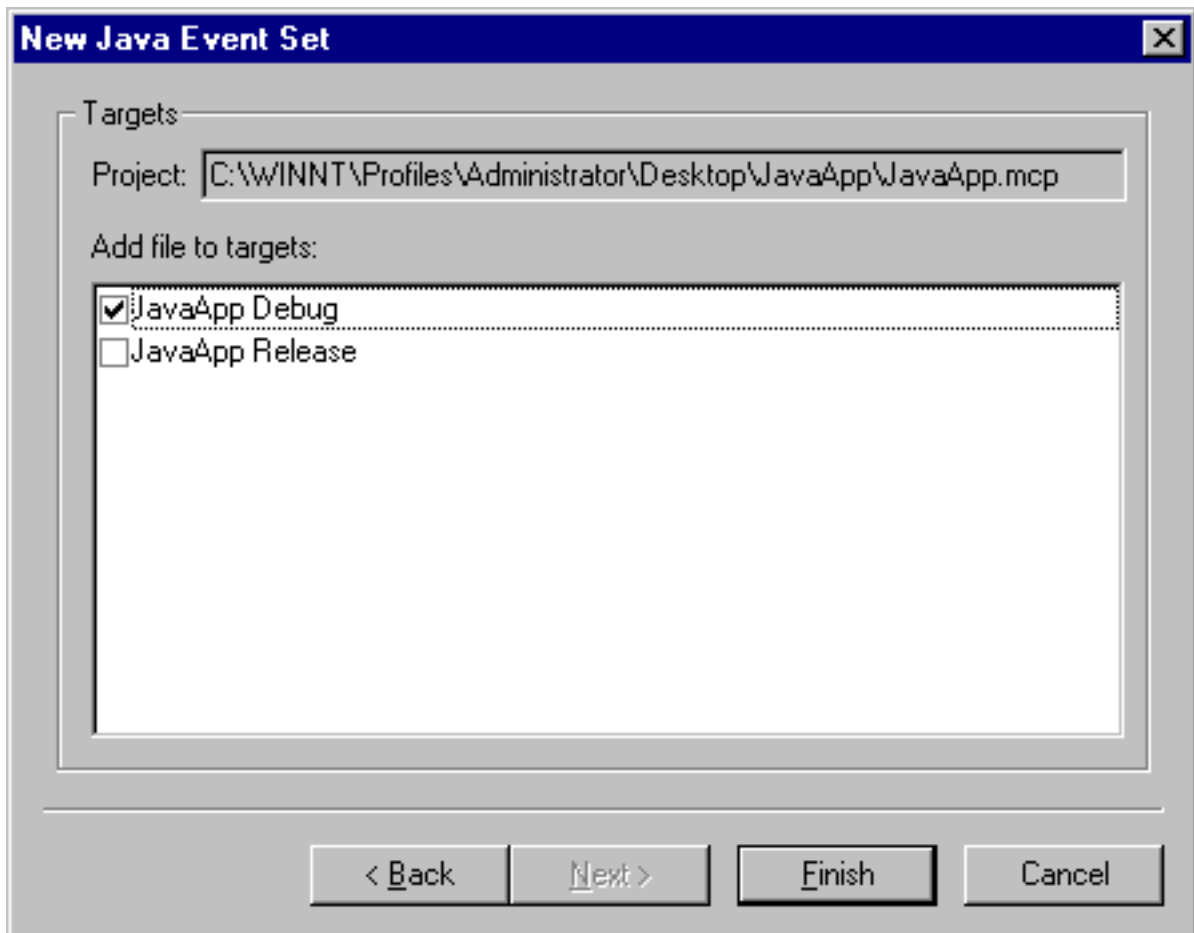
Implements List

In this part, type a list of interfaces to be implemented by the base class. Example interfaces are provided. Separate multiple interfaces with commas.

3. Assign the new event set to the project's build targets.

This section of the wizard, shown in [Figure 15.12](#), lets you assign the event set to particular build targets within the active project.

To assign the event set to a specific build target, enable the checkbox next to the build target's name in the **Add file to targets** list. For example, in [Figure 15.12](#), the new event set is assigned to the **JavaApp Debug** build target, and it is not assigned to **JavaApp Release**.

Figure 15.12 New Event Set wizard - Targets

New Event Window

The New Event window helps you create a new event for a selected event set in the Browser window. To display this window, make sure that the Browser window is frontmost and that an event set is selected, then choose **New Event** from the Browser menu.

Alternatively, click the icon described in the Events pane, shown in [Figure 15.4 on page 583](#). The **New Event** command changes slightly, based on the framework you use to develop your code. For example, when developing a Java RAD project, the command changes to **New Bean Event**.

Figure 15.13 **New Event window**

New Java Event

Adding event to class:
EventSetNameListener

Name:
Event1

Parameters: usually: <EventObjectType> evt

Optional throws:

Package required for parameters (optional):

Modifiers
☐ Synchronized

Declaration:
public abstract void Event1();

Add Cancel

The New Event Set window is shown in [Figure 15.13](#). This window consists of the following parts:

- [Name](#)
- [Parameters](#)
- [Optional throws](#)
- [Package required for parameters](#)

- [Modifiers](#)
- [Declaration](#)
- [Add and Cancel buttons](#)

Name

Enter a name for the new method in this field. The name you enter must comply with naming conventions in the framework programming language. For example, you cannot use spaces in names for some languages.

Parameters

If desired, enter event parameters in this field. A sample construct for a parameter is listed above this field.

Optional throws

If desired, enter exception-handling information (*throws*) in this field.

Package required for parameters

If you wish, you can enter in this field a required package for the event.

Modifiers

Enable the **Synchronized** checkbox to modify the event accordingly.

Declaration

This section shows you the current form of the declaration as specified in the window. After you change a particular value or checkbox, the **Declaration** field updates to reflect your changes. This field displays information only after you complete the **Name** field.

Add and Cancel buttons

Click **Add** to confirm the current window information and add the new method, or click **Cancel** to discard your changes.

Using the CodeWarrior IDE with Version Control Systems



This chapter explains how to use the CodeWarrior IDE version control integration facilities to control your source code. Topics include:

- [Using Version Control Systems](#)
- [Activating VCS Operations](#)
- [Accessing VCS Features](#)
- [Common VCS Operations](#)

Using Version Control Systems

A revision control or version control system (VCS) allows you to maintain a database of your source code, and then check files in or out of the database. This makes it easy to track changes to your code, particularly when more than one person is working on a software project.

Commercially Available VCS Plug-ins

The CodeWarrior IDE plug-in architecture supports a variety of version control systems. [Table 16.1](#) describes some of the major version control plug-ins available from Metrowerks and third parties.

Table 16.1 **Version control plug-ins**

Product	Vendor	Windows	Mac OS	Solaris/ Linux
ClearCase	Rational	1		1
	Metrowerks	3		3
CVS	Electric Fish		✓	
	Metrowerks	✓	2	2
Perforce	Perforce	1	✓	1
Projector	Apple Computer		1	
	Electric Fish		3	
	Metrowerks		3	
PVCS	Intersolv	1		1
	Synergex		✓	
SourceSafe	Metrowerks	3	✓	
	Microsoft	✓		
	Mainsoft			1
VOODOO	UNI Software		✓	

LEGEND: ✓ IDE support

1 Platform support only

2 Under development

3 Requires VCS vendor's tools be pre-installed

A blank field indicates an unsupported plug-in.

For the latest plug-in tool list, visit the Metrowerks website at the following URL:

http://www.metrowerks.com/desktop.qry?x=Version_Control

Activating VCS Operations

Before you can use any version control system with the CodeWarrior IDE, it must be properly installed and activated.

Installing VCS Software

Installing most revision control system software is as simple as copying the VCS plug-in to the appropriate folder for your platform and restarting the CodeWarrior IDE. This is usually all it takes to make the VCS software available for use.

NOTE Be sure to follow the installation instructions that accompany the VCS software you plan to use. If you encounter any problems, contact the VCS software vendor for assistance.

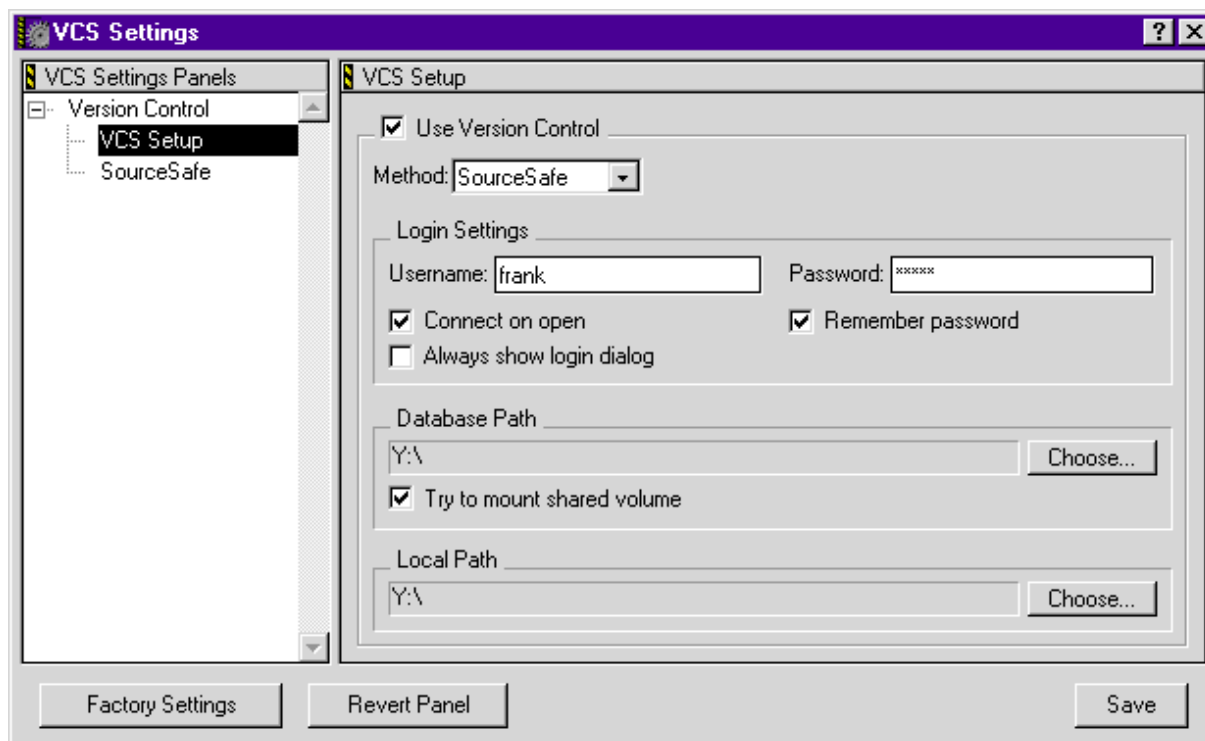
VCS plug-ins normally reside in the Version Control folder. Use the following path descriptions to help you locate where your VCS plug-in should be copied to:

Windows	{Compiler}\Plugins\Version Control
Mac OS	{Compiler}:CodeWarrior Plugins:Version Control
Solaris and Linux	{Compiler}/Plugins/Version_Control

Activating VCS Software

The **VCS Setup** panel, as shown in [Figure 16.1](#), is present if you have installed recognized revision control software. You use the **VCS Setup** panel to specify client options for connecting to a revision control systems.

Figure 16.1 The VCS settings panel



Setting up a CodeWarrior project for version control is simple. You just need to enter some information in the **VCS Setup** panel while a project is open. You'll need to input the information separately for each project that uses revision control.

NOTE Be sure to follow the activation instructions that accompany the VCS software you plan to use. If you encounter any problems, contact the VCS software vendor for assistance.

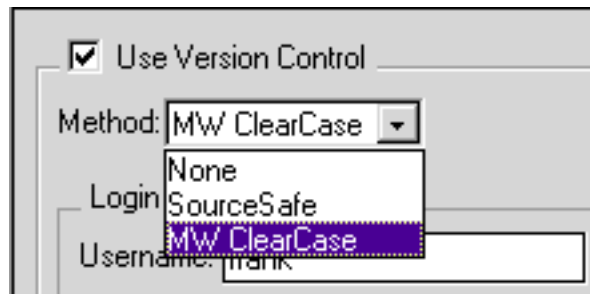
To Activate Your VCS Settings:

1. **Choose Version Control Settings from the Edit menu.**
The VCS Settings window appears ([Figure 16.1](#)).
2. **Enable the Use Version Control option.**
Click the **Use Version Control** option to activate version control.

3. **Choose a version control system from the Method menu.**

The CodeWarrior IDE is capable of supporting several different types of version control. If your VCS software has been correctly installed, its name should be present on the **Method** list as shown in [Figure 16.2](#).

Figure 16.2 Method pop-up list.



4. **Enter your username.**

Enter your user name in the **Username** field.

5. **Enter your password (optional).**

Enable the **Remember Password** option, then enter your password in the **Password** field.

6. **Set additional Login Settings options.**

To customize your development interaction with the chosen VCS setting you can enable any or all of the following options:

- Connect on Open
- Always Show Login Dialog
- Remember Password (required to save password)

7. **Choose the Database Path.**

Click **Choose** to select the VCS database that you want to access. The actual setting of the Database Path field is dependent upon which VCS software you are using. See the VCS software documentation for more information.

8. **Choose the Local Path.**

Click **Choose** to select the destination folder where your local files will be stored. The actual setting of the Local Path field is dependent

upon which VCS software you are using. See the VCS software documentation for more information.

9. **Click Save.**

Save your changes using the **Save** button.

If all your data was correct, you should be connected to the VCS database set in the Database Path field. If you aren't, review the above steps, ensuring that the data entered is correct.

NOTE Some VCS software may have additional setup requirements. If you encounter any problems, contact the VCS software vendor for assistance.


Accessing VCS Features

The CodeWarrior IDE provides several ways to access common VCS operations and view status and log information. These include:

- [VCS Menu](#)
- [VCS Login dialog](#)
- [VCS Pop-up](#)
- [Project Window](#)
- [VCS Messages Window](#)

VCS Menu

Once version control is activated for a project, a **VCS** menu appears as shown in [Figure 16.3](#). You select commands from the **VCS** menu to perform **Get**, **Checkout**, **Checkin**, and other common operations available from the specified version control system.

Macintosh On the Macintosh, the **VCS** menu appears on the menubar as an icon that looks like this: .

NOTE The VCS menu appearance may differ depending upon which VCS operations are available. See the documentation that came with the VCS software for more information.

Figure 16.3 VCS menus for Windows and Mac OS

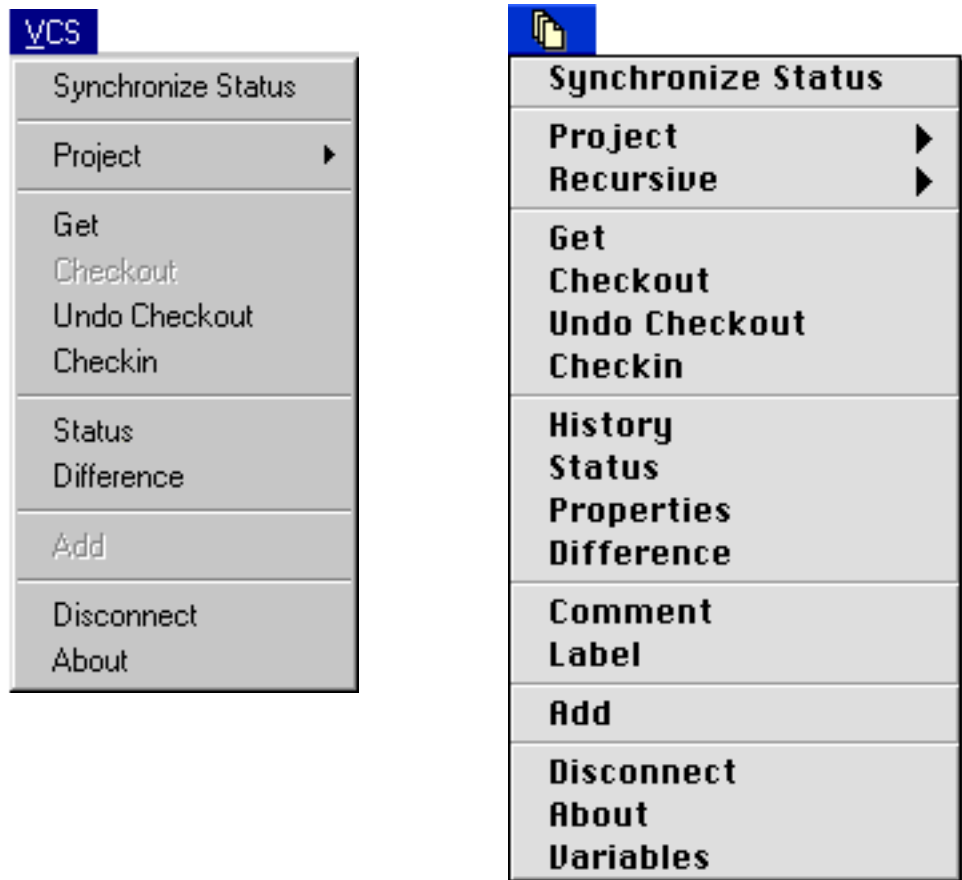


Table 16.2 Supported VCS Operations

Command	Description
Synchronize Status	Updates the VCS Status column in the Project window by examining each project file's status and updating its information.
Project	Contains a submenu of VCS commands that enable you to perform Get, Checkout, Undo Checkout, Checkin, Status, and Add operations on project files themselves.
Recursive	Contains a submenu of VCS commands that enable you to perform recursive operations.
Get	Retrieves a copy of the file without checking it out of the project database.

Command	Description
Checkout	Checkout files for exclusive modification.
Undo Checkout	Cancels a checkout, voiding all changes.
Checkin	Returns a modified file to the database and relinquishes the checkout.
History	Displays a modification history of a project or file.
Status	Displays a file's checkout status.
Properties	Displays database information about a project or file.
Comment	Changes a comment for a specific version of a project or file.
Label	Assigns a label to a project or file.
Add	Adds a file to the database.
Connect or Disconnect	Connects or disconnects you from the project database depending upon the current open status.
About	Displays VCS plug-in copyright and version information.
Variables	Displays the VCS user's variable settings in an editor window.

NOTE Each VCS plug-in may assign different names to these operations. See the documentation that accompanied your VCS software to determine which operations are supported.

Figure 16.4 VCS Login dialog



Version Control Login Window

The **Version Control Login** window ([Figure 16.4](#)) appears when you open a project that uses a revision control system and the **Always Show Login Dialog** option has been enabled. Enter your username and password, then click **OK** to continue opening the project. Otherwise, click **Cancel** to stop.

VCS Pop-up


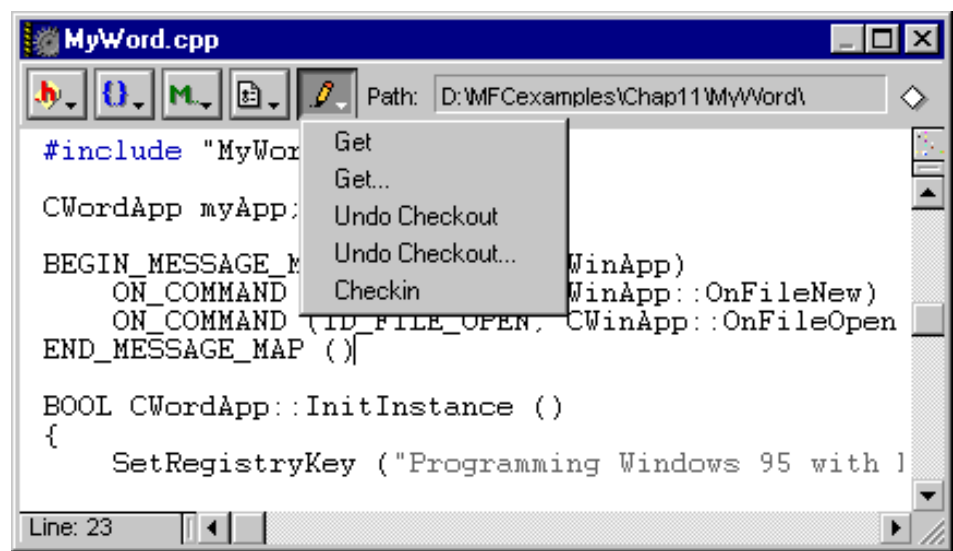
The **VCS Pop-up** menu () ([Figure 16.5](#)) appears in editor windows when a version control system is activated. The icon that represents the **VCS Pop-up** menu indicates the current permission setting for the file. For more information on permission settings, see ["VCS States" on page 610](#).




Figure 16.5 VCS Pop-Up Menu





VCS States

When using a version control system, each file in a project can have a different permission setting. A file’s permission setting can be viewed in the Project window’s Checkout Status Column (Figure 16.6) and as part of the file’s editor window’s **VCS Pop-up** menu (Figure 16.6). See Table 16.3 for a description of each state indicator.

Table 16.3 VCS state Icons

If the icon is...		Then...
	Checked out	You can edit the file and add your changes to the revision control database.
	Checked in	You cannot edit the file, and it is part of the revision control database.
	Writable	You can edit the file, but you cannot add your changes to the revision control database because the file was not properly checked-out for modification.

If the icon is...		Then...
	Unlocked	The file can be edited, and it is not checked into a revision control database.
	Locked	You cannot edit the file, and it is <i>not</i> part of the revision control database. You may not have the access privileges needed to access the file, or someone may have locked the file.

Depending on the state of the **VCS Pop-Up** menu when you click on it, certain operations may be performed on the file you are editing. These supported operations are described in [Table 16.4](#).

You choose an item from the **VCS Pop-up** menu to execute a VCS operation. If the menu item contains an ellipsis character "...", a dialog box of advanced options for the operation appears, enabling you to customize the operation before executing it.

Mac OS Press the **Option** key before choosing a menu command to access any advanced VCS options. If an ellipsis character "..." appears next to a menu command, advanced VCS options are available for that operation.

For more information on these operations, refer to the documentation that came with your revision control system.

Table 16.4 Supported VCS Pop-up Operations

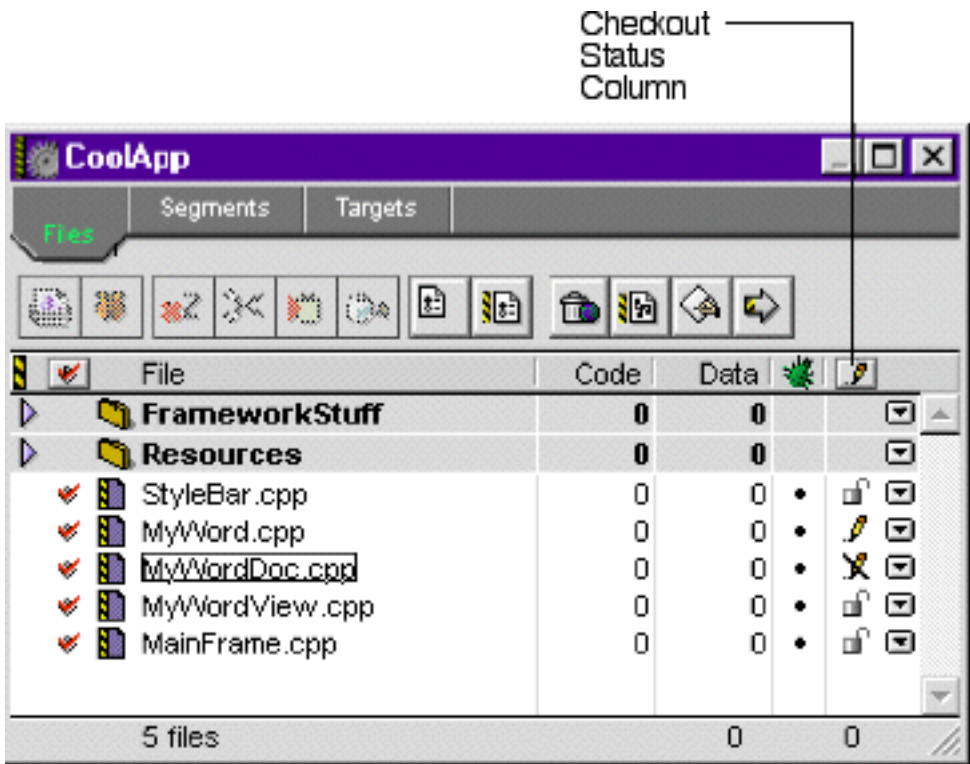
Command	Description
Unlock	Change the lock on the file (if possible) so that the file is writable.
Add	Add the file to the revision control database.
Get	Retrieve a fresh copy of the file from the revision control database.
Checkout	Check the file out from the revision control database for modifications.
Undo Checkout	Discard any changes made to the file, and tell the revision control database to cancel the checkout of the file.

Command	Description
Checkin	Tell the revision control database to accept the file with the changes that have been made to it.
Make Writable	Make the file writable for experimentation, though it will not be able to be checked into the revision control database.

Project Window

When version control is activated for a project, the **Checkout Status** column appears in the Project window (Figure 16.6). This column uses various icons to represent the current checkout status, or file permission, of the project files. Whenever you change the checkout status of a file in the CodeWarrior IDE, this column updates to reflect the change.

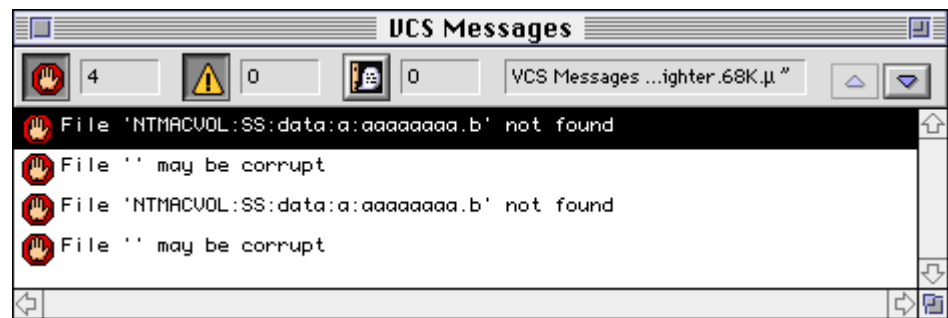
Figure 16.6 Checkout Status Column



VCS Messages Window

The CodeWarrior IDE uses a message window to display a log of revision control messages, as shown in [Figure 16.7](#). To learn how to use the controls in the window, refer to [“Using the Message Window” on page 382](#).

Figure 16.7 VCS Messages window



Common VCS Operations

In addition to the standard revision control operations like **Add**, **Get**, **Checkout**, **Checkin**, and others, various other operations are possible. These include:

- [Getting a File's Status](#)
- [Modifying a Checked In File](#)
- [Dealing with 'ckid' Resources \(Mac OS\)](#)
- [Other Operations](#)

Getting a File's Status

If you have configured your project file to use a version control system, and have checked in your files in accordance with the procedures described in the VCS documentation, you will see one of the icons of [Table 16.3](#) in the Checkout Status column of the Project window for every file under revision control in your project. Refer to [Figure 16.6](#) to see where the Checkout Status column is located in the Project window.

Clicking the **Checkout Status** column icon at the top of the column will perform a **Synchronize Status** command on the project's files against the revision control database. The status of all files you have on your hard disk will be compared and synchronized with the status of the files in the revision control database.

In addition, the **VCS Pop-up** menu in the file's editor window will always indicate the current checkout status of the file.

Modifying a Checked In File

You can change the status of a Checked in source code control file so you can modify it, but after modification, you will not be able to check it back into the source code control database. This feature is provided to allow you to experiment with a file by temporarily making it writable. If the revision control database allowed you to override permissions on files without consequence, it would not be providing any valuable control services for you.

Windows	Right-click on the file in Explorer, then click Properties . In the window see if the Locked attribute is checked. If you uncheck it, the file can then be edited.
Mac OS	Choose Make Writable from the VCS Pop-up menu in the editor window (Figure 16.8). You can now modify the file, but you won't be able to check it back into its source code control database.

Figure 16.8 Changing Read-Only to Modify Read-Only



Dealing with 'ckid' Resources (Mac OS)

The CodeWarrior IDE *almost* always respects the presence of 'ckid' resources in text files. The 'ckid' resource was created for use by *Projector* and is used by most Macintosh VCS's.

The one time that the CodeWarrior IDE may do something different than indicated by a 'ckid' resource is when dealing with a project file that contains a 'ckid' resource.

Other Operations

Other operations may be available depending upon which revision control system plug-in you use. You will need to refer to the documentation that accompanied the VCS software you are using in order to determine this.

CodeWarrior Reference



This chapter describes each menu command in the CodeWarrior IDE and the default key bindings for those commands. You can use this chapter as a convenient reference when you want to find information quickly.

The following information is discussed:

- [IDE Menu Reference](#)
- [IDE Default Key Bindings](#)

IDE Menu Reference

This section provides an overview of the menu commands in the CodeWarrior IDE.

There are several menus in the IDE's menu bar:

- [File Menu](#)
- [Edit Menu](#)
- [Search Menu](#)
- [Project Menu](#)
- [Debug Menu](#)
- [Data Menu](#)
- [Browser Menu](#)
- [Window Menu](#)
- [Layout Menu](#)
- [Version Control System \(VCS\) Menu](#)
- [Help Menu](#)
- [Java Exceptions Submenu](#)
- [Toolbar Submenu](#)

- [Apple Menu \(Mac OS\)](#)
- [Tools Menu \(Mac OS\)](#)
- [Scripts Menu \(Mac OS\)](#)
- [Editor Extensions Menu \(Mac OS\)](#)
- [Info Menu \(Solaris\)](#)

Windows In Windows, the [File Menu](#), [Edit Menu](#), [Search Menu](#), [Project Menu](#), [Debug Menu](#), [Window Menu](#), and [Help Menu](#) are visible at all times.

Mac OS In the Mac OS, the [Apple Menu \(Mac OS\)](#), [File Menu](#), [Edit Menu](#), [Search Menu](#), [Project Menu](#), [Debug Menu](#), [Window Menu](#), and [Help Menu](#) are visible at all times. If you activate ToolServer via the IDE Extras preference panel, then the [Tools Menu \(Mac OS\)](#) is also visible at all times.

The [Version Control System \(VCS\) Menu](#) is displayed only if you have installed and configured your CodeWarrior product to work with a compatible revision control system that you purchased separately. To learn more about revision control systems, and how to use them with CodeWarrior, refer to the documentation that came with the additional revision control software.

The [Data Menu](#), [Browser Menu](#), and [Layout Menu](#) are displayed when a window that can use their respective commands is present on the screen.

Menus for Mac OS only

The [Tools Menu \(Mac OS\)](#) appears if ToolServer is active. The [Scripts Menu \(Mac OS\)](#) and [Editor Extensions Menu \(Mac OS\)](#) are displayed when you set the correct preferences in the Preferences window. Refer to [“IDE Extras” on page 259](#) for more information on how to do this.

In addition, you can make the commands that appear in many menus change by holding down the Option or Shift keys when pulling the menu down.

The text of some menu commands may change depending on the context of the action, or actions, performed.

Many menu commands can also have button equivalents on the toolbars. To learn more about how to customize the toolbars, refer to [“Customizing the IDE” on page 296](#).

File Menu

The File Menu contains commands you use when opening, creating, saving, closing, and printing existing or new source code files and projects. The File Menu also provides different methods for saving edited files.

New Text File

Creates a new editable text file.

To learn more about this command, refer to [“Creating a New File” on page 109](#) for more information.

New

Opens the New dialog box. This dialog box helps you create various items in the CodeWarrior IDE.

Open

Allows you to open an existing file.

To learn more about this command, refer to [“Opening Files from the File Menu” on page 110](#).

Open Recent

Exposes a submenu of projects and files that were recently opened. You can choose a file from the submenu to instantly open that item.

If two or more files in the submenu have identical names, the full paths to those files are displayed in order to distinguish them.

To learn more about this command, refer to [“Opening Files from the File Menu” on page 110](#).

Find and Open File

Opens an existing file, searching the current access paths as specified in the [Access Paths](#) panel of the Target Settings window.

See [“Opening Files from an Editor Window” on page 115](#) for more information.

Find and Open ‘Filename’

Opens an existing text file, using the currently selected text in the editor window as the target file name.

See [“Opening Files from an Editor Window” on page 115](#) for more information.

Close

Closes the active window.

See [“Closing a File” on page 123](#) for more information.

To learn how to close all open editor windows, refer to [“Closing All Files” on page 125](#).

Close All

Closes all open windows of a certain type. The menu command is based on the type. For example, when several editor windows are currently open and one of them is selected, the menu command is **Close All Editor Documents**.

To learn more about this topic, refer to [“Closing All Files” on page 125](#).

Save

Saves the contents of the active window to disk.

For more information on this topic, refer to [“Saving one file” on page 117](#).

Save All

Saves all editor files that are currently open.

For more information on this topic, refer to [“Saving all files” on page 118.](#)

Save As

Saves the contents of the active window to disk under another name of your choosing.

For more information, see [“Renaming and saving a file” on page 118.](#)

Save A Copy As

Saves the active window in a separate file. This command operates in different ways, depending on the active window.

For more information, see [“Backing up files” on page 120.](#)

Revert

Reverts the active editor window to its last saved version.

To learn more about reverting a file, see [“Reverting to a Previously-Saved File” on page 127.](#)

Import Components

Imports components from another catalog for use with the current catalog.

For more information about this command, see [“Component Catalog Toolbar” on page 551.](#)

Close Catalog

Closes the current catalog and removes it from the [Component Catalog window](#) and the [Component Palette](#).

For more information about this command, see [“Component Catalog Toolbar” on page 551.](#)

Import Project

Imports an extensible markup language (XML) file into the IDE so you can save the XML file as a CodeWarrior project. The IDE

prompts you to choose a name and location to save the new project file.

For more information, see [“Importing and Exporting a Project” on page 103](#).

Export Project

Exports a CodeWarrior project to extensible markup language (XML) format. The IDE prompts you to choose a name and location to save the new XML file.

To learn more, see [“Importing and Exporting a Project” on page 103](#).

Print Setup (Windows)

Sets the options used when printing files from the CodeWarrior IDE.

For more information about this command, see [“Setting Print Options” on page 125](#).

Page Setup (Mac OS)

Sets the options used when printing files from the CodeWarrior IDE.

For more information about this command, see [“Setting Print Options” on page 125](#).

Print

Prints files from the CodeWarrior IDE on your printer.

For more information on printing files, see [“Printing a Window” on page 126](#), or read the documentation that came with your printer.

Exit (Windows)

Exits the CodeWarrior IDE immediately, provided one of the following conditions has been met:

- All changes to the open Editor files have already been saved, or
- The open Editor files have not been changed.

If a Project window is open, all changes to the project file are saved before the IDE exits. If an Editor window is open and changes have not been saved, the CodeWarrior IDE asks if you want to save the changes before exiting.

Quit (Mac OS)

Quits the CodeWarrior IDE immediately, provided one of the following conditions has been met:

- All changes to the open Editor files have already been saved, or
- The open Editor files have not been changed.

If a Project window is open, all changes to the project file are saved before the IDE quits. If an Editor window is open and changes have not been saved, the CodeWarrior IDE asks if you want to save the changes before quitting.

Edit Menu

The Edit menu contains all the customary editing commands, along with some CodeWarrior additions. This menu also includes the commands that open the Preferences and Target Settings windows.

Undo

The text of this menu command varies depending on the most recent action, and your Editor options settings.

Undo reverses the effect of your last action. The name of the Undo command varies depending on the type of operation you last executed. For example, if you have just typed in an open Editor window, the Undo command is renamed **Undo Typing**. Choosing the **Undo Typing** command will remove the text you have just typed.

To learn more about this topic, refer to [“Undoing the last edit” on page 156](#), and [“Undoing and redoing multiple edits” on page 156](#).

If you don't have [Use multiple undo](#) turned on in the [Editor Settings](#) preference panel, the Undo command toggles between Undo and Redo. To learn more about how to configure this option, refer to [“Editor Settings” on page 273](#).

Redo, Multiple Undo, and Multiple Redo

Once an operation has been undone, it may be redone. For example, if you select the **Undo Typing** command, the command is changed to **Redo Typing**. Choosing this command overrides the previous undo.

If you have [Use multiple undo](#) turned on in the [Editor Settings](#) preference panel, you have more flexibility with regard to Undo and Redo operations. Choose **Undo** multiple times to undo multiple actions. Choose **Redo** multiple times to redo multiple actions.

To learn more about undo operations, refer to [“Undoing the last edit” on page 156](#), and [“Undoing and redoing multiple edits” on page 156](#).

To learn about how to configure multiple undo, refer to [“Editor Settings” on page 273](#).

Cut

Deletes the selected text and puts it in the system Clipboard, replacing the contents of the Clipboard.

Copy

Copies the selected text in the active Editor window onto the system Clipboard. If the Message Window is active, the Copy command copies all the text in the Message Window onto the Clipboard.

Paste

Pastes the contents of the system Clipboard into the active Editor window.

The **Paste** command replaces the selected text with the contents of the Clipboard. If no text is selected, the Clipboard contents are placed after the text insertion point.

If the active window is the Message Window, the **Paste** command is dimmed and cannot be executed.

Clear

Deletes the selected text without placing it in the system Clipboard. The **Clear** command is equivalent to pressing the Delete or Backspace key.

Select All

Selects all the text in the active window. This command is usually used in conjunction with other **Edit** menu commands such as **Cut**, **Copy**, and **Clear**.

To learn more about selecting text, refer to [“Selecting Text” on page 152.](#)

Balance

Selects the text enclosed in either parentheses (), brackets [], or braces {}. For a complete procedure on how to use this command and how to balance while typing, consult [“Balancing Punctuation” on page 154.](#)

Shift Left

Shifts the selected source code one tab size to the left. The tab size is specified in the Preferences window.

To learn more about this feature, refer to [“Shifting Text Left and Right” on page 155.](#)

Shift Right

Shifts the selected source code one tab size to the right.

To learn more about this feature, refer to [“Shifting Text Left and Right” on page 155.](#)

Insert Reference Template (Mac OS)

Inserts a routine template corresponding to the selected Mac OS Toolbox call in the active window. The CodeWarrior IDE uses the online reference database application selected in the **Find Reference Using** pop-up menu to search for the routine’s definition.

To learn about configuring the online reference database application and the **Find Reference Using** pop-up menu, refer to [“IDE Extras” on page 259](#).

Preferences

Use this command to change the global preferences for the CodeWarrior IDE.

To learn more about configuring preferences, refer to [“Choosing Preferences” on page 256](#).

Target Settings

Use this command to display the Target Settings window where you can change settings for the active build target. Note that the name of this menu command will vary depending on the name of your current build target.

To learn more about the Settings window, refer to [“Choosing Target Settings” on page 324](#). To learn how to change the current build target, refer to [“Set Default Target” on page 638](#).

Version Control Settings

This menu command displays the Version Control System options panel. To learn more about this panel, refer to the section of this manual entitled [“Using the CodeWarrior IDE with Version Control Systems” on page 601](#).

If this command is not enabled, you do not have a revision control system configured for use with the CodeWarrior IDE.

Commands & Key Bindings

This command displays the Customize IDE Commands window.

Search Menu

The Search Menu contains all the necessary commands used to find text, replace text, and compare files. There are also some commands for code navigation.

Mac OS Use the commands in this menu to find the definitions of routines in your source code and in a library reference database like THINK Reference™ or Apple Macintosh Programmer's Toolbox Assistant™.

Find

Opens the Find dialog box which is used to find and/or replace the occurrences of a specific string in one or many files.

To learn more about the Find window and its capabilities, refer to [“Guided Tour of the Find Dialog Box” on page 169.](#)

Find Next

Finds the next occurrence of the [Find text box](#) string in the active window. This is an alternative to clicking the **Find** button in the Find dialog box.

To learn more about this feature, refer to [“Finding Search Text” on page 182.](#)

Find Previous

Find Previous operates the same way as **Find Next**, except that it finds the *previous* occurrence of the [Find text box](#) string.

To learn more about this feature, refer to [“Finding Search Text” on page 182.](#)

Find in Next File

Finds the next occurrence of the [Find text box](#) string in the next file listed in the Multi-File Search portion of the Find window (as exposed by the [Multi-File Search Disclosure triangle](#) in the Find window). This is an alternative to using the Find window. If the [Multi-File Search button](#) is not enabled as shown in [Figure 6.3 on page 175](#), this command is dimmed.

To learn more about this feature, refer to [“Finding and Replacing Text in Multiple Files” on page 189.](#)

Find in Previous File

This command operates in much the same way as **Find in Next File**. The **Find in Previous File** command begins at the end of the previous file in the file list and searches for the next occurrence of the [Find text box](#) string.

To learn more about this feature, refer to [“Finding and Replacing Text in Multiple Files” on page 189](#).

Enter ‘Find’ String

This command copies the selected text in the active window into the [Find text box](#), making it the search target string. This is an alternative to copying text and pasting it into the Find window.

To learn how to select text, refer to [“Selecting Text” on page 152](#).

Enter ‘Replace’ String

This command copies the selected text in the active window into the [Replace text box](#), making it the replacement string. This is an alternative to selecting the string and copying it into the Find window.

To learn more about replacing text, refer to [“Replacing Found Text” on page 185](#).

Find Selection

Finds the next occurrence of the selected text in the active text editor window.

To learn more about this feature, refer to [“Finding Search Text” on page 182](#).

Find Previous Selection

This command finds the previous occurrence of the selected text in the active text editor window.

To learn more about this feature, refer to [“Finding Search Text” on page 182](#).

To learn how to select text, refer to [“Selecting Text” on page 152](#).

Replace

This command replaces the selected text in the active window with the text string in the [Replace text box](#) of the Find window. If no text is selected in the active editor window, this command is dimmed.

This command is useful if you wish to replace one instance of a text string without having to open the Find window. For example, say that you have just replaced all the occurrences of the variable “icount” with “jcount”. While scrolling through your source code, you notice one instance of the variable “icount” is misspelled as “icont”. To replace this variable with “jcount”, select “icont” and choose the **Replace** command from the [Search Menu](#).

To learn more about replacing text, refer to [“Replacing Found Text” on page 185](#).

To learn how to select text, refer to [“Selecting Text” on page 152](#).

Replace & Find Next

This command replaces the selected text with the string in the [Replace text box](#) of the Find window, and then performs a [Find Next](#). If no text is selected in the active editor window and there is no text in the [Find text box](#) string field of the Find window, this command is dimmed.

To learn more about replacing text, refer to [“Replacing Found Text” on page 185](#).

To learn how to select text, refer to [“Selecting Text” on page 152](#).

Replace & Find Previous

This command operates the same way as [Replace & Find Next](#) except that it performs a [Find Previous](#) after replacing text.

Replace All

Finds all the occurrences of the Find string and replaces them with the Replace string. If no text is selected in the active editor window

and there is no text in the Find text box in the Find dialog box, this command is dimmed.

Find Definition

This command searches for the definition of the routine name selected in the active window. Searching occurs in the source files belonging to the open project. If the definition is found, the CodeWarrior IDE opens the source code file where the routine is defined and highlights the routine name.

If the CodeWarrior IDE finds more than one definition, a Message window appears warning you of multiple definitions. For more information on the Message window, consult [“Using the Message Window” on page 382](#).

If no definition is found, a system beep sounds.

Find Reference (Mac OS)

This command searches for the definition of the routine name selected in the active editor window using the online help system specified in the IDE Extras preference panel.

Searching takes place in QuickHelp™, QuickView™, THINK Reference™ version 2.x, or Toolbox Assistant™.

If no definition is found, a system beep sounds.

For more information about choosing an online reference database, refer to [“IDE Extras” on page 259](#).

To learn more about online databases, refer to [“Online References” on page 162](#).

Find Definition & Reference (Mac OS)

This command searches for the definition of the routine name selected in the active editor window. Searching starts within the source code files belonging to the open project.

To use this command, first select a routine name. Then hold down the Option key and choose **Find Definition & Reference** from the Search Menu.

If the routine definition is not found within the project files, searching continues in QuickHelp™, QuickView™, THINK Reference™ version 2.x, or Toolbox Assistant™.

If no definition is found, a system beep sounds.

For more information about choosing an online reference database, refer to [“IDE Extras” on page 259](#).

To learn more about online databases, refer to [“Online References” on page 162](#).

Go Back

This command returns you to the previous view in the Browser.

To learn more about this feature, refer to [“Go Back and Go Forward” on page 246](#).

Go Forward

This command moves you to the next view in the Browser (after you have used the [Go Back](#) command to return to a previous view).

Refer to [“Go Back and Go Forward” on page 246](#) for more information.

Go To Line

Opens a dialog box (in which you enter a line number) and then moves the text insertion point to the line number you specify.

For more information about this feature, refer to [“Going to a Particular Line” on page 161](#).

Compare Files

Opens a dialog box to choose two files or folders to compare and merge. After choosing files to compare, a file comparison window appears, showing differences between the two files. If two folders are compared, the differences between the folders are shown in the Compare Folders window. For more information, see [“Comparing and Merging Files & Folders” on page 127](#).

Apply Difference

Adds, removes, or changes text in the destination file shown in a file comparison window that is different from the text in the comparison window's source file.

Unapply Difference

Reverses the action of an **Apply Difference** command in a file comparison window.

Project Menu

The Project menu lets you add and remove files and libraries from your project. It also lets you compile, build, and link your project. All of these commands are covered in this section.

Add Window

This command adds the file in the active Editor window to the open project.

To learn more about this feature, refer to [“Using the Add Window command” on page 85.](#)

Add Files

This menu command adds files to the Project window.

To learn more about this feature, refer to [“Using the Add Files command” on page 79.](#)

Create New Group

The **Create New Group** command allows you to create a new group in the current project. This command is present in the Project menu if the Files category is selected in the current project window.

For more information about creating groups, refer to [“Creating Groups” on page 87.](#)

Create New Target

The **Create New Target** command allows you to create a new build target for the current project. This command is present in the Project menu if the Targets category is selected in the current project window.

For more information about creating targets, refer to [“Working with Complex Projects” on page 91.](#)

Create New Segment (Mac OS)

The **Create New Segment** command allows you to create a new segment (also referred to as a group of files) in the current Mac OS 68K project. This command is in the Project menu if the Segments category is selected in the current project window.

For more information about managing segments, refer to [“Managing Files in a Project” on page 75.](#)

Remove Selected Items

This menu command removes the currently selected items from the Project window. To learn more about removing items from the Project window, refer to [“Managing Files in a Project” on page 75.](#)

WARNING!

This command cannot be undone.

Check Syntax

This command checks the syntax of the source code file in the active Editor window or the selected file(s) in the open Project window. If the active Editor window is empty, or no project is open, this command is dimmed.

Check Syntax does not generate object code. This command only checks the source code for syntax errors. The progress of this operation is tracked in the Toolbar’s message area.

To abort this command at any time, press Esc (Windows) or Command-Period (Mac OS).

If one or more errors are detected, the Message window appears. For information on how to correct compiler errors, consult [“Correcting Compiler Errors and Warnings” on page 385.](#)

Preprocess

This command performs preprocessing on selected source code files in any language that has a preprocessor, including C, C++, and Pascal.

To learn more about this command, refer to [“Preprocessing Source Code” on page 378.](#)

Precompile

This command precompiles the text file in the active Editor window into a precompiled header file.

To learn more about this topic, refer to [“Using Precompiled or Preprocessed Headers” on page 371.](#)

Compile

This command compiles selected files. If the project window is active, the selected files and segments/groups are compiled. If a source code file in an Editor window is active, the source code file is compiled. The source code file must be in the open project.

To learn more about this topic, refer to [“Compiling and Linking a Project” on page 363.](#)

Disassemble

This command disassembles the compiled source code files selected in the project window, and displays object code in new windows with the title of the source code file and the extension `.dump`.

To learn more about this feature, refer to [“Disassembling Source Code” on page 379.](#)

Bring Up To Date

This command updates the open project by compiling all of its modified and “touched” files.

To learn more about this topic, refer to [“Updating a Project” on page 366](#).

Make

This command builds the selected project by compiling and linking the modified and “touched” files in the open project. The results of a successful build depend on the selected project type.

To learn more about this topic, refer to [“Making a Project” on page 366](#).

Stop Build

This command halts the build currently in progress.

To learn more about this topic, refer to [“Compiling and Linking a Project” on page 363](#).

Remove Object Code

This command removes all compiled source code binaries from the open project. The numbers in the [Code column](#) and [Data column](#) of each file are reset to zero.

To learn more about this topic, refer to [“Removing Object Code” on page 369](#).

Remove Object Code & Compact

This command removes all binaries from the project and compacts it. Compacting the project removes all binary and debugging information and retains only the information regarding the files that belong to the project and project settings.

To learn more about this topic, refer to [“Removing Object Code” on page 369](#).

Re-search for Files

To speed up builds and other project operations, the IDE caches the locations of project files after it has found them in the access paths. **Re-search for Files** forces the IDE to forget the cached locations of files and re-search for them in the access paths. This command is

useful if you have moved files around on disk and want the IDE to find them in their new locations.

If the [Save project entries using relative paths](#) setting is enabled the IDE does not reset the relative path information stored with each project entry, so re-searching for files will find the source files in the same location (the exception is if the file no longer exists in the old location). In this case the IDE will only re-search for header files. To force the IDE to also re-search for source files, you must first select **Reset Project Entry Paths**.

If the **Save Project Entries Using Relative Paths** setting is disabled, the IDE will re-search for both header and source files.

Reset Project Entry Paths

This command resets the location information stored with each project entry when the [Save project entries using relative paths](#) setting is enabled. The next time the project entries are accessed, the IDE will re-search for the project entries in the access paths. This command does nothing if the **Save Project Entries Using Relative Paths** setting is disabled.

Synchronize Modification Dates

This command updates the modification dates stored in the project file. It checks the modification date for each file in the project, and if the file has been modified since it was last compiled, the CodeWarrior IDE marks it for recompilation.

To learn more about this topic, refer to [“Synchronizing modification dates” on page 90](#).

Enable Debugger

Use these commands to change allowances for project debugging. The **Enable Debugger** command sets preferences to allow your project to be debugged. The **Disable Debugger** sets preferences so no debugging can occur.

Mac OS When you choose the **Enable Debugger** command, the [Run](#) command changes to **Debug** and lets the debugger launch and debug your project. When you choose the **Disable Debugger** command, the **Run** command runs your project normally.

To learn more about this topic, refer to [“Controlling Debugging in a Project” on page 104.](#)

Disable Debugger

Use these commands to change allowances for project debugging. The **Enable Debugger** command sets preferences to allow your project to be debugged. The **Disable Debugger** sets preferences so no debugging can occur.

To learn more about this topic, refer to [“Controlling Debugging in a Project” on page 104.](#)

Run

This command compiles, links, creates a stand-alone application, and launches that application.

Windows If the project type is set as a library or a shared library, then the **Run** command is dimmed.

Mac OS If the project type is set as a code resource, library, MPW Tool, or shared library, then the **Run** command is dimmed. Press the Option key to change this command to **Debug**.

To learn more about this topic, refer to [“Running a Project” on page 367.](#)

Resume (Mac OS)

This command appears in the **Project** menu when the application created by the CodeWarrior IDE is running already. You choose this command to switch into the application from the CodeWarrior IDE.

Debug

This menu command compiles and links your project and then opens the project’s debugger file in the debugger. This command runs the debugger for any project that the debugger can work with.

Mac OS This command appears in the **Project** menu when you press the Option key to change the **Run** command to **Debug**, or when the debugger is enabled.

Set Default Project

This menu command selects which project is the default project. To learn more about what a default project is, refer to [“Choosing a Default Project” on page 74.](#)

Set Default Target

This menu command allows you to choose a different target within the current project to work with. This menu command might be useful if you want to switch between multiple targets in a project and do a build for each one.

Debug Menu

The Debug menu contains commands that allow you to manage program execution. All of these commands are available in both the integrated and external debuggers. There is also a submenu for customizing breaks for Java exceptions.

Kill

Permanently terminates execution of the target program and returns control to the IDE.

Restart

Restarts the debugging session.

Reset (Embedded)

Resets the program and returns control to the IDE.

Hard Reset (Embedded)

Resets the program and also resets the hardware before returning control to the IDE.

Step Over

Executes a single statement, stepping over function calls.

Step Into

Executes a single statement, stepping into function calls.

Step Out

Executes the remainder of the current function until it exits to its caller.

Stop

Temporarily suspends execution of the target program and returns control to the debugger.

Set Breakpoint

Sets a breakpoint at the currently-selected line. This command toggles between **Set Breakpoint** and **Clear Breakpoint**, depending on whether you have a breakpoint currently set in a line.

If you have your editor window set up to show breakpoints (by choosing **Debug -> Show Breakpoints**), you will see the breakpoints column on the left side of the editor window. If you set a breakpoint in a certain line of code, a marker appears in the breakpoints column next to that line. If you clear an existing breakpoint from a certain line, the marker to the left of that line disappears from the breakpoints column.

Clear Breakpoint

Clears the breakpoint at the currently-selected line. This command toggles between **Set Breakpoint** and **Clear Breakpoint**, depending on whether you have a breakpoint currently set in a line.

If you have your editor window set up to show breakpoints (by choosing the **Show Breakpoints** command from the Debug menu), you will see the breakpoints column on the left side of the editor window. If you set a breakpoint in a certain line of code, a marker appears in the breakpoints column next to that line. If you clear an existing breakpoint from a certain line, the marker to the left of that line disappears from the breakpoints column.

Enable Breakpoint

Enables the breakpoint at the currently-selected line. This command toggles between **Enable Breakpoint** and **Disable Breakpoint**, depending on whether you have a breakpoint currently enabled in the line.

If you have your editor window set up to show breakpoints (by choosing **Debug > Show Breakpoints**), you will see the breakpoints column on the left side of the editor window. Darkened markers in the breakpoints column indicate enabled breakpoints. Dimmed markers indicate disabled breakpoints.

Disable Breakpoint

Disables the breakpoint at the currently-selected line. This command toggles between **Enable Breakpoint** and **Disable Breakpoint**, depending on whether you have a breakpoint currently enabled in the line.

If you have your editor window set up to show breakpoints (by choosing **Debug -> Show Breakpoints**), you will see the breakpoints column on the left side of the editor window. Darkened markers in the breakpoints column indicate enabled breakpoints. Dimmed markers indicate disabled breakpoints.

Clear All Breakpoints

Clears all breakpoints in all source-code files belonging to the target program.

Show Breakpoints

Displays the breakpoints column, which appears in editor windows to the left of the source code. This command toggles between **Show Breakpoints** and **Hide Breakpoints**, depending on whether the breakpoints column is currently visible on the active editor window.

Hide Breakpoints

Hides the breakpoints column, which appears in editor windows to the left of the source code. This command toggles between **Show Breakpoints** and **Hide Breakpoints**, depending on whether the

breakpoints column is currently visible on the active editor window.

Set Watchpoint

Sets a watchpoint for the selected variable or range of memory. This command toggles between **Set Watchpoint** and **Clear Watchpoint**, depending on whether the watchpoint is currently set in the active editor window.

The watchpoint is indicated by an underline. You can configure the color of this underline using the Display Settings preference panel in the IDE Preferences window. For more information, refer to [“Display Settings” on page 283](#).

Clear Watchpoint

Clears a watchpoint for the selected variable or range of memory. This command toggles between **Set Watchpoint** and **Clear Watchpoint**, depending on whether the watchpoint is currently set in the active editor window.

The watchpoint is indicated by an underline. You can configure the color of this underline using the Display Settings preference panel in the IDE Preferences window. For more information, refer to [“Display Settings” on page 283](#).

Enable Watchpoint

Enables a watchpoint for the selected variable or range of memory. This command toggles between **Enable Watchpoint** and **Disable Watchpoint**, depending on whether you have a watchpoint currently enabled.

Enabled watchpoints are indicated by an underline for the selected variable or range of memory. Disabled watchpoints do not have the underline. The underline’s color can be configured in the Display Settings preference panel of the IDE Preference window. See [“Display Settings” on page 283](#) for more information.

Disable Watchpoint

Disables a watchpoint for the selected variable or range of memory. This command toggles between **Enable Watchpoint** and **Disable**

Watchpoint, depending on whether you have a watchpoint currently enabled.

Enabled watchpoints are indicated by an underline for the selected variable or range of memory. Disabled watchpoints do not have the underline. The underline's color can be configured in the Display Settings preference panel of the IDE Preference window. See [“Display Settings” on page 283](#) for more information.

Clear All Watchpoints

Clears all watchpoints in the current program.

Break on C++ Exception

Causes the debugger to break at `__throw()` every time a C++ exception occurs.

Break on Java Exceptions

This menu item causes the Java Exceptions submenu to appear. To learn more about this submenu, refer to [“Java Exceptions Submenu” on page 653](#).

Switch to Monitor

Gives control to an external debugger that you may have installed on your computer.

Mac OS The low-level Macintosh ROM Monitor program and MacsBug are two examples of external debuggers.

Data Menu

The Data menu lets you control how data values are displayed in the debugger. All of these commands are available in both the integrated and external CodeWarrior debuggers. This menu is present anytime a window that can employ the Data commands is frontmost on the screen. Otherwise, it is hidden from view.

Show Types

Shows the data types of all local and global variables displayed in the active variable pane or variable window.

New Expression

Creates a new entry in the Expressions window, prompting you to enter a new expression.

Copy to Expression

Copies the variable selected in the active pane to the Expressions window. You can also drag an expression to the Expressions window from source code or from another window or pane.

View As...

Displays a selected variable as a value of a specified data type.

View Variable

Creates a separate window to display a selected variable.

View Array

Creates a separate window to display a selected array.

View Memory

Displays the contents of memory as a hexadecimal/ASCII character dump.

View Memory As...

Displays the memory a selected variable occupies or a selected register points to.

Default

Displays the selected variable in its default format based on the variable type.

Signed Decimal

Displays the selected variable as a signed decimal value.

Unsigned Decimal

Displays the selected variable as an unsigned decimal value.

Hexadecimal

Displays the selected variable as a hexadecimal value.

Character

Displays the selected variable as a character value.

C String

Displays the selected variable as a C character string.

Pascal String

Displays the selected variable as a Pascal character string.

Unicode String

Displays the selected variable as a Unicode character string.

Floating Point

Displays the selected variable as a floating-point value.

Enumeration

Displays the selected variable as an enumeration.

Fixed (Mac OS)

Displays the selected variable as a numerical value of type `Fixed`.

Fract (Mac OS)

Displays the selected variable as a numerical value of type `Fract`.

Browser Menu

The Browser menu lets you create new classes, member functions, and data members in the active project. This menu is present when a window that can employ the Browser commands is open. Otherwise, the menu is hidden from view.

New Class

Displays a dialog box to help you create a new class for your project.

New Member Function

Displays a dialog box to help you create a new member function for your project.

New Data Member

Displays a dialog box to help you create a new data member for your project.

New Property

Displays a dialog box to help you create a new property for a selected class in your project.

New Method

Displays a dialog box to help you create a new method for a selected class in your project.

New Event Set

Displays a dialog box to help you create a new event set for a selected class in your project.

New Event

Displays a dialog box to help you create a new event for a selected class in your project.

Window Menu

The Window menu includes commands that tile open editor windows, switch between windows, and open debugger windows. There is also a submenu for customizing the toolbars.

Stack Editor Windows

This command opens all editor windows to their full screen size and stacks them one on top of another, with their window titles

showing. This command is dimmed when the active window is the Project window or Message window.

Tile Editor Windows

This command arranges all editor windows so that none overlap. This command is dimmed when the active window is the Project window or Message window.

Tile Editor Windows Vertically

This command arranges all editor windows so that none overlap. Additionally, the windows are oriented vertically across the screen.

Tile Vertical

This command arranges all the Editor windows in a single row.

This command is disabled when the active window is the Project window or Message window.

Mac OS ToolServer Worksheets also disable this command.

Zoom Window

This menu command expands the active window to the largest possible size. If you choose the menu command again, the window returns to its original size.

Minimize Window (Windows)

Minimizes the currently active window.

Collapse Window (Mac OS)

Collapses the currently active window.

Restore Window (Windows)

Restores a window to its original size.

Expand Window (Mac OS)

Expands a window to its original size.

Save Default Window

This command saves the settings of the active browser window, so that the next time you open a browser window, the CodeWarrior IDE opens it with the saved settings.

To learn more about this command, refer to, [“Saving Editor Window Settings” on page 148](#), or [“Saving a Default Browser” on page 251](#).

Toolbar

This menu item causes the Toolbar submenu to appear. To learn more about this submenu, refer to [“Toolbar Submenu” on page 653](#).

Browser Contents

This command displays the Browser [Contents Window](#). This menu command is dimmed when the Browser is not activated.

To learn more about this feature, refer to [“Contents Window” on page 215](#). To learn how to activate the Browser, refer to [“Activating the Browser” on page 207](#).

Class Hierarchy Window

This command displays the Browser’s [Multi-Class Hierarchy Window](#). This menu command is dimmed when the Browser is not activated.

To learn more about this feature, refer to [“Multi-Class Hierarchy Window” on page 224](#).

To learn how to activate the Browser, refer to [“Activating the Browser” on page 207](#).

New Class Browser

This command displays the Browser’s [Browser Window](#). This menu command is dimmed when the Browser is not activated.

To learn more about this feature, refer to [“Browser Window” on page 216](#).

To learn how to activate the Browser, refer to [“Activating the Browser” on page 207.](#)

Build Progress Window

This menu command displays the progress window for builds, as shown in [Figure 10.1 on page 364.](#)

Errors & Warnings Window

This command displays the Errors and Warnings window.

To learn more about this window, refer to [“Guided Tour of the Message Window” on page 379.](#) Also, refer to [“Using Batch Searches” on page 187.](#)

Project Inspector

This menu command allows you to view information about your project and enable debug information generation.

To learn more about this command’s window, refer to [“Guided Tour of the Project Window” on page 42.](#)

ToolServer Worksheet (Mac OS)

Displays the ToolServer Worksheet window. This window is used in conjunction with ToolServer. This command is disabled provided one of the following conditions have been met:

- ToolServer is not installed on your machine.
- ToolServer is installed on your machine, but has not yet been started. To start ToolServer, select the Start ToolServer command in the Tools menu.

To learn more about ToolServer, refer to *Targeting Mac OS*.

Processes Window

Displays the Processes window.

Expressions Window

Displays the Expressions window.

Global Variables Window

Displays the Global Variables window. Within this window you can view the global variables for the entire project or those contained in a file. Click on a file name in the Files list to display the file's global variables in the Variables list.

Breakpoints Window

Displays the Breakpoints window.

Watchpoints Window

Displays the Watchpoints window.

Registers Windows

Displays the Registers submenu, from which you can choose to view general registers or FPU registers.

Component Catalog

Displays the [Component Catalog window](#) for use with RAD projects.

Component Palette

Displays the [Component Palette](#) for use with RAD projects.

Object Inspector

Displays the [Object Inspector](#) for use with RAD projects.

Other Window Menu Items

The other **Window** menu items depend solely on which project, source files, header files, interface files, and other windows you have open.

All of the open windows are shown in this menu and the first nine files (1 through 9) are given key equivalents. The current project is always assigned the number 0 (zero). Press Ctrl/Command and a number to open a specific Editor window. A check mark is placed beside the active window.

Mac OS A file whose modifications have not been saved is underlined.

To make one of your open CodeWarrior files active and bring its window to the front, do one of the following:

- Click in its window.
- Select it from the [Window Menu](#).
- Use the key equivalent shown in the [Window Menu](#).

Layout Menu

The Layout menu includes commands that manipulate objects in a rapid application development (RAD) layout window. This menu is present when a window that can employ the Layout commands is open. Otherwise, the menu is hidden from view.

Group

Combines selected objects so that they move together as a group.

Ungroup

Separates a selected group so you can move each object independently.

Send To Back

Moves the selected objects so that they are displayed behind all other objects.

Bring To Front

Moves the selected objects so that they are displayed in front of all other objects.

Display Grid

When checked, displays the graphical grid in the layout window.

Snap To Grid

When checked, the Layout editor automatically aligns objects with the graphical grid.

Align To Grid

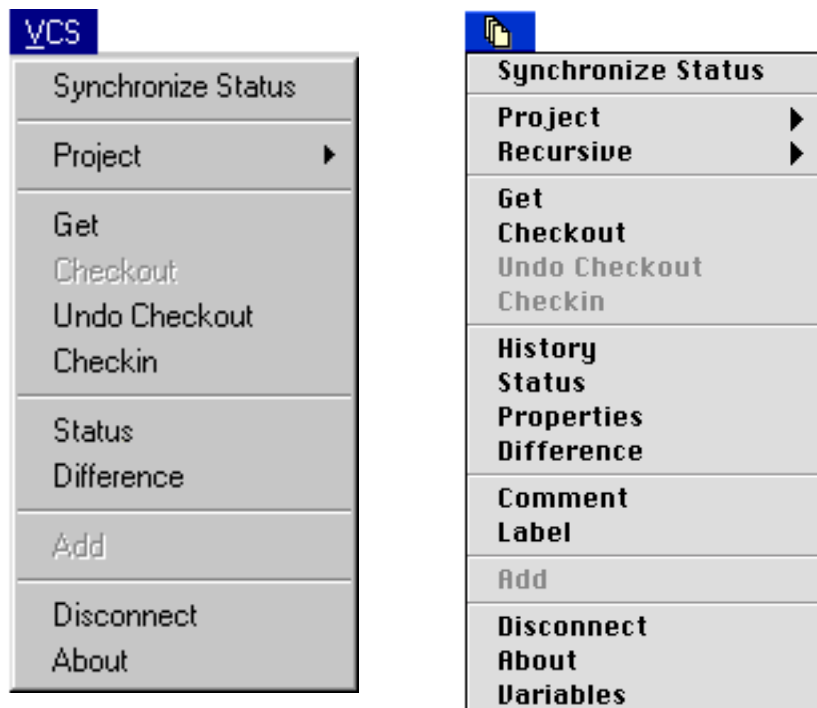
Positions selected objects so that they are aligned with a graphical grid in the layout window. This grid can be displayed or hidden from view.

Version Control System (VCS) Menu

The Version Control System (VCS) menu, shown in [Figure 17.1](#), may appear in the menu bar of your CodeWarrior IDE if you have purchased the MW Visual SourceSafe for Macintosh Version Control System (available separately) for use with the CodeWarrior IDE.

To learn how to configure MW Visual SourceSafe to use revision control with your projects, refer to the documentation for your MW Visual SourceSafe product.

Figure 17.1 VCS menus for Windows and Mac OS



Help Menu

Online help is available from the Help menu. When you are working in the CodeWarrior IDE, select one of the items to get interactive, online help.

CodeWarrior Help

Displays help for using the CodeWarrior.

How to

Displays step-by-step help topics.

Glossary

Displays a list of vocabulary terms.

IDE

Displays help for using the IDE.

Debugger

Displays help for using the debugger.

Error Reference

Displays help for referencing IDE errors.

C/C++ Compiler Reference

Displays help for the C/C++ compilers.

MSL C Reference

Displays help for the Metrowerks Standard Library (MSL) for C.

MSL C++ Reference

Displays help for the Metrowerks Standard Library (MSL) for C++.

Other

Displays other information specific to your CodeWarrior product.

About Metrowerks (Windows)

Displays the Metrowerks About Box.

Java Exceptions Submenu

The [Debug Menu](#) has another submenu under it for the **Break on Java Exceptions** command. The Java Exceptions submenu contains options that tell the debugger what to do when Java exceptions occur. This command is available when you are working with Java source code, and is dimmed for all other source code.

All Exceptions

This menu command causes the debugger to break every time an exception occurs. This includes exceptions thrown by the virtual machine, your own classes, the debugger, classes in `classes.zip`, and similar exceptions. Java throws many exceptions in the normal course of execution, so catching all exceptions causes the debugger to break often.

Exceptions in Targeted Classes

This menu command causes the debugger to break only on exceptions that are thrown by your own classes in the project. Select this menu command if you are interested in breaking on the exceptions thrown by your classes, rather than the exceptions that Java throws in the normal course of execution.

No Exceptions

When you select this menu command, the debugger will not break when exceptions occur.

Toolbar Submenu

The [Window Menu](#) has another submenu under it for the **Toolbar** command. The Toolbar submenu contains all the commands used to customize the toolbars that appear in CodeWarrior IDE windows.

To learn more about how to customize the toolbars, read the information in [“Customizing the IDE” on page 296](#).

Show Window Toolbar

This menu command cause the IDE to display the toolbar in the active window. The actual command shown in the menu will toggle between **Show Window Toolbar** and **Hide Window Toolbar**, depending on whether the active window's toolbar is visible or not.

Hide Window Toolbar

This menu command cause the IDE to hide the toolbar in the active window. The actual command shown in the menu will toggle between **Show Window Toolbar** and **Hide Window Toolbar**, depending on whether the active window's toolbar is visible or not.

Reset Window Toolbar

This menu command causes the toolbar in the active window to reset to a default state. You should use this menu command if you want to return the Editor window toolbar to its original default settings.

Clear Window Toolbar

This menu command causes the toolbar in the active Editor, Project, or Browser window to have all icons removed from it. Once all the icons have been removed, you can add icons using the Toolbar Elements window.

Use the **Reset Window Toolbar** command to cause all the default icons to come back.

Show Floating Toolbar

This menu command causes the Floating Toolbar to appear. The actual command shown in the menu will toggle between **Show Floating Toolbar** and **Hide Floating Toolbar**, depending on whether the Floating Toolbar is already visible or not.

Hide Floating Toolbar

This menu command causes the Floating Toolbar to disappear. The actual command shown in the menu will toggle between **Show Floating Toolbar** and **Hide Floating Toolbar**, depending on whether the Floating Toolbar is already visible or not.

Anchor Floating Toolbar (Mac OS)

This menu command causes the Floating Toolbar to anchor itself to the left edge of the screen, immediately below the menu bar. The actual command shown in the menu will toggle between **Anchor Floating Toolbar** and **Unanchor Floating Toolbar**, depending on whether the Floating Toolbar is currently anchored or not.

Unanchor Floating Toolbar (Mac OS)

This menu command causes the Floating Toolbar to unanchor itself from the left edge of the screen, immediately below the menu bar. The actual command shown in the menu will toggle between **Anchor Floating Toolbar** and **Unanchor Floating Toolbar**, depending on whether the Floating Toolbar is currently anchored or not.

Reset Floating Toolbar

This menu command causes the Floating Toolbar to return to its default state. You should use this menu command if you want to return the Floating Toolbar to the original default settings.

Clear Floating Toolbar

This menu command causes the Floating Toolbar to have all icons removed from it. Once all the icons have been removed, you can add icons using the Toolbar Elements window.

Apple Menu (Mac OS)

The Apple menu contains one IDE-related item.

About Metrowerks...

Choose this item to see the way-cool About Box.

Tools Menu (Mac OS)

The Tools menu contains commands used to do things like start and stop ToolServer. It also contains commands that execute Macintosh Programmer's Workbench (MPW) tools and scripts. Additional commands reference the MPW 411 database.

To learn more about how to use the CodeWarrior command-line tools, refer to the *CodeWarrior Command Line Tools* manual for more information.

To learn more about ToolServer, refer to *Targeting Mac OS*.

Start ToolServer

Initiates ToolServer handshaking. If you do not have ToolServer installed on your computer, this command is replaced by **Can't Find ToolServer**.

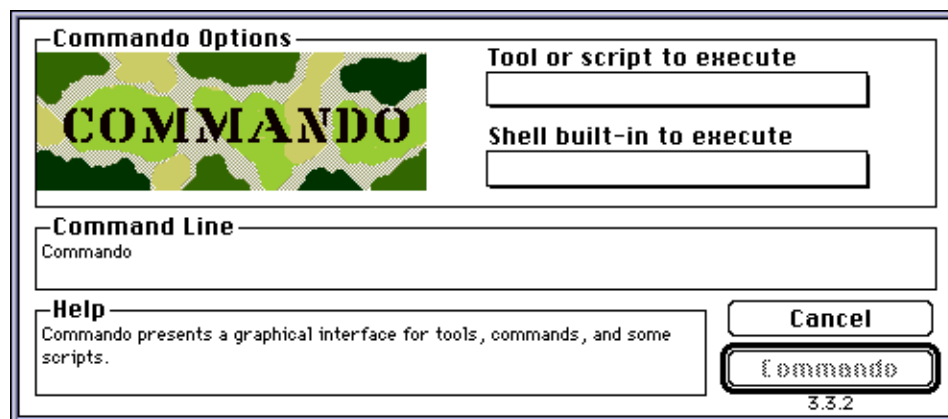
Stop ToolServer

When ToolServer is launched, the **Stop ToolServer** command replaces the **Start ToolServer** command. Stop ToolServer quits ToolServer and removes the extra ToolServer menu from the CodeWarrior IDE's menu bar.

Commando

This command brings up the Commando dialog box, as shown in [Figure 17.2](#). To learn more about Commando, refer to *Targeting Mac OS*.

Figure 17.2 Commando Dialog Box



Execute as ToolServer Script

Executes the ToolServer script in the active editor window. For more information, see *Targeting Mac OS*.

Lookup Symbol

Searches for information about the selected symbol in the MPW 411 database. For more information, see *Targeting Mac OS*.

Insert Template

Inserts the parameter list template for a function after the selected text. For more information, see *Targeting Mac OS*.

ToolServer Tools

You can use this submenu to run an MPW tool or script that is installed in the Tools folder of your ToolServer application. To learn more about ToolServer, refer to *Targeting Mac OS*.

Scripts Menu (Mac OS)

The Script menu, shown in [Figure 17.3](#), contains a list of the AppleScripts in the (Scripts) folder.

This menu will only appear if the **Use Script Menu** setting is turned on in the IDE Extras preference panel. Also, this menu will only be shown if the (Scripts) folder is present in your Metrowerks CodeWarrior folder. To learn more about how to configure the appropriate preferences, refer to [“IDE Extras” on page 259](#).

To learn more about AppleScripts and scripting the CodeWarrior IDE, refer to *Targeting Mac OS*.

To learn about what the individual scripts do, open them with a text editor or an AppleScript editor application.

Figure 17.3 The Script menu



Note that the menu is hierarchical, based on the directory structure of the (Scripts) folder. You may place scripts in the (Scripts) folder inside other directories to create a hierarchical menu.

Open Scripts Folder

Brings the Finder to the front and opens the (Scripts) folder. This command is always available if the Script menu is active.

Other Script menu items

The other items in this menu are the names of the AppleScripts in the (Scripts) folder. If you do not have any AppleScripts, no items are listed.

Editor Extensions Menu (Mac OS)

The Editor Extensions menu ([Figure 17.4](#)) is similar to the Script Menu. If the Use BBEdit™ Extensions preference is enabled, a list of BBEdit™ extensions are added to the menu bar. You must have an

alias or folder called (Editor Extensions), including the parentheses, in the same folder as the CodeWarrior IDE application. Place the editor extensions that you wish to use inside the (Editor Extensions) folder. You can also place aliases of the editor extensions inside the (Editor Extensions) folder.

To learn more about how to configure the appropriate preferences so this menu becomes visible, refer to [“IDE Extras” on page 259](#).

For more information about BBEdit, refer to the documentation that came with the product.

Figure 17.4 Editor Extensions menu



Info Menu (Solaris)

The Info menu ([Figure 17.5](#)) contains the following menu items:

About Metrowerks...

Choose this item to see the Metrowerks CodeWarrior about box.

About CodeWarrior Latitude...

Choose this item to see the CodeWarrior Latitude about box.

Figure 17.5 Solaris Info menu



Copy Files...

This item launches a desk accessory which enables you to copy files and folders. From within the CodeWarrior IDE, you can copy files and folders between file systems or devices accessible from your Solaris environment. See [“Copy Files Accessory” on page 679](#) for a detailed description of this accessory.

File Systems...

This facility allows you to mount volumes from within the CodeWarrior IDE. For more information on using this facility, refer to [“File Systems Facility” on page 682](#).

Keyboard Preferences...

Use this command to display the Keyboard Preferences dialog box where you can change the default modifier key mappings. To learn more about this settings panel, refer to [“Keyboard Preferences Dialog Box” on page 687](#).

IDE Default Key Bindings

This section describes the default key bindings assigned to commands in the CodeWarrior IDE.

CodeWarrior does not assign default key bindings to all of the available commands, so their respective key-binding lists appear blank. You can assign key bindings to any command. For more

information about key bindings, see [“Customizing Key Bindings” on page 304](#).

The key-binding sections include:

- [File Menu](#)
- [Edit Menu](#)
- [Search Menu](#)
- [Project Menu](#)
- [Debug Menu](#)
- [Data Menu](#)
- [Browser Menu](#)
- [Miscellaneous](#)
- [Editor Commands](#)
- [Catalog](#)
- [Layout Menu](#)
- [Window Menu](#)

Mac OS, Solaris, and Linux Modifier Key Legend

Modifier keys are used in combination with other keys to generate key bindings. For instance, to execute the Ctrl/Command - N keyboard shortcut, you first press the Ctrl or Command modifier key, and while holding it down, press the N key on your keyboard. For Mac OS-, Solaris-, and Linux-hosted versions of the CodeWarrior IDE, symbols are used to represent modifier keys in menus as well as the Customize IDE Commands window. The legend in [Table 17.1](#) describes how these symbols correspond to their associated names on Mac OS, Solaris, and Linux platforms.

Table 17.1 Mac OS, Solaris, and Linux modifier key legend

Symbol	Mac OS	Solaris and Linux
⌘	Command key	Meta key
⌥	Option key	Alt key
⇧	Shift key	Shift key
⌃	Control key	Control key

NOTE (Solaris and Linux) You can change modifier key mappings via the Keyboard Preferences dialog box. See [“Keyboard Preferences Dialog Box” on page 687](#) for more information on changing the default mappings.

File Menu

[Table 17.2](#) contains the default key bindings for manipulating projects and files from within the CodeWarrior IDE.

Table 17.2 File menu key bindings

Command	Windows	Mac OS	Solaris and Linux
New Text File	Ctrl-N	Command-N	Meta-N
New	Ctrl-Shift-N	Shift-Command-N	Shift-Meta-N
New Project			
New Empty Project			
Open	Ctrl-O	Command-O	Meta-O
Open Recent			

Command	Windows	Mac OS	Solaris and Linux
Find and Open 'selection'	Ctrl-D	Command-D	Meta-D
Find and Open File	Ctrl-Shift-D	Option-Command-D	Alt-Meta-D
Close	Ctrl-W	Command-W	Meta-W
Close All	Ctrl-Shift-W	Option-Command-W	Alt-Meta-W
Save	Ctrl-S	Command-S	Meta-S
Save All	Ctrl-Shift-S	Option-Command-S	Alt-Meta-S
Save As			
Save A Copy As			
Revert			
Import Components			
Close Catalog			
Import Project			
Export Project			
Page Setup			
Print	Ctrl-P	Command-P	Meta-P
Quit		Command-Q	Meta-Q

Edit Menu

[Table 17.3](#) contains the default key bindings for the commands in the Edit menu of the CodeWarrior IDE.

Table 17.3 **Edit key bindings**

Command	Windows	Mac OS	Solaris and Linux
Undo	Ctrl-Z	Command-Z	Meta-Z
Redo	Ctrl-Shift-Z	Shift-Command-Z	Shift-Meta-Z
Cut	Ctrl-X	Command-X	Meta-X
Copy	Ctrl-C	Command-C	Meta-C
Paste	Ctrl-V	Command-V	Meta-V
Clear		Clear	
Delete	Del		
Select All	Ctrl-A	Command-A	Meta-A
Balance	Ctrl-B	Command-B	Meta-B
Shift Left	Ctrl-[Command-[Meta-[
Shift Right	Ctrl-]	Command-]	Meta-]
Insert Reference Template			
Preferences			
Target Settings			
Version Control Settings			
Commands & Key Bindings			

Search Menu

[Table 17.4](#) contains the default key bindings for the commands in the Search menu of the CodeWarrior IDE.

Table 17.4 Search key bindings

Command	Windows	Mac OS	Solaris and Linux
Find	Ctrl-F	Command-F	Meta-F
Find Next	F3	Command-G	Meta-G
Find Previous		Shift-Command-G	Shift-Meta-G
Find in Next File	Ctrl-T	Command-T	Meta-T
Find in Previous File		Shift-Command-T	Shift-Meta-T
Enter Find String	Ctrl-E	Command-E	Meta-E
Enter Replace String		Shift-Command-E	Shift-Meta-E
Find Selection	Ctrl-F3	Command-H	Meta-H
Find Previous Selection		Shift-Command-H	Shift-Meta-H
Replace	Ctrl-=	Command-=	Meta-=
Replace & Find Next	Ctrl-L	Command-L	Meta-L
Replace & Find Previous		Shift-Command-L	Shift-Meta-L
Replace All			
Find Definition	Ctrl-'	Command-'	Meta-'
Find Definition & Reference		Option-Command-'	Alt-Meta-'
Find Reference		Shift-Command-'	Shift-Meta-'
Go Back	Ctrl-Shift-B	Shift-Command-B	Shift-Meta-B

Command	Windows	Mac OS	Solaris and Linux
Go Forward	Ctrl-Shift-F	Shift-Command-F	Shift-Meta-F
Go to Line	Ctrl-,	Command-,	Meta-,
Compare Files			
Apply Difference			
Unapply Difference			

Project Menu

[Table 17.5](#) contains the default key bindings for managing projects, controlling compilations, and much more.

Table 17.5 Project key bindings

Command	Windows	Mac OS	Solaris and Linux
Add Window			
Add Files			
Create Group/ Segment/ Target			
Check Syntax	Ctrl-;	Command-;	Meta-;
Preprocess			
Precompile			
Compile	Ctrl-F7	Command-K	Meta-K
Disassemble			
Bring Up To Date	Ctrl-U	Command-U	Meta-U
Make	F7	Command-M	Meta-M
Stop Build	Ctrl-Break	Command-.	Meta-.

Command	Windows	Mac OS	Solaris and Linux
Remove Object Code	Ctrl - -	Command - -	Meta - -
Remove Object Code & Compact		Option - Command - -	Alt - Meta - -
Re-search For Files			
Reset Project Entry Paths			
Synchronize Modification Dates			
Enable Debugging			
Run/Debug	F5	Command-R	Meta-R
Debug/Run	Ctrl-F5	Option-Command-R	Alt-Meta-R
Set Default Project			
Set Default Target			

Debug Menu

[Table 17.6](#) contains the default key bindings for handling commands on the Debug menu in the CodeWarrior IDE.

Table 17.6 Debug menu key bindings

Command	Windows	Mac OS	Solaris and Linux
Kill	Shift-F5	Control-K	Control-K
Restart	Ctrl-Shift-F5		
Reset			

Command	Windows	Mac OS	Solaris and Linux
Hard Reset			
Step Over	F10	Control-S	Control-S
Step Into	F11	Control-T	Control-T
Step Out	Shift-F11	Control-U	Control-U
Stop			
Set/Clear Breakpoint	F9		
Enable/Disable Breakpoint	Ctrl-F9		
Clear All Breakpoints			
Show/Hide Breakpoints			
Set/Clear Watchpoint			
Enable/Disable Watchpoint			
Clear All Watchpoints			
Break on C++ Exception			
Break on Java Exception			
Process Attach			
Switch to Monitor			

Data Menu

[Table 17.7](#) contains the default key bindings for handling debugger variable displays in the IDE.

Table 17.7 Data menu key bindings

Command	Windows	Mac OS	Solaris and Linux
Show Types			
New Expression	Alt-N	Control-N	Control-N
Copy To Expression			
View As			Control-Y
View Variable			
View Array		Control-A	Control-A
View Memory		Control-M	Control-M
View Memory As			
View As Default			
View As Signed Decimal	Alt-Shift-D	Control-Shift-D	Control-Shift-D
View As Unsigned Decimal	Alt-Shift-U	Control-Shift-U	Control-Shift-U
View As Hexadecimal	Alt-Shift-H	Control-Shift-H	Control-Shift-H
View As Character	Alt-Shift-C	Control-Shift-C	Control-Shift-C
View As C String	Alt-Shift-S	Control-Shift-S	Control-Shift-S
View As Pascal String	Alt-Shift-P	Control-Shift-P	Control-Shift-P
View As Unicode String	Alt-Shift-O	Control-Shift-O	Control-Shift-O
View As Floating Point	Alt-Shift-F	Control-Shift-F	Control-Shift-F

Command	Windows	Mac OS	Solaris and Linux
View As Enumeration	Alt-Shift-E	Control-Shift-E	Control-Shift-E
View As Fixed		Control-Shift-I	Control-Shift-I
View As Fract		Control-Shift-R	Control-Shift-R

Browser Menu

[Table 17.8](#) contains the default key bindings for use with the class browser in the CodeWarrior IDE.

Table 17.8 Browser menu key bindings

Command	Windows	Mac OS	Solaris and Linux
New Class			
New Member Function			
New Data Member			
New Property			
New Method			
New Event Set			
New Event			

Miscellaneous

[Table 17.9](#) contains the default key bindings for handling miscellaneous tasks in the CodeWarrior IDE.

Table 17.9 Miscellaneous key bindings

Command	Windows	Mac OS	Solaris and Linux
Quote Key			
Go to Header/ Source File	Ctrl-`	Command-`	Meta-`
Go to Previous Error Message	Shift-F4	Option-Command-Up Arrow	Alt-Meta-Up Arrow
Go to Next Error Message	F4	Option-Command-Down Arrow	Alt-Meta-Down Arrow
Run Script			
Stop Script			

Editor Commands

[Table 17.10](#) contains the default key bindings for handling Editor windows in the CodeWarrior IDE.

Table 17.10 Editor window key bindings

Command	Windows	Mac OS	Solaris and Linux
Move Character Left	Left Arrow	Left Arrow	Left Arrow
Move Character Right	Right Arrow	Right Arrow	Right Arrow
Move Word Left	Ctrl-Left Arrow	Option-Left Arrow	Alt-Left Arrow
Move Word Right	Ctrl-Right Arrow	Option-Right Arrow	Alt-Right Arrow
Move Sub-word Left		Control-Left Arrow	Control-Left Arrow

Command	Windows	Mac OS	Solaris and Linux
Move Sub-word Right		Control-Right Arrow	Control-Right Arrow
Move to Start of Line	Home	Command-Left Arrow	Meta-Left Arrow
Move to End of Line	End	Command-Right Arrow	Meta-Right Arrow
Move Line Up	Up Arrow	Up Arrow	Up Arrow
Move Line Down	Down Arrow	Down Arrow	Down Arrow
Move to Top of Page	Page Up	Option-Up Arrow	Alt-Up Arrow
Move to Bottom of Page	Page Down	Option-Down Arrow	Alt-Down Arrow
Move to Top of File	Ctrl-Home	Command-Up Arrow	Meta-Up Arrow
Move to Bottom of File	Ctrl-End	Command-Down Arrow	Meta-Down Arrow
Delete Character Left	Backspace	Delete	Delete
Delete Character Right	Del	del	del
Delete to End of File			
Character Select Left	Shift-Left Arrow	Shift-Left Arrow	Shift-Left Arrow
Character Select Right	Shift-Right Arrow	Shift-Right Arrow	Shift-Right Arrow
Select Word Left	Ctrl-Shift-Left Arrow	Option-Shift-Left Arrow	Alt-Shift-Left Arrow
Select Word Right	Ctrl-Shift-Right Arrow	Option-Shift-Right Arrow	Alt-Shift-Right Arrow

Command	Windows	Mac OS	Solaris and Linux
Select Sub-word Left		Control-Shift-Left Arrow	Control-Shift-Left Arrow
Select Sub-word Right		Control-Shift-Right Arrow	Control-Shift-Right Arrow
Select Line Up	Shift-Up Arrow	Shift-Up Arrow	Shift-Up Arrow
Select Line Down	Shift-Down Arrow	Shift-Down Arrow	Shift-Down Arrow
Select to Start of Line	Shift-Home	Shift-Command-Left Arrow	Shift-Meta-Left Arrow
Select to End of Line	Shift-End	Shift-Command-Right Arrow	Shift-Meta-Right Arrow
Select to Start of Page	Shift-Page Up	Option-Shift-Up Arrow	Alt-Shift-Up Arrow
Select to End of Page	Shift-Page Down	Option-Shift-Down Arrow	Alt-Shift-Down Arrow
Select to Start of File	Ctrl-Shift-Home	Shift-Command-Up Arrow	Shift-Meta-Up Arrow
Select to End of File	Ctrl-Shift-End	Shift-Command-Down Arrow	Shift-Meta-Down Arrow
Scroll Line Up	Ctrl-Up Arrow	Control-Up Arrow	Control-Up Arrow
Scroll Line Down	Ctrl-Down Arrow	Control-Down Arrow	Control-Down Arrow
Scroll Page Up		Page Up	Page Up
Scroll Page Down		Page Down	Page Down
Scroll to Top of File		Home	Home

Command	Windows	Mac OS	Solaris and Linux
Scroll to End of File		End	End
Scroll to Selection		Enter	
Find Symbols with Prefix	Ctrl-\	Control-\	Control-\
Find Symbols with Substring	Ctrl-Shift-\	Control-Shift-\	Control-Shift-\
Get Next Symbol	Ctrl-.	Control-.	Control-.
Get Previous Symbol		Control-,	Control-,

Catalog

[Table 17.11](#) contains the default key bindings for use with the CodeWarrior class browser.

Table 17.11 **Catalog key bindings**

Command	Windows	Mac OS	Solaris and Linux
New Catalog			
New Folder			
Open Catalog			
Toggle Index View			
Edit Item Properties			

Layout Menu

[Table 17.12](#) contains the default key bindings for use with the CodeWarrior RAD layout editor.

Table 17.12 Layout menu key bindings

Command	Windows	Mac OS	Solaris and Linux
Align			
Resize			
Display Grid			
Snap To Grid			
Group			
Ungroup			
Customize			
Properties			

Window Menu

[Table 17.13](#) contains the default key bindings for handling many common windows in the CodeWarrior IDE.

Table 17.13 Window menu key bindings

Command	Windows	Mac OS	Solaris and Linux
Stack Editor Window			
Tile Editor Window			
Tile Editor Windows Vertically			
Zoom Window	Ctrl-/	Command-/	Meta-/
Collapse/Expand Window			
Save Default Window			

CodeWarrior Reference*IDE Default Key Bindings*

Command	Windows	Mac OS	Solaris and Linux
Toolbar			
Browser Contents			
Class Hierarchy Window			
Single Class Hierarchy Window			
New Class Browser			
Build Progress Window			
Errors & Warnings Window		Command-I	Meta-I
Project Inspector			
ToolServer Worksheet			
Processes Window			
Expressions Window			
Global Variables Window			
Breakpoints Window			
Watchpoints Window			
Register Windows			

Command	Windows	Mac OS	Solaris and Linux
Component Catalog			
Component Palette			
Object Inspector			
Select Default Project	Ctrl-0	Command-0	Meta-0
Select Document 1	Ctrl-1	Command-1	Meta-1
Select Document 2	Ctrl-2	Command-2	Meta-2
Select Document 3	Ctrl-3	Command-3	Meta-3
Select Document 4	Ctrl-4	Command-4	Meta-4
Select Document 5	Ctrl-5	Command-5	Meta-5
Select Document 6	Ctrl-6	Command-6	Meta-6
Select Document 7	Ctrl-7	Command-7	Meta-7
Select Document 8	Ctrl-8	Command-8	Meta-8
Select Document 9	Ctrl-9	Command-9	Meta-9

Solaris and Linux Utilities



This appendix describes additional utilities provided in the Solaris and Linux versions of the CodeWarrior IDE. These utilities are accessible from the Info menu on the right-hand side of the menu bar in the IDE windows. This menu is only available in the Solaris and Linux versions of the CodeWarrior IDE.

This appendix discusses the following topics:

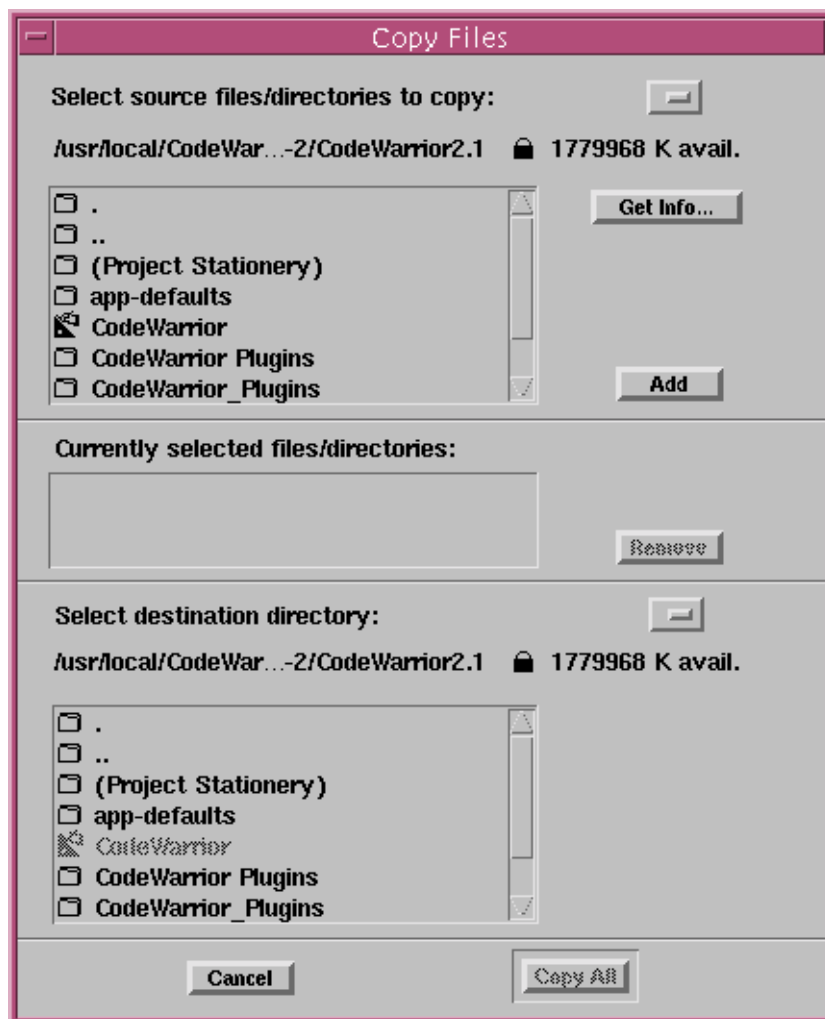
- [Copy Files Accessory](#)
- [File Systems Facility](#)
- [Keyboard Preferences Dialog Box](#)

Copy Files Accessory

When you choose the **Copy Files** menu command, CodeWarrior launches a desk accessory. This desk accessory lets you copy files and folders between the Solaris or Linux environment from within the CodeWarrior IDE. The environment includes such file systems and devices as Macintosh diskettes; AppleDouble; K-AShare; EtherShare and Partner file systems; and NFS-mounted UNIX file systems. For example, you can use the **Copy Files** command to copy a file from a Macintosh diskette to a UNIX directory.

To display the Copy Files dialog box ([Figure A.1](#)), choose **Copy Files** from the Info menu.

Figure A.1 The Copy Files dialog box



This dialog box contains three sections: [Source File/Directory Selection](#), [Currently Selected Files/Directories](#), [Destination Directory Selection](#)

Source File/Directory Selection

This area of the dialog box is used for selecting files to copy from the source volume. It contains the following items:

Path Display

The path display shows the path to the current directory being listed.

Free Space Display

This display shows how much disk space is available on the currently selected volume.

Source File/Directory List

This is a scrollable list of files in the current directory. Double-clicking a directory or volume in the list displays the contents of that directory or volume. Double-clicking a file adds it to the list of files to be copied. Only one item can be selected at a time.

Get Info Button

Pressing this button displays a dialog box that provides file information about the currently selected file or directory in the File/Directory List. If no file or directory is currently selected in the list, an alert message appears with instructions.

Add button

This buttons allows you to add the directory, volume, or file currently selected in the source directory to the list of files to be copied.

Currently Selected Files/Directories

This area of the dialog box is for viewing the files you have selected for copying. It contains the following items:

Selected Files/Directories List

This list displays the truncated path(s) of the file(s) to be copied. As entries are being copied, they are highlighted. After a file is copied, it is removed from the list. If an error occurs during the copying process, the copy is aborted and the list displays the names of files that have not yet been copied.

Remove button

Pressing this button removes the selected files/directories from the Source Files/Directories List.

Destination Directory Selection

This area of the dialog box is for setting the destination directory where files will be placed. It contains these items:

Destination Path Display

This display shows the path to the currently selected destination.

Free Space Display

This display shows how much disk space is available on the destination volume.

Destination File/Directory List

This is a scrollable list of files in the current directory. Double-clicking a directory or volume in the list displays the contents of that directory or volume.

Cancel Button

Pressing this button closes the dialog box without performing the copy operation.

Copy All Button

Pressing this button starts the copy procedure. You are first prompted by a notice to verify the destination directory. Once you verify this, the copy proceeds.

File Systems Facility

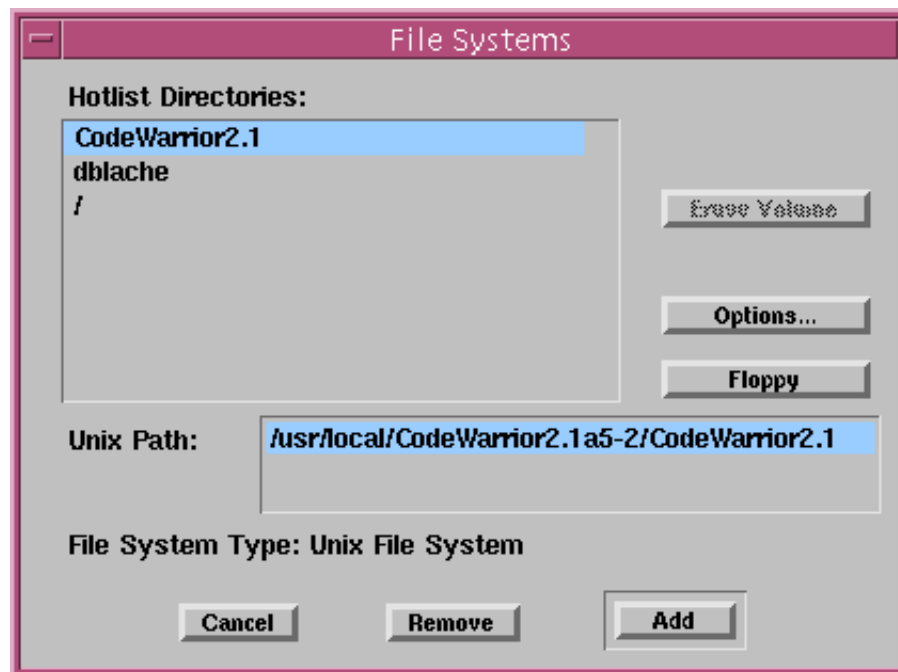
The Solaris and Linux versions of the CodeWarrior IDE provide the File Systems Facility, which enables you to mount volumes on your workstation. When the File Systems... menu item is chosen from the

Info menu, the File Systems Facility dialog box is displayed ([Figure A.2](#)).

The File Systems Dialog Box

The File Systems dialog box contains the following items: Hotlist Directories, Erase Volume Button, Options Button, Floppy Button, Unix Path Text Box, File System Type Display, Cancel Button, Remove Button, Add Button.

Figure A.2 The File Systems dialog box



Hotlist Directories

This list shows all of the volumes currently mounted on your system. You can select any volume listed and then unmount or erase it, or display the volume's full pathname and file system type, according to the options you choose below.

Erase Volume Button

Pressing this button deletes all files in the chosen volume.

Options Button

When this button is pressed, the Options dialog box is displayed. This dialog box lets you set line breaks in files you mount. See [“The File System Options Dialog Box” on page 684](#) for more information.

Floppy Button

Lists any diskette in the diskette drive (in the Unix Path field) and enables the Mount button.

Unix Path Text Box

This text box contains the full UNIX pathname of the chosen volume. If no volume is currently chosen, you can type a path into this area to mount a volume by its path.

File System Type Display

This display shows the type of format for the chosen volume—UNIX or HFS.

Cancel Button

Pressing this button closes the dialog box without mounting any file systems.

Remove/Eject Button

If the chosen volume is on a hard drive, this option unmounts it. If the chosen volume represents a diskette, this option ejects the diskette from the diskette drive.

Add Button

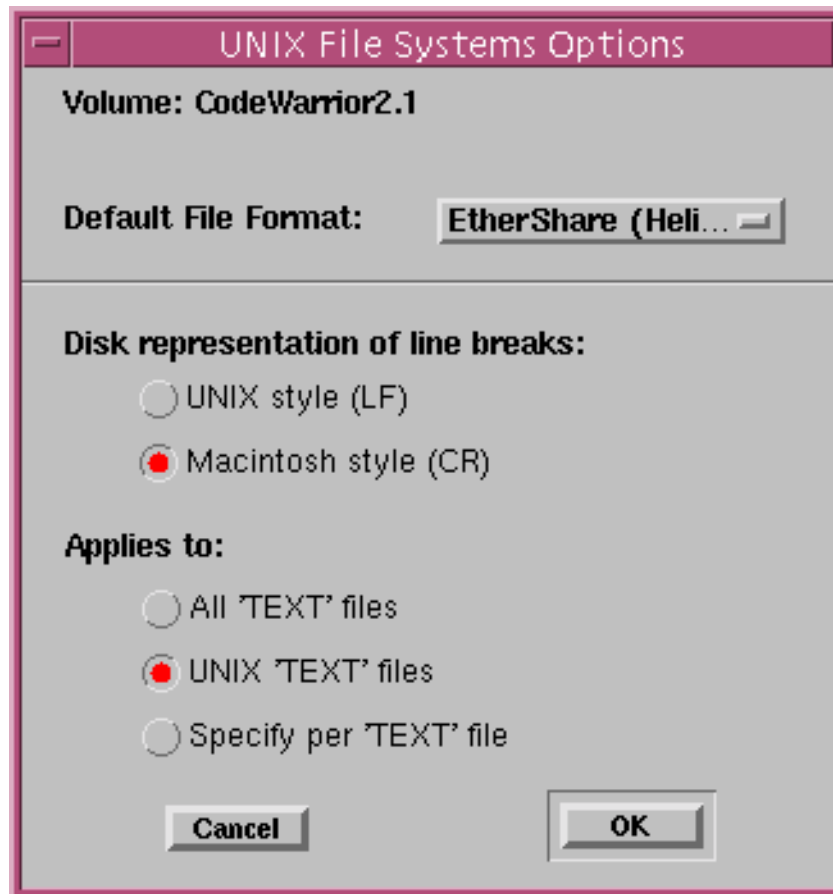
Pressing this button mounts the currently selected volume, and closes the dialog box.

The File System Options Dialog Box

Some applications may have difficulty representing line breaks appropriately as their files move from foreign operating systems to UNIX systems. The File System Options dialog box enables you to solve this problem by explicitly choosing how to represent line

breaks in certain types of files. To view this dialog box, press the Options button described above.

Figure A.3 The File System Options dialog box



This dialog box contains the following items: Volume Display, Default File Format, "Disk representation of line breaks" button group, "Applies to" button group, Cancel button, OK button.

Volume Display

This display shows the name of the volume which will be affected by the settings in this dialog box.

Default File Format

This popup menu lets you specify the file format for any new files that will be created on the volume in question.

Disk representation of line breaks

This group of buttons determines how the IDE will convert line break characters when it reads or writes files on the current volume.

UNIX Style (LF)

If this option is chosen, line breaks are replaced with line feeds whenever a file on the current volume is read or written by the CodeWarrior IDE.

Macintosh Style (CR)

If this option is chosen, line breaks will be replaced with carriage returns whenever a file on the current volume is read or written by the CodeWarrior IDE.

Applies to

This group of buttons determines which files will be affected by your settings in the File Systems dialog. Only one of these buttons may be selected at a time.

All 'TEXT' Files

If this option is selected, conversion will be applied to all text files read or written by the IDE on the current volume.

UNIX 'TEXT' Files

If this option is selected, conversion will be applied only to UNIX (non-Macintosh) text files read or written by the IDE on the current volume.

Specify per 'TEXT' Files

If this option is selected, a dialog box will be displayed each time a text file is read or written by the IDE, allowing you to specify how line breaks should be converted for that file.

Cancel button

Pressing this button dismisses the dialog box without applying any settings.

OK button

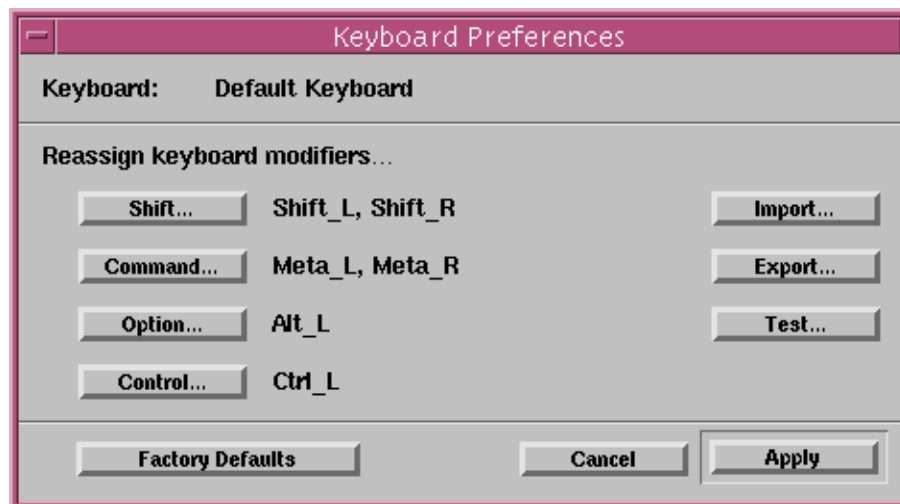
Pressing this button applies the selected settings and dismisses the dialog box.

Keyboard Preferences Dialog Box

Modifier keys are used in combination with other keys to generate keyboard shortcuts to certain IDE functions. For instance, to execute the Ctrl/Command - N keyboard shortcut, you depress the Ctrl or Command modifier key, and while holding it down, press the N key on your keyboard.

The Keyboard Preferences dialog box ([Figure A.4](#)) allows you to define which physical keys on your keyboard correspond to certain modifier keys used in the CodeWarrior IDE. To display the dialog box, choose **Keyboard Preferences** from the Info menu.

Figure A.4 The Keyboard Preferences dialog box



This dialog box contains the following items: Keyboard Display, Reassign Keyboard Modifiers Section, Import Button, Export Button, Test Button, Factory Defaults Button, Cancel Button, Apply Button.

Keyboard Display

This display shows the active keyboard.

Reassign Keyboard Modifiers Section

The Shift, Command, Option, and Control buttons in this section correspond to modifier keys used in the CodeWarrior IDE.

Pressing one of these buttons will cause CodeWarrior to scan the keyboard and record keys that are pressed until a key is released. The key(s) that are pressed during the scan will be mapped to the corresponding modifier key. Once scanning is complete, a list of keys that were pressed will appear to the right of the button.

Once the Apply button is pressed, the changes you have made go into effect. From this point on, when you type any of these keys, CodeWarrior will respond as if the corresponding modifier key has been pressed.

For example, to map both the left and right Shift keys on your keyboard to the Shift modifier key, follow these steps:

1. Press the Shift button in this dialog box.
2. Press and hold down both of the Shift keys on your keyboard.
3. Release the Shift keys on your keyboard.

Shift_L and Shift_R should now appear next to the Shift button in the dialog box.

Import Button

Press this button to display a file open dialog, which will allow you to select a file to import keyboard modifier settings from.

Export Button

Press this button to display a file save dialog which will allow you to export the current keyboard modifier settings to a file on disk.

Test Button

Press this button to test your settings.

Factory Defaults Button

Press this button to reset the settings to the factory defaults.

Cancel Button

Press this button to dismiss the dialog box without applying your changes.

Apply Button

Press this button to apply any changes you have made and dismiss the dialog box.

Perl Scripts



You can run a Perl script as a prefix file with CodeWarrior projects. To enable this functionality, you must first install additional software plug-ins. After you install these plug-ins, you can configure CodeWarrior to recognize the Perl script.

This chapter discusses the following topics:

- [Installing the Perl Software Plug-ins](#)
- [Configuring the Perl Target Settings Panel](#)
- [Perl Scripting](#)
- [Special Considerations](#)

Installing the Perl Software Plug-ins

The CodeWarrior IDE uses various software plug-ins to extend its functionality. Recognizing and implementing Perl scripts in the IDE requires various Perl plug-ins. The specific plug-ins depend on the host platform you are using. The following instructions describe how to properly install the Perl plug-ins for your particular host.

Windows

CodeWarrior implements Perl script functionality with the MWPerl.dll plug-in, the PerlDLL.dll plug-in, and the MWPerl Panel.dll plug-in. You must install all three of these plug-ins for the Perl scripts to work properly.

Place the MWPerl.dll and PerlDLL.dll plug-ins in the Compiler folder. This folder resides at the following location:

```
\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\Compiler\
```

Perl Scripts

Configuring the Perl Target Settings Panel

Place the `MWPerl Panel.dll` plug-in within the Preference Panel folder. This folder resides at the following location:

```
\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\Preference  
Panel
```

Mac OS

CodeWarrior implements Perl script functionality with the MW Perl plug-in, the Perl Panel plug-in, and the Perl extension. You must install all three of these components for the Perl scripts to work properly.

Place the MW Perl plug-in within the Compilers folder. This folder resides at the following location:

```
Metrowerks CodeWarrior:CodeWarrior Plugins:Compilers
```

Place the Perl Panel inside the Preference Panels folder. This folder resides at the following location:

```
Metrowerks CodeWarrior:CodeWarrior Plugins:Preference Panels
```

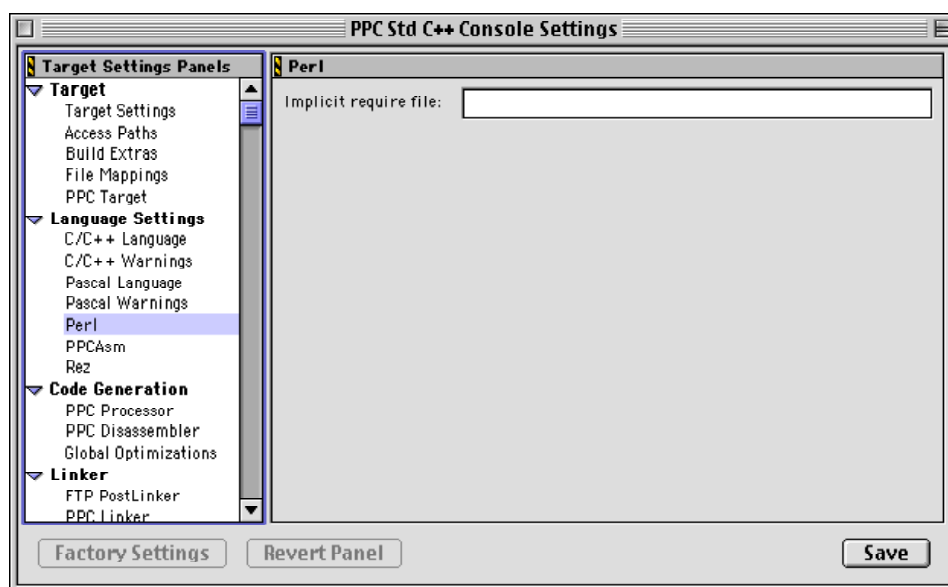
Finally, place the Perl extension inside the Extensions folder of the System Folder.

Configuring the Perl Target Settings Panel

After you install the software plug-ins, CodeWarrior allows you to specify a Perl script as a prefix file for your project. You use the Perl target settings panel to specify the Perl script. CodeWarrior treats this script as an implicit `require` file. The `require` directive is the Perl equivalent of the `#include` directive in C/C++.

To display the Perl settings panel, choose the **Target Settings** command from the Edit menu. The actual name of this command depends on your currently selected build target. The Target Settings window in [Figure B.1](#) displays.

Figure B.1 Perl Target Settings Panel



Scroll through the list on the left side of the Target Settings window and highlight the **Perl** item. The Perl settings panel displays on the right side of the Target Settings window, as shown in [Figure B.1](#).

To include a Perl script as a prefix file in the current project, type the file name for the Perl script in the **Implicit require file** text field. CodeWarrior treats this file as an implicit Perl `require` file when you build the project.

NOTE The Perl plug-in for CodeWarrior uses the find-and-load functionality of the IDE. This functionality depends on the ability of the IDE to find referenced files using absolute paths. You must specify the access paths for any files in the Perl script or the implicit require file that are not referenced via absolute paths. You specify this path information in the Access Paths settings panel. For more information, see [“Access Paths” on page 329](#)

Perl Scripting

The Perl script implementation takes advantage of the IDE’s plug-in application programming interface (API).

[Listing B.1](#) shows a sample Perl script:

Listing B.1 Perl Script Example

```
# Simple Perl Example

# Print a line of text
print "Hello World!\n";

$scale0 = 0;
$scale1 = 1;
$scale2 = 2.5;

# Create and open a file for output
open (theFile, ">output.txt");

# Dump some text into the file
print theFile "The file should now be open\n";
print theFile "Let's try a few things:\n\n";

# Arithmetic
print theFile "***Arithmetic \n";
print theFile $scale1 + $scale2 . "\n";
print theFile $scale1 * $scale2 . "\n";
print theFile $scale1 % $scale2 . "\n\n";

# Boolean logic
print theFile "***Boolean \n";
print theFile ($scale0 && $scale0) . "\n";
print theFile ($scale0 && $scale1) . "\n";
print theFile ($scale1 && $scale1) . "\n";

print theFile ($scale0 || $scale0) . "\n";
print theFile ($scale0 || $scale1) . "\n";

print theFile (!$scale0) . "\n\n";

# Comparison
print theFile "***Comparisons \n";
print theFile ($scale2 == $scale2) . "\n";

print "That's it, closing file\n";
```

```
# Close the file  
close theFile;
```

Special Considerations

When using CodeWarrior to recognize and implement Perl scripts in your project, you must keep in mind some special considerations. These considerations include:

- [StdIn Usage](#)
- [Avoiding Link Errors](#)

StdIn Usage

StdIn is not supported in the prefix file. This means that the Perl script cannot accept keyboard input.

Avoiding Link Errors

The CodeWarrior linker treats the Perl script as a prefix file. Therefore, the linker expects to apply that prefix file to some source code in your project. If your project does not contain source code, the linker will display the following error message when you compile the project:

```
Link Error: 'main' is undefined.
```

Compiling the project successfully runs the script but also produces the linker error. To avoid this error message, either ensure that your project contains some source code or set the **Linker** option to None in the Target Settings preference panel. For more information, see [“Target Settings” on page 327](#).

Index

Symbols

#include files 235
#pragma, adding markers by using 159
\$ in variable name 500
%file command-line string 260
%line command-line string 260
(Helper Apps) folder 261
.ctlg extension 559
.dump 379
.MAP extension 369
.mcp extension 56, 513
.mkb extension 310
? in variable name 500

A

about
 RAD 509–510
 the CodeWarrior IDE 17–19
About (VCS operation) 608
About Metrowerks command (Mac OS) 655
About Metrowerks command (Windows) 653
absbottom option, of Java Applet wizard 520
absmiddle option, of Java Applet wizard 520
Absolute Path 269, 334, 347
Absolute Path option
 of Source Trees preference panel 267
 of Source Trees settings panel 347
Absolute Path option, in Type pop-up menu 267, 347
Abstract option, of Specifier pop-up menu 529, 589, 595
Abstract Window Toolkit *See* AWT.
accelerating
 compiling and linking 371
access paths 329
 recursive search 330
Access Paths settings panel 329
 Add button 334
 Add Default button 333
 Always Search User Paths checkbox 332
 Change button 336
 Host Flags pop-up menu 333
 Interpret DOS And Unix paths checkbox (Mac OS) 332
 Remove button 336
 System Paths pane 333
 System Paths radio button (Windows) 332
 User Paths pane 333
 User Paths radio button (Windows) 332
Access pop-up menu
 of Java Bean wizard 529
 of New Class wizard 235
 of New Event Set wizard 595
 Private option 239, 242, 588, 595
 Protected option 239, 242, 588, 595
 Public option 239, 242, 529, 588, 595
Accessors section, of New Property dialog box 586
Action section
 of Customize IDE Commands window 300
actions
 defining for custom commands 302
 undoing and redoing a series of 276
Activate Browser checkbox
 in Build Extras settings panel 338
Activate Browser Coloring checkbox
 of Browser Display preference panel 273
activating 104
 breakpoints 423
 debugging for projects 104
active pane 402
Add (VCS operation) 608
Add button
 in Access Paths settings panel 334
 of Custom Keywords dialog box 281
 of File Mappings settings panel 342
 of New Event dialog box 599
 of New Method dialog box 589
 of New Property dialog box 586
 of Source Trees preference panel 268
 of Source Trees settings panel 347
Add button, of Java Applet wizard 522
Add Default button
 in Access Paths settings panel 333
Add Design To Project checkbox, of Project panel 532
Add File command 632

Index

- Add File To Targets list, of New Event Set wizard 596
- Add Files command 632
- Add Files To Targets list
 - of New Class wizard 237
- Add Window command 85, 374, 376, 632
- adding
 - designs to projects 531
 - elements to toolbars 318
 - file name extensions 342
 - files by using drag and drop 83
 - files to projects 78
 - files to projects (Macintosh) 81
 - files to projects (Solaris) 82
 - files to projects (Windows) 80
 - key bindings 308
 - keyboard shortcuts 308
 - markers 158
 - markers by using #pragma 159
 - menu commands 300
 - preprocessor symbols to projects 106–107
 - text in editor 151
- Additional Header Include Files
 - of New Class wizard 235
 - of New Data Member wizard 244
 - of New Member Function wizard 241
- Address field
 - of Register Details window 427
- adjusting
 - tab size 279
 - time allotted to balancing punctuation 277
 - time allotted to display contextual menu 277
- advanced options
 - for compiling 371
- After option, of New Class wizard 233
- After option, of New Event Set wizard 594
- Alert Yourself After Build option (Mac OS) 371
- Align command 543
- Align pop-up menu, of Java Applet wizard 520
- Align To Grid command 651
- All Exceptions command 653
- All Info option
 - of Plugin Diagnostics 264
- All option
 - in Host Flags pop-up menu 333
- Alpha 90
- alphabetizing
 - C++ methods 285
- Alt/Option 23
- Altivec Registers command 433
- Altivec Registers window 433
- Always Search User Paths checkbox
 - of Access Paths settings panel 332
- Always Show Login Dialog (VCS option) 605, 609
- Always Use File Mapping For Symbolics checkbox
 - of MetroNub Settings preference panel (Mac OS) 294
- analyzing
 - inheritance in browser 250
- Ancestor Class pop-up menu 226
- Anchor Floating Toolbar command 320
- Anchor Floating Toolbar command (Mac OS) 655
- anchoring
 - floating toolbar 320
- ANSI
 - C code in Pascal project 503
- Appears in Menus checkbox
 - of Customize IDE Commands window 299, 301
- Apple Menu
 - About Metrowerks command (Mac OS) 655
- Apple Menu (Mac OS) 655
- AppleScript 261, 262
- Applet Class section, of Java Applet wizard 516
- Applet Name field, of Java Applet wizard 519
- Applet Parameters section, of Java Applet wizard 521
- Applet Title field, of Java Applet wizard 523
- applets
 - additional information for 522, 526
 - creating parameters for 521
 - specifying HTML page for 518
- Application Class section, of Java Application wizard 523
- Application Title field, of Java Application wizard 526
- applications
 - describing frame class for 525
- Arguments field
 - of Customize IDE Commands window (Windows) 302
- Arithmetic Optimizations 354
- Array window 436
 - setting base address 436
- arrays

- in separate windows 471
- arrow keys 43
- assembler 34
- assembly language
 - memory display 406, 408
 - register display 406, 408
 - viewing 396, 407, 415
- assigning
 - files to build targets 98
 - Quote Key prefix 312
- assigning files
 - with the Project Inspector 99
 - with the Target column 98
- assigning files with 98, 99
- Attempt To Use Dynamic Type Of C++, Object Pascal And SOM Objects checkbox
 - of Display Settings preference panel 285
- Auto Indent checkbox
 - of Font & Tabs preference panel 279
- Auto option
 - of Text View pop-up menu 429
- Auto Repeat checkbox
 - of Customize IDE Commands window 309
- Auto Target Libraries checkbox
 - of Global Settings preference panel 290
- auto-completing
 - names of symbols 247
- automatic
 - launch of applications when opening SYM files 290
- automatic punctuation balancing 155
- Automatic updating 373
- automatic updating
 - of precompiled headers 373
- automatically
 - indenting lines of text 279
- automatically editing symbols 248
- automatically entering symbols 248
- Automatically Launch Applications When SYM File Opened checkbox
 - of Global Settings preference panel 290
- automation
 - with RAD tools 509
- Auto-target libraries checkbox
 - of Debugger Settings panel 358
- Auto-variables view
 - in debugger 26

- AWT
 - RAD components 569–570
- AWT Applet radio button
 - of Java Applet wizard 518
- AWT Frame radio button
 - of Java Application wizard 526
 - of Java Frame wizard 546

B

- Back button
 - of Browser wizards 230
 - of RAD wizards 515, 544, 591
- Background Color setting 275
- Backing up files 120
- Backspace/Delete 22
- backup files 73, 120
- Balance command 155, 625
- Balance While Typing checkbox
 - of Editor Settings preference panel 275
- balancing punctuation 154
 - adjusting time allotted to 277
- balancing punctuation automatically 155
- Balloon Help (Mac OS) 262
- Base Class and Interfaces section
 - of Java Bean wizard 529
 - of New Event Set wizard 595
- Base Class section
 - of Java Bean wizard 530
 - of New Event Set wizard 596
- Base Classes and Methods section, of New Class wizard 234
- Base Classes field
 - of New Class wizard 234
- baseline option, of Java Applet wizard 520
- basic debugging 439–481
- basic navigation concepts 149
- Batch search 187
- BEdit 90
- BEdit Extensions Menu (Mac OS) 658
- beans
 - setting base class and interfaces of 529
 - setting name and location of 527
- Before option, of New Class wizard 233
- Before option, of New Event Set wizard 594
- beginners conventions 21
- binaries
 - removing 369, 370

Index

- binary
 - viewing data as 473
- Bit Value field
 - of Register Details window 427
- Bit Value Modifier pop-up menu
 - of Register Details window 428
- bit values
 - determining meaning of 428
- Bitfield Description option
 - of Text View pop-up menu 429
- Bitfield Name pop-up menu
 - None option 427
 - of Register Details window 427
- bits
 - changing register bit values 428
- blocks
 - shifting left and right 155
- bottom option, of Java Applet wizard 520
- Branch Optimizations 354
- Break on C++ Exception command 642
- Break on Java Exceptions command 642
- breakpoints
 - conditional 423, 461
 - conditional expressions 483
 - conditional, and loops 462
 - conditional, creating 462
 - defined 457
 - effect of temporary breakpoints on 460
 - impact of optimizing code 463
 - making active 423
 - making inactive 423
 - missing 497
 - no response from 497
 - setting 458, 497
 - setting and clearing 458
 - setting for redefined functions 462
 - setting for templated functions 462
 - setting in Source Code Browser window 415
 - temporary 449, 460
 - troubleshooting 496
 - viewing 460
- Breakpoints window 423, 460
- Breakpoints Window command 423, 460, 649
- Bring To Front command 544, 650
- Bring Up To Date command 257, 365, 366, 367, 371, 373, 634
- Browse button
 - in Build Extras settings panel (Windows) 339
 - of Register Details window 426
- Browser
 - assigning build targets to classes 236
 - specifying `#include` files for 235
 - specifying base classes and methods for classes 234
 - specifying file locations for member functions 239, 243
- browser
 - activating 207, 338
 - analyzing inheritance 250
 - automatic completion of symbols 248
 - base classes in hierarchy 226
 - Class Declaration button 221
 - Classes pane 220
 - Classes Pane button 221
 - Contents view 209, 215
 - controlling lines in hierarchy window 224
 - customizing windows 251
 - Data Members pane 222
 - describing name and location of new classes 231
 - displaying tab views (RAD) 580
 - editing code 250
 - Events tab view(RAD) 583
 - finding function overrides 250
 - Hierarchy view 211
 - identifier icon 222
 - identifying symbols in database 245
 - including subprojects 338
 - interface 212–230
 - List button 220
 - Member Functions pane 222
 - Methods tab view(RAD) 582
 - multi-class hierarchy 224–226
 - navigating code with 213, 246
 - Open File icon 223
 - opening a source file 248
 - Properties tab view(RAD) 581
 - RAD dialog boxes 584–600
 - RAD features 579–583
 - resize bar 219
 - saving windows 251
 - seeing a declaration 249
 - seeing a routine definition 249
 - seeing MFC classes in 251
 - seeing PowerPlant classes in 251
 - setting options 245
 - showing subclasses in hierarchy 225

-
- single-class hierarchy 226
 - Source pane 223
 - status area 224
 - strategy 208–212
 - Symbol window 227
 - synchronized class selection 220, 224
 - synchronized data member selection 222
 - synchronized member function selection 222
 - tab control (RAD) 579
 - tab view limitations 580
 - toolbar 218
 - using 244–252
 - view 210
 - viewing member functions and data members 218
 - viewing options 208
 - Browser Access Filters pop-up menu 218
 - Browser Contents command 647
 - Browser Display preference panel 272
 - Activate Browser Coloring checkbox 273
 - Include Insert Template Commands In Context Menu checkbox 273
 - Browser Menu 644–645
 - New Class command 645
 - New Data Member command 645
 - New Event command 645
 - New Event Set command 645
 - New Member Function command 645
 - New Method command 645
 - New Property command 645
 - Browser menu 229
 - Browser window 216–224
 - pane zoom box 219
 - browser window
 - navigating code in 454
 - Browser wizards 230–244
 - Back button 230
 - Cancel button 230
 - Finish button 230
 - Generate button 231
 - Next button 230
 - browsing
 - across subprojects 247
 - source code 36
 - Build Before Running pop-up menu
 - of Build Settings preference panel 256
 - Build Extras settings panel 337
 - Activate Browser checkbox 338
 - Browse button 339
 - Cache Subprojects checkbox 338
 - Dump Internal Browse Information After Compile checkbox 338
 - Use Modification Date Caching checkbox 337
 - Use Third Party Debugger checkbox (Windows) 339
 - Build Progress Window command 648
 - Build Settings preference panel 256
 - Build Before Running pop-up menu 256
 - Compiler Thread Stack field 258
 - Failure pop-up menu 259
 - Include File Cache field 258
 - Play Sound After ‘Bring Up To Date’ & ‘Make’ checkbox (Mac OS) 258
 - Save Open Files Before Build checkbox 257
 - Show Message After Building Up-To-Date Project checkbox 257
 - Success pop-up menu 259
 - build target
 - preparing for debugging 397
 - build target configurations, in Target Settings window 326
 - build target settings 321–324
 - guided tour 321–324
 - build targets 35, 41
 - assigning files to 98
 - changing settings 96
 - changing the name of 95
 - choosing settings 324–360
 - creating 94
 - creating dependencies 97
 - defined 18, 91
 - setting 96
 - build, alert when completed 371
 - building projects 59
 - Button component 569
 - buttons
 - Add 334
 - Add Default 333
 - Cancel 255, 297, 323, 336
 - Change 336
 - Choose 302
 - Close Catalog 548, 552
 - Component Catalog 548
 - Component Palette 552
 - Delete 304
 - Don’t Save 255, 297, 323
-

Index

- Edit Item Properties 552
 - Export 310
 - Factory Settings 255, 297, 323
 - Import 310
 - Import Components 552
 - List View 553
 - Live View 553
 - New Catalog 551
 - New Command 301
 - New Folder 551
 - New Group 301
 - OK 336
 - Open Catalog 548, 552
 - Revert Panel 255, 298, 324
 - Save 255, 297, 298, 323, 324
 - Sort Order 48
 - Toggle Index View 552
- C**
- C popup parsing
 - relaxing in K&R-styled code 276
 - C string
 - entering data as 475
 - viewing data as 473
 - C String command 644
 - C++
 - debugging 290
 - methods, alphabetizing 285, 413
 - C/C++ Compiler Reference command 652
 - Cache Edited Files Between Debug Sessions checkbox
 - of Global Settings preference panel 289
 - Cache Subprojects checkbox
 - of Build Extras settings panel 338
 - Cache Symbolics Between Runs checkbox
 - of Debugger Settings panel 359
 - call-chain navigation 453
 - call-chain navigation through code 453
 - Can Run As A Standalone Application checkbox
 - of Java Applet wizard 518
 - Cancel button 255, 297, 323, 336
 - of Browser wizards 230
 - of Makefile Importer wizard 66
 - of New Event dialog box 599
 - of New Method dialog box 589
 - of New Property dialog box
 - New Property dialog box
 - Cancel button 586
 - of RAD wizards 515, 544, 591
 - Canvas component
 - components
 - Canvas 569
 - Catalog pane
 - of Component Catalog window 552
 - Catalog pop-up menu 548
 - catalogs
 - defined 511
 - Category pop-up menu, in Contents window 216
 - CFM68K 393
 - Change button
 - in Access Paths settings panel 336
 - of Source Trees preference panel 271
 - of Source Trees settings panel 350
 - Change Program Counter command 450
 - Change Program Counter dialog box
 - and skipping statements 450
 - changing
 - background color 275
 - build target names 95
 - build target settings 96
 - data in debugger 469
 - environment variables 341
 - font and color 457
 - main text color 274
 - memory 439
 - memory, dangers of 439
 - register bit values 428
 - registers 430, 431, 432
 - syntax highlighting colors 280
 - the current build target 96
 - variable values 475
 - character
 - viewing data as 473
 - Character command 644
 - character constant
 - entering data as 475
 - Check Syntax command 633
 - Checkbox components 569
 - Checkin 606
 - Checkin (VCS operation) 608
 - Checkout 606
 - Checkout (VCS operation) 608
 - Checkout Status Column 610
 - Checkout Status column 50, 612
 - Choice component 570

-
- Choose button 302
 - in Debugger Settings panel 357
 - in Runtime Settings panel
 - Runtime Settings panel
 - Choose button 340
 - in Target Settings panel 329
 - of Java Bean wizard 529
 - of Source Trees preference panel 268
 - of Source Trees settings panel 347
 - choosing
 - a compiler 362
 - a default project 74-75
 - class authoring, in RAD 510
 - Class Declaration button, in browser 221
 - Class field, of New Class wizard 232
 - Class field, of New Event Set wizard 594
 - Class For Debugging field
 - of Java Settings preference panel (Windows) 293
 - Class Hierarchy Window command 647
 - Class Is A Bean checkbox
 - of Java Bean wizard 528
 - of New Event Set wizard 592
 - Class item
 - in Component Information Bar 556
 - Class Name field
 - of Java Applet wizard 517
 - of Java Application wizard 523, 525
 - of Java Bean wizard 528
 - of Java Frame wizard 545
 - of New Class wizard 231
 - of New Event Set wizard 592
 - classes
 - assigning to build targets 236
 - auto-completion of names 247
 - describing name and location of 231
 - specifying #include files for 235
 - specifying base classes and methods for 234
 - Classes pane
 - and items not displayed in 220
 - Classes Pane button, in browser 221
 - Classes pane, in browser 220
 - Clear All Breakpoints command 459, 640
 - Clear All Watchpoints command 642
 - Clear Breakpoint command 459, 639
 - Clear button
 - in Runtime Settings panel 340
 - in Target Settings panel 329
 - Clear command 154, 468, 625
 - Clear command (Mac OS) 541
 - Clear command (Mac OS, Solaris, Linux) 421, 424
 - Clear Floating Toolbar command 319, 655
 - Clear Main Toolbar command (Windows) 319
 - Clear Watchpoint command 424, 467, 641
 - Clear Window Toolbar command 319, 654
 - ClearCase VCS plugin 602
 - clearing
 - breakpoints 458
 - watchpoints 424, 466
 - Close All command 620
 - Close All Variable Windows command 422
 - Close Catalog button 548, 552
 - Close Catalog command 621
 - Close command 620
 - commands
 - Close 557
 - Close Non-Debugging Windows radio button
 - of Windowing preference panel 287
 - closing
 - projects 74
 - Closing all files 125
 - Closing one file 123
 - code
 - compiling and linking 37
 - deadstripping 46
 - debugging and refining 34
 - editing and browsing 36
 - editing in browser 250
 - killing execution 451
 - navigating 452
 - navigating in the browser 246
 - navigation via Source Code Browser
 - window 454
 - preprocessing 378
 - restarting execution 452
 - running 443
 - running in debugger 445
 - selecting
 - font and color 457
 - source code file 32
 - stepping into routines 447
 - stepping out of routines 447
 - stepping through 443
 - stepping through a single line 446
 - stopping 443
 - stopping execution 450
-

Index

- viewing as assembly language 396
- viewing source as assembly language 407
- viewing source as mixed 409
- Code column 45
- code generation configurations, in Target Settings window 352
- code navigation contextual menu 213
- code optimization
 - and impact on breakpoints 463
- Codebase/Archive field, of Java Applet wizard 519
- CodeWarrior
 - and VCS 38
 - available tools 18
 - browser 207
 - browser (RAD) 579
 - configuring options 253
 - configuring target options 321
 - converting makefiles to projects 65–68
 - converting multiple IDE 1.7 projects to latest version 71
 - converting single IDE 1.7 project to latest version 71
 - customizing 38
 - debugging source code 395
 - getting started 29
 - IDE 31
 - IDE, defined 17
 - installation 31
 - introduction 17, 35–39
 - Linux system requirements 31
 - Mac system requirements 30
 - menu reference 617–660
 - online documentation 162
 - programming concepts 31–35
 - RAD components 569
 - RAD designs and layouts 509
 - RAD Layout editor 537
 - RAD tools defined 37
 - reference information 617
 - scripting 38
 - Scripts menu (Mac OS) 657
 - software generation 34
 - Solaris system requirements 31
 - system requirements 29–31
 - Windows system requirements 30
 - working with projects 41
- CodeWarrior Help command 652
- CodeWarrior IDE
 - about 17
 - CodeWarrior QuickStart Guide* 27
 - CodeWarrior Relative 336
 - Collapse Non-Debugging Windows radio button
 - of Windowing preference panel (Mac OS) 287
 - Collapse Window command (Mac OS) 646
 - collapsing
 - groups 75
 - color
 - for syntax 157
 - setting for background 275
 - setting for main text 274
 - Color Settings
 - of Editor Settings preference panel 274
 - Color Syntax option
 - and printing 126
 - coloring
 - comments 280
 - custom keywords 280, 281
 - keywords 280
 - strings 280
 - colors
 - changing for syntax highlighting 280
- COM 265
- command groups
 - creating 300
 - deleting 304
- Commando command 656
- commands
 - Add Window 85, 374, 376
 - adding to menu bar 300
 - Align 543
 - Altivec Registers 433
 - Anchor Floating Toolbar 320
 - Balance 155
 - Breakpoints Window 423, 460
 - Bring To Front 544
 - Bring Up To Date 257, 365, 366, 367, 373
 - C/C++ Compiler Reference 652
 - Change Program Counter 450
 - Clear 468
 - Clear (Mac OS) 541
 - Clear (Mac OS, Solaris, Linux) 421, 424
 - Clear All Breakpoints 459
 - Clear Breakpoint 459
 - Clear Floating Toolbar 319
 - Clear Main Toolbar (Windows) 319
 - Clear Watchpoint 424, 467

-
- Clear Window Toolbar 319
 - Close All Variable Windows 422
 - CodeWarrior Help 652
 - Commands & Key Bindings 296
 - Compile 257, 365
 - Component Catalog 550
 - Component Palette 547
 - Copy 435
 - Copy to Expression 420
 - creating 301
 - creating groups for 300
 - Customize Object 567
 - customizing 298
 - Debug 340, 366, 368
 - Debugger 652
 - defining actions for 302
 - Delete 567
 - Delete (Windows) 421, 424, 541
 - deleting 304
 - Disable Breakpoint 459
 - Disable Debugger 367, 397
 - Disassemble 257, 379
 - Display Grid 544
 - Enable Breakpoint 459
 - Enable Debugger 367, 397
 - Error Reference 652
 - examining 299
 - Export Project 103
 - exporting 310
 - Expressions Window 420, 477
 - Find Definition 163
 - Find Definition & Reference 164
 - Find Reference 164
 - FPU Registers 431
 - FPU Registers Window 431
 - General Registers 430, 480
 - Global Variables Window 470
 - Glossary 652
 - Go Back 161
 - Go Forward 161
 - Group 544
 - Hide Breakpoints 460
 - Hide Floating Toolbar 316
 - Hide Main Toolbar (Windows) 316
 - Hide Window Toolbar 317, 403
 - How To 652
 - IDE 652
 - Import Project 103
 - importing 310
 - Insert Template (in contextual menu) 273
 - Kill 451
 - Lock 558
 - Make 257, 366, 373, 400
 - modifying existing 299
 - MSL C Reference 652
 - MSL C++ Reference 652
 - New 53, 511, 531, 538, 558
 - New Class 229
 - New Event 597
 - New Event Set 589
 - New Folder 557
 - New Member Function 230
 - New Method 586
 - New Property 584
 - New Text File 373, 375, 377
 - Object Inspector 562
 - Open 69
 - Open Recent 69, 259
 - Other 652
 - Precompile 257, 371, 373, 374, 376, 378
 - Preferences 253
 - Preprocess 257, 379
 - Processes Window 417
 - Properties 544
 - Register Details Window 424
 - Register Window 430
 - Remove Files 88
 - Remove Object Code 370
 - Remove Toolbar Item 319
 - removing from menu bar 303
 - Rename 558
 - Reset Floating Toolbar 320
 - Reset Main Toolbar (Windows) 320
 - Reset Window Toolbar 320
 - Resize 543
 - Revert 157
 - Run 257, 366, 367, 400, 445, 450
 - Save 376, 378
 - Save A Copy As 73
 - Save As 435
 - Select All 567
 - Send To Back 544
 - Set Watchpoint 466
 - Set/Clear Breakpoint At 458
 - Shift Left 155
 - Shift Right 155
 - Show Breakpoints 459
 - Show Floating Toolbar 316
-

Index

- Show Main Toolbar (Windows) 316
- Show Private 219
- Show Protected 219
- Show Public 219
- Show Types 472, 500
- Show Window Toolbar 317, 403
- Snap To Grid 544
- Step Into 447
- Step Out 447
- Step Over 446
- Stop 445, 450
- Synchronize Modification Dates 369
- Syntax coloring 280
- Target Settings 375, 377, 378
- Touch 89
- Unanchor Floating Toolbar 320
- Ungroup 544
- Unlock 558
- View 557
- View Alphabetical 567
- View Array 422, 436
- View As Implementor 219
- View As Subclass 219
- View As User 219
- View In Groups 567
- View Memory 478
- View Memory As 436, 438, 478
- View Variable 422
- Watchpoints Window 424, 468
- Commands & Key Bindings command 296, 626
- Commands tab
 - of Customize IDE Commands window 298
- Comment (VCS operation) 608
- comments
 - coloring 280
- Comments item
 - in Component Information Bar 556
- Common Object Model. *See* COM. 265
- Common Subexpression Elimination 354
- Compare Files command 631
- Comparing and Merging Files & Folders 127
- Compile command 257, 365, 634
- compiler 34
- Compiler pop-up menu
 - of File Mappings settings panel 345
- Compiler Relative Path 269, 335, 348
- Compiler Thread Stack field
 - of Build Settings preference panel 258
- compiling 361–391
 - one file 365
 - projects 363–371
 - selected files 365
 - source files 365
- compiling and linking
 - choosing a compiler 362
 - compiling files 364
 - debugging 368
 - disassembling 379
 - guided tour 379–382
 - link map 369
 - making a project 366
 - options 371
 - plugin compilers 362
 - precompiling headers 371–378
 - preprocessing 378
 - removing binaries 369, 370
 - removing object code 370
 - running 367
 - setting file extension 362
 - speeding 371
 - synchronizing modification dates 369
- compiling code 37
- completing symbols 247
- completing symbols automatically 248
- complex projects
 - creation strategy 94
 - working with 91–99
- Component Catalog button 548
- Component Catalog command 550, 649
- Component Catalog contextual menu 557
- Component Catalog File option, of New window 559
- Component Catalog toolbar 551
- Component Catalog window 550–562
 - Catalog pane 552
 - Component Information Bar 555
 - Component list 556
 - Component pane 553
 - Content View buttons 553
 - creating catalogs 558
- component editors 572–577
- Component Information Bar 555, 557
 - Class item 556
 - Comments item 556
 - Lock item 556
 - Modified item 556

-
- Name item 556
 - resizing columns 557
 - sort order 557
 - sorting 557
 - Component list
 - in Component Catalog window 556
 - component model, in RAD 510
 - Component Palette 546–550
 - Catalog pop-up menu 548
 - component tools 548
 - toolbar 548
 - Component Palette button 552
 - Component Palette command 547, 649
 - Component pane
 - and Component Information Bar 555
 - of Component Catalog window 553
 - Component pane views, contrasted 554
 - component tools 548
 - components 569
 - Button 569
 - Checkbox 569
 - Choice 570
 - defined 510
 - Jbutton 571
 - JCheckBox 571
 - JComboBox 571
 - JEditorPane 571
 - JLabel 571
 - JList 571
 - JMenuBar 571
 - JOptionPane 571
 - JPanel 571
 - JPasswordField 571
 - JPopupMenu 571
 - JProgressBar 571
 - JRadioButton 571
 - JScrollBar 572
 - JScrollPane 572
 - JSlider 572
 - JSplitPane 572
 - JTabbedPane 572
 - JTable 572
 - JTextArea 572
 - JTextField 572
 - JTextPane 572
 - JToggleButton 572
 - JToolBar 572
 - JTree 572
 - List 570
 - MenuBar 570
 - Panel 570
 - PopupMenu 570
 - Scrollbar 570
 - ScrollPane 570
 - TextArea 570
 - TextField 570
 - concepts
 - programming 31–35
 - conditional breakpoints 423, 461
 - and loops 462
 - creating 462
 - expressions and 483
 - configurations
 - for build targets 326
 - for code generation 352
 - for debugger 357
 - for editor 356
 - configuring Makefile Importer wizard options 66
 - configuring options 253
 - configuring target options 321
 - Confirm “Kill Process” When Closing Or Quitting checkbox
 - of Global Settings preference panel 290
 - Confirm Invalid File Modification Dates When Debugging checkbox
 - of Global Settings preference panel 289
 - confirming
 - process kills when closing or quitting 290
 - Connect (VCS operation) 608
 - Connect on Open (VCS option) 605
 - consequences
 - of skipping statements 448
 - Const checkbox
 - of New Data Member wizard 242
 - of New Member Function wizard 239
 - constants
 - auto-completion of names 247
 - Constructor Parameters field, of New Class wizard 235
 - Content View buttons, Component Catalog window 553
 - contents
 - of register 424
 - of symbolics files 396–397
 - Contents view 209
 - Contents window
-

Index

- browser 215
- Context Popup Delay field
 - of Editor Settings preference panel 277
- contextual menu
 - for debugger 416
 - in browser 213
 - in Component Catalog window 557
 - in Layout editor 543
 - in Object Inspector 566
- contextual menus
 - adjusting time allotted to display 277
- controlling
 - color 157
 - project debugging 104–106
 - syntax highlighting within a window 280
 - time allotted to balancing punctuation 277
 - time allotted to display contextual menu 277
- controls
 - Pane Resize boxes 148
 - Pane Splitter 147
- conventions 20
 - figures 22
 - for beginners 21
 - host terminology 21
 - hypertext 21
 - keyboard shortcuts 22
 - modifier keys 22
 - notes 20
 - tips 21
 - typefaces 21
 - typographical 20
 - warnings 20
- converting
 - from IDE 1.7 project format 71
- Copy And Expression Propagation 354
- Copy command 154, 435, 624
- Copy Files... 660
- Copy Propagation 354
- Copy to Expression command 420, 643
- Copy to Expression command (debugger) 477
- crashing
 - debugger 496
- Create A Frame On Startup checkbox
 - of Java Application wizard 524
- Create Folder checkbox 513
- Create New Group command 632
- Create New Segment command (Mac OS) 633
- Create New Target command 633
- creating
 - build target dependencies 97
 - build targets 94
 - conditional breakpoints 462
 - environment variables 340
 - groups 87
 - input files 32
 - layouts 537
 - menu command groups 300
 - menu commands 301
 - menu items in Menu editor 574
 - panes in editor window 147
 - precompiled headers 372
 - project stationery 61
 - projects 52–59
 - projects from stationery 54
 - RAD component catalogs 558
 - RAD objects in layouts 540
 - RAD projects 511–515
 - subprojects within projects 99
- .ctlg extension 559
- Ctrl/Command 23
- Current Target pop-up menu 533
- current-statement arrow 406
 - and skipping statements 448
 - and Source Code Browser window 415
 - at breakpoint 457
 - defined 444
 - dragging 448
- custom commands
 - defining actions for 302
- custom keywords
 - coloring 280, 281
 - deleting 282
 - exporting 282
 - importing 282
- Custom Keywords dialog box
 - Add button 281
 - Custom Keywords list 281
 - Done button 282
 - Edit button 281
- Custom Keywords list
 - of Custom Keywords dialog box 281
- Custom Keywords settings panel 356
 - Edit button 357
- Custom option, of New Event Set wizard 596
- Customize IDE Commands window
 - Action section 300

-
- Appears in Menus checkbox 299, 301
 - Arguments field (Windows) 302
 - Auto Repeat checkbox 309
 - Commands tab 298
 - dialog box buttons 296
 - Directory field (Windows) 302
 - discarding changes 297
 - Dvorak KCHR Support checkbox 307, 308, 312
 - Execute field (Windows) 302
 - Key Bindings section 300
 - Name field 299, 301
 - Numeric Keypad Bindings checkbox 306, 308, 312
 - Prefix Key Timeout field 312
 - Run App/Script field 302
 - Customize Object command 567
 - customizing
 - commands 298
 - IDE 38, 296–320
 - key bindings 304
 - menu commands 300, 303
 - toolbars 313
 - Cut command 154, 624
 - CVS VCS plugin 602
 - CW Core Tutorials folder 27
- D**
- data
 - in separate windows 471
 - viewing and changing in debugger 469
 - viewing as different types 474
 - viewing in different formats 473
 - Data column 46
 - data formats
 - availability 474
 - for variables 475
 - Data Member Declaration section, of New Data Member wizard 241
 - data members
 - declaring 241
 - viewing in browser 218
 - Data Members pane, in browser 222
 - Data Menu 642–644
 - C String command 644
 - Character command 644
 - Copy to Expression command 643
 - Default command 643
 - Enumeration command 644
 - Fixed command (Mac OS) 644
 - Floating Point command 644
 - Fract command (Mac OS) 644
 - Hexadecimal command 644
 - New Expression command 643
 - Pascal String command 644
 - Show Types command 642
 - Signed Decimal command 643
 - Unicode String command 644
 - Unsigned Decimal command 643
 - View Array command 643
 - View As command 643
 - View Variable command 643
 - data types
 - anon 500
 - enumerated 500
 - multiple 477
 - showing 472
 - viewing 472
 - viewing structured 406
 - Data Update Interval
 - in Debugger Settings panel 359
 - Database Path (VCS option) 605
 - dates
 - synchronizing 369
 - synchronizing modification dates 90
 - Dead Code Elimination 354
 - Dead Store Elimination 355
 - deadstripping code 46
 - Debug All Java Class Files In Directory Hierarchy checkbox
 - of Global Settings preference panel 289
 - Debug column 46, 104
 - debug column in Project window 399
 - Debug command 340, 366, 368, 400, 494
 - Debug Info marker 104
 - for groups 106
 - Debug Menu 638–642
 - Break on C++ Exception command 642
 - Break on Java Exceptions command 642
 - Clear All Watchpoints command 642
 - Clear Breakpoint command 639
 - Clear Watchpoint command 641
 - Disable Breakpoint command 640
 - Disable Watchpoint command 641
 - Enable Breakpoint command 640
 - Enable Watchpoint command 641
 - Hard Reset command (Embedded) 638
-

Index

- Hide Breakpoints command 640
- Kill command 638
- Reset command (Embedded) 638
- Restart command 638
- Set Breakpoint command 639
- Set Watchpoint command 641
- Show Breakpoints command 640
- Step Into command 639
- Step Out command 639
- Step Over command 638
- Stop command 639
- Switch To Monitor command 642
- View Memory As command 643
- View Memory command 643
- Debugger
 - Display Settings preference panel 283
 - Global Settings preference panel 288
 - Java Debugging preference panel 291
 - Java Settings preference panel 292
 - MetroNub Settings preference panel 293
 - Windowing preference panel 286
 - x86 Settings preference panel 295
- debugger 34
 - Auto-variables view 26
 - basic debugging 439–481
 - contextual menu 416
 - crashing 496
 - defined 395
 - editing source code 481
 - error messages 504
 - expressions 482–493
 - font selection 457
 - general problems 494
 - guided tour 401–439
 - launch problems 494
 - launching 440
 - problems while running 496
 - running code 443, 445
 - starting up 440
 - stepping into routines 447
 - stepping out of routines 447
 - stepping through code 443, 446
 - stopping code 443
 - symbol hint 415
 - toolbar 403
 - troubleshooting 493–508
 - using 400–401
 - viewing and changing data 469
- Debugger command 652
- debugger configurations, in Target Settings window 357
- Debugger Preferences
 - of IDE Preferences window 283
- Debugger Settings panel 357
 - Auto-target libraries checkbox 358
 - Cache Symbolics Between Runs checkbox 359
 - Choose button 357
 - Data Update Interval 359
 - Location Of Relocated Libraries, Code Resources Or Remote Debugging Folder field 357
 - Log System Messages checkbox 359
 - Stop At Temp Breakpoint On Application Launch checkbox 358
 - Stop At Watchpoints checkbox 359
- Debugger Trap Settings
 - of MetroNub Settings preference panel (Mac OS) 294
- Debugger () 294
- debugging
 - activating for source code 104
 - all Java class files in directory hierarchy 289
 - and refining code 34
 - configuration 367
 - DLLs automatically 291, 358
 - embedded start-up process 443
 - enabling debug info 104
 - general start-up process 440
 - generating debugging information 104–106
 - generating symbolics information 399
 - on multiple monitors 287
 - preparation 397–400
 - preparing a build target 397
 - preparing a file 398
 - projects 37, 368
 - static constructors 290
- debugging for files 104
- debugging information file 35
- debugging source code 395
- DebugStr () 294, 295
- decimal
 - entering data as 475
- Declaration field
 - of New Data Member wizard 244
 - of New Member Function wizard 239
- Declaration File field
 - of New Class wizard 231

-
- Declaration section
 - of New Event dialog box 599
 - of New Method dialog box 589
 - declarations
 - seeing in browser 249
 - default
 - IDE key bindings 660–677
 - Default applet parameter, of Java Applet wizard 521
 - Default command 643
 - default file name extensions 343–345
 - Default Size For Unbounded Arrays field
 - of Display Settings preference panel 286
 - Default Text File Format pop-up menu
 - of Editor Settings preference panel 277
 - defaults
 - for toolbars 320
 - defining
 - actions for custom commands 302
 - Defining Symbols for C/C++ 375
 - Defining Symbols for Pascal 377
 - definition
 - of build target 91
 - of design 93
 - of subproject 92
 - Definition field
 - of New Data Member wizard 244
 - of New Member Function wizard 240
 - Delete button 304
 - Delete command 567
 - Delete command (Windows) 421, 424, 541
 - deleting
 - command groups 304
 - custom keywords 282
 - environment variables 341
 - expressions 421
 - key bindings 309
 - keyboard shortcuts 309
 - menu commands 303, 304
 - text in editor 151
 - dereferencing handles 406
 - Description applet parameter, of Java Applet wizard 521
 - Description field
 - of Java Applet wizard 523
 - of Java Application wizard 526
 - of Register Details window 428
 - Description File field
 - in Register Details window 426
 - design
 - defined 93
 - Design Name text field, of Project panel 532
 - Design pop-up menu, of New window 538
 - Design tab, in Project window 533
 - Design view 42, 533
 - of Project window 50
 - designs
 - adding to projects 531
 - in Targets view 533
 - limitations of 532
 - working with 531–534
 - details
 - viewing for registers 424
 - Details field
 - of Global Optimizations settings panel 354
 - Diagnostic Settings option for makefiles 68
 - dialog box buttons
 - for IDE preferences 254
 - for settings panels 323
 - in Customize IDE Commands window 296
 - in IDE Preferences window 254
 - in Target Settings window 323
 - dialog boxes
 - Set/Clear Breakpoint 458
 - dialog boxes, for RAD browser 584–600
 - Directory field
 - of Customize IDE Commands window (Windows) 302
 - Dirty File marker 143
 - Disable Breakpoint command 459, 640
 - Disable Debugger command 367, 397, 637
 - Disable Third Party Plugins checkbox
 - of Plugin Settings preference panel 265
 - Disable Watchpoint command 641
 - Disassemble command 257, 379, 634
 - disassembling source code 379
 - discarding
 - changes in Customize IDE Commands window 297
 - changes in IDE Preferences window 255
 - changes in Target Settings window 323
 - Disconnect (VCS operation) 608
 - display
 - of added files 79
-

Index

- Display Grid command 544, 650
- Display Settings preference panel 283
 - Attempt To Use Dynamic Type Of C++, Object Pascal And SOM Objects checkbox 285
 - Default Size For Unbounded Arrays field 286
 - In Variable Panes Show All Locals By Default checkbox 285
 - In Variable Panes Show Variable Types By Default checkbox 284
 - Show Tasks In Separate Windows checkbox 285
 - Show Variable Values In Source Code checkbox 286
 - Sort Functions By Method Name In Browser checkbox 285
 - Variable Change Hilite option 284
 - Watchpoint Hilite option 284
- displaying
 - a tab view in the browser 580
- DLL 291, 358
 - automatically debugging 291, 358
- Do Nothing To Non-Debugging Windows radio button
 - of Windowing preference panel 286
- documentation
 - platform targets 27
 - QuickStart and tutorials 27
 - viewers 27
- dollar sign in variable names 500
- Don't Save button 255, 297, 323
- Don't Step Into Runtime Support Code checkbox
 - of Global Settings preference panel 290
- Done button
 - of Custom Keywords dialog box 282
- DOS text files 121
- Drag & Drop Editing checkbox
 - of Editor Settings preference panel 276
- drag and drop
 - adding files to projects 83
 - text 154
 - to remove files and groups 88
- Dump Internal Browse Information After Compile checkbox
 - of Build Extras settings panel 338
- dump memory 437, 643
- Dvorak KCHR Support checkbox
 - of Customize IDE Commands window 307, 308, 312
- dynamic library 33
- Dynamic Scrolling checkbox
 - of Editor Settings preference panel 275
- dynamically linked library. *See* DLL. 291, 358
- E**
- Edit button
 - of Custom Keywords dialog box 281
 - of Custom Keywords settings panel 357
- Edit Item Properties button 552
- Edit Menu 623–626
 - Balance command 625
 - Clear All Breakpoints command 640
 - Clear command 625
 - Commands & Key Bindings command 626
 - Copy command 624
 - Cut command 624
 - Insert Reference Template command (Mac OS) 625
 - Multiple Redo 624
 - Multiple Undo 624
 - Paste command 624
 - Preferences command 626
 - Redo command 624
 - Select All command 625
 - Shift Left command 625
 - Shift Right command 625
 - Target Settings command 626
 - Undo command 623
 - Version Control Settings command 626
- editing
 - in browser 250
 - source code 36
 - source code in debugger 481
- editing support for drag & drop 276
- editing symbols automatically 248
- editing, redoing 156
- editing, undoing 156
- Editor
 - drag and drop 276
 - remembering
 - font preferences 275
- editor 137–162
 - adding text 151
 - balancing punctuation 154
 - color syntax 157
 - configuration 145–149
 - creating panes in window 147

-
- deleting text 151
 - finding a routine in 158
 - font 145
 - go to line number 161
 - going back and Forward 161
 - guided tour 137–144
 - markers 158–160
 - moving text 154
 - navigating text 157–162
 - opening related file 160
 - panes 147–148
 - removing panes in window 148
 - resizing panes in window 148
 - saving window settings 148
 - selecting text 152
 - showing and hiding window toolbars 146
 - text editing 149–157
 - text size 145
 - third-party support 260
 - undoing changes 156–157
 - user interface elements 137
- editor configurations, in Target Settings window 356
- Editor Extensions Menu (Mac OS) 658
- Editor Preferences
 - of IDE Preferences window 272
- Editor Settings preference panel 273
 - Balance While Typing checkbox 275
 - Color Settings 274
 - Context Popup Delay field 277
 - Default Text File Format pop-up menu 277
 - Drag & Drop Editing checkbox 276
 - Dynamic Scrolling checkbox 275
 - Flashing Delay field 277
 - Font Preferences checkbox 275
 - Left Margin Click Selects Line checkbox 276
 - Other Settings 275
 - Relaxed C Popup Parsing checkbox 276
 - Remember section 275
 - Selection Position checkbox 275
 - Sort Function Pop-Up checkbox 276
 - Use Multiple Undo checkbox 276
 - Window Position And Size checkbox 275
- Editor window
 - setting the font 278
 - setting the font size 278
- editors
 - creating menus with Menu editor 574
 - for RAD components 572–577
 - for RAD menu bars 573
 - for RAD pop-up menus 577
 - modifying menus with Menu editor 575
 - removing menus with Menu editor 576
 - third-party 90
- Electric Fish
 - CVS plug-in 602
 - Projector plug-in 602
- elements
 - in toolbars 315
- Emacs text editor 260
- empty project 53
- Empty Project option, of New window 55
- Enable 641
- Enable Automatic Toolbar Help checkbox
 - of IDE Extras preference panel (Mac OS) 262
- Enable Breakpoint command 459, 640
- Enable Debugger command 367, 397, 636
- Enable Debugging command 494
- Enable Watchpoint command 641
- End key 43, 150, 151, 153
- Enter ‘Find’ String command 181, 628
- Enter ‘Replace’ String command 628
- Enter/Return 22
- entering data
 - formats 475
- entering symbols automatically 248
- Entire Word checkbox 185
- enumeration
 - viewing data as 473
- Enumeration command 644
- enumerations
 - auto-completion of names 247
- Environment Settings section
 - of Runtime Settings panel (Windows) 340
- Environment Variable option
 - of Source Trees preference panel 267
 - of Source Trees settings panel 347
- Environment Variable option, in Type pop-up menu 267, 347
- environment variables
 - creating 340
 - deleting 341
 - modifying 341
- Error Button 380
- error messages 383
 - compiler 384

Index

- debugger 504
 - Error Reference command 652
 - Errors & Warnings Window command 648
 - Errors Only option
 - of Plugin Diagnostics 264
 - event sets
 - assigning to build targets 596
 - setting base class and interfaces of 595
 - setting name and location of 592
 - Events tab
 - in Object Inspector 565
 - Events tab view, in browser (RAD) 583
 - examining
 - commands 299
 - project information 100–101
 - RAD objects 541
 - examples
 - of expressions 487
 - Exceptions in Targeted Classes command 653
 - Execute as ToolServer Script command 656
 - Execute field
 - of Customize IDE Commands window (Windows) 302
 - execution
 - killing 451
 - of a single statement 446
 - restarting 452
 - stopping 450
 - Existing button
 - of New Class wizard 233
 - of New Member Function wizard 240
 - Existing Custom Set option, in New Event Set dialog box 590
 - Exit command 622
 - Expand Window command (Mac OS) 646
 - expanding
 - groups 75
 - variables 405, 436, 469
 - Export button 310
 - Export Project command 103, 622
 - exporting
 - commands 310
 - custom keywords 282
 - key bindings 310
 - projects 103–104
 - Expression Simplification 354
 - expressions 482–493
 - and decimal values 487
 - and registers 485
 - and structure members 488
 - and variable values 487
 - as source addresses 484
 - attaching to breakpoints 483
 - changing conditions by dragging 484
 - creating 482
 - creating by dragging 483
 - defined 482
 - deleting 421
 - examples 487
 - formal syntax 489
 - in Breakpoints window 483
 - in Expressions window 482
 - in Memory window 484
 - interpretation of 482
 - limitations of 485
 - logical 488
 - pointers in 488
 - reordering 420
 - special features 485
 - using 485
 - Expressions window 420
 - adding caller variables 477
 - adding items 477
 - and variables 476
 - changing order of items 477
 - Expressions Window command 420, 477, 648
 - Extension column
 - in File Mappings settings panel 342
 - Extension field
 - of File Mappings settings panel 343
 - extension, file name 342
 - external development tools 39
 - external editor support 260
 - Extra Information Button 381
- ## F
- Factory Settings button 255, 297, 323
 - Failure pop-up menu
 - of Build Settings preference panel 259
 - Faster Execution Speed radio button
 - of Global Optimizations settings panel 353
 - fat libraries 393
 - FDI 260
 - figure conventions 22
 - File column 44, 113

-
- %file command-line string 260
 - File Control pop-up menu 49
 - File field
 - of Java Bean wizard 529
 - of New Event Set wizard 592
 - File Locations section, of New Member Function wizard 239, 243
 - File Mappings List 342
 - File Mappings list
 - of File Mappings settings panel 342
 - File Mappings settings panel 341
 - Add button 342
 - Compiler pop-up menu 345
 - Extension column 342
 - Extension field 343
 - File Mappings list 342
 - File Type column 342
 - File Type field 343
 - Flags pop-up menu 345
 - Mapping Info section 342
 - File Menu 619–623
 - Close All command 620
 - Close Catalog command 621
 - Close command 620
 - Exit command (Windows) 622
 - Export Project command 622
 - Find and Open 'Filename' command 620
 - Find and Open File command 620
 - Import Components command 621
 - Import Project command 621
 - New command 619
 - New Text File command 619
 - Open command 619
 - Open Recent command 619
 - Page Setup command (Mac OS) 622
 - Print command 622
 - Print Setup command (Windows) 622
 - Quit command (Mac OS) 623
 - Revert command 621
 - Save A Copy As command 621
 - Save All command 620
 - Save As command 621
 - Save command 620
 - file name extensions 342
 - adding 342
 - default settings 343–345
 - file name suffix 342
 - File Name text field, of File panel 559
 - File pane 421
 - File panel
 - File Name text field 559
 - Location text field 560
 - File Path Caption 143
 - File Sets List 176
 - File Sets Pop-up Menu 176
 - File Sets, saving 194
 - File Systems... 660
 - File tab, of New window 558
 - File Type column
 - in File Mappings settings panel 342
 - File Type field
 - of File Mappings settings panel 343
 - File view
 - Checkout Status column 50
 - Code column 45
 - Data column 46
 - Debug column 46
 - File column 44
 - File Control pop-up menu 49
 - Interface pop-up menu 49
 - Project Checkout Status icon 50
 - Target column 47
 - Touch column 47
 - Files
 - opening 160
 - saving file sets 194
 - saving in default text format 277
 - files 109–127
 - activating debugging for 104
 - adding to projects 78
 - adding to projects (Macintosh) 81
 - adding to projects (Solaris) 82
 - adding to projects (Windows) 80
 - assigning to build targets 98
 - assigning with the Project Inspector 99
 - assigning with the Target column 98
 - automatic stationery setup 58
 - backing up 73
 - building projects 59
 - choosing a default project 74–75
 - closing 123–125
 - compiling 364
 - compiling one file 365
 - compiling selected files 365
 - converting projects from IDE 1.7 format 71
 - creating 109
-

Index

- debugging information file 35
- display of added files 79
- drag and drop adding to projects 83
- dynamic library 33
- empty project 53
- header file 33
- include file 33
- input file creation 32
- interface file 33
- items saved with projects 73
- library file 33
- managing project files 75–90
- modifying projects 59
- moving 86
- naming projects 55
- opening existing 110–116
- opening project files from other hosts 70
- opening projects from earlier IDE versions 71
- opening projects prior to IDE 1.7 (Mac OS) 72
- output file 34, 41
- permissions
 - Checked out 610
 - Locked 611
 - Modify Read-Only 610
 - Read-Only 610
 - Unlocked 611
- preparing for debugging 398
- printing 125–127
- printing options 125
- project file 33
- project stationery 52, 53
- recompiling 365
- removing 87
- removing with drag and drop 88
- removing with menu commands 87
- resource file 32
- reverting 157
- reverting to saved 127
- saving 117–121
- saving a copy of a project 73
- selecting 77
- setting extensions 362
- source code file 32
- static library 33
- symbolics file 35
- symbolics files 396–397
- touching and untouching 89
- types of project files 52
- Files pane 412
 - and breakpoints 412
 - navigating code with 455
- Files view 42
 - of Project window 44
 - sorting items 47
- Final option, of Specifier pop-up menu 529, 589, 595
- Find and Open ‘Filename’ command 620
- Find and Open File command 620
- Find command 627
- Find Definition & Reference command 164, 630
- Find Definition command 163, 630
- Find in Next File command 627
- Find in Previous File command 628
- Find Next command 181, 627
- Find Previous command 181, 183, 627
- Find Previous Selection command 628
- Find Reference command 164
- Find Reference command (Mac OS) 630
- Find Reference Using pop-up menu
 - of IDE Extras preference panel (Mac OS) 263
- Find Selection command 628
- Find symbols with prefix 247
- Find symbols with substring 247
- Find Window
 - guided tour 169–179
- finding
 - function overrides in browser 250
- finding all implementations of function 227
- finding symbol definitions 163
- Finish button
 - of Browser wizards 230
 - of Makefile Importer wizard 66
 - of RAD wizards 515, 544, 591
- Fixed
 - viewing data as 474
- Fixed command (Mac OS) 644
- Flags pop-up menu
 - Ignored By Make flag 345
 - Launchable flag 345
 - of File Mappings settings panel 345
 - Precompiled flag 345
 - Resource File flag 345
- Flashing Delay field
 - of Editor Settings preference panel 277
- Floating Document Interface. *See* FDI. 260
- floating point

- entering data as 475
- viewing data as 473
- Floating Point command 644
- floating toolbar
 - anchoring 320
- folders
 - Registers 425
- Font & Tabs preference panel 277
 - Auto Indent checkbox 279
 - Font pop-up menu 278
 - Sample field 278
 - Size pop-up menu 278
 - Tab Indents Selection checkbox 279
 - Tab Settings section 278
 - Tab Size field 279
- Font pop-up menu
 - of Font & Tabs preference panel 278
- Font Preferences checkbox
 - of Editor Settings preference panel 275
- font selection in debugger 457
- font size
 - setting 278
- fonts
 - setting 278
- Format pop-up menu
 - of Register Details window 427
- formats
 - entering data in 475
 - of data views 473
- FPU Register window 408
- FPU Registers command 431
- FPU Registers window 406, 431
- FPU Registers Window command 431
- Fract
 - viewing data as 474
- Fract command (Mac OS) 644
- Frame Class section, of Java Application wizard 525
- Frame Class section, of Java Frame wizard 545
- function overrides
 - finding in browser 250
- Function pop-up menu 409, 412, 415
 - sorting alphabetically 409, 415
- functions
 - auto-completion of names 247
 - describing name and location of 238
 - specifying file locations for 239, 243

- Functions pane 413, 415
- functions, finding all 227

G

- General Preferences
 - of IDE Preferences window 256
- General Registers command 430, 480
- General Registers window 406, 408, 430
- General Settings section
 - of Runtime Settings panel (Windows) 340
- Generate button
 - of Makefile Importer wizard 66
- Generate button, of Browser wizards 231
- Generate button, of RAD wizards 516, 545
- Generate Constructors and Destructors checkbox,
 - of New Class wizard 235
- Generate HTML Page checkbox
 - of Java Applet wizard 518
- Generate Standard Methods checkbox
 - of Java Applet wizard 518
- generating
 - a link map 369
 - symbolics information 399
- generating software 34
- Get 606
- Get (VCS operation) 607
- Get next symbol 247
- Get previous symbol 247
- getting started with IDE 29
- Global Optimizations settings panel 352
 - Details field 354
 - Faster Execution Speed radio button 353
 - Optimization Level Slider 353
 - Optimize For section 353
 - Smaller Code Size radio button 353
- Global Register Allocation 354
- Global Register Allocation Only For Temporary Values 354
- Global Settings preference panel 288
 - Auto Target Libraries checkbox 290
 - Automatically Launch Applications When
SYM File Opened checkbox 290
 - Cache Edited Files Between Debug Sessions
checkbox 289
 - Confirm “Kill Process” When Closing Or
Quitting checkbox 290

Index

- Confirm Invalid File Modification Dates When Debugging checkbox 289
- Debug All Java Class Files In Directory Hierarchy checkbox 289
- Don't Step Into Runtime Support Code checkbox 290
- Maintain Files In Cache For field 289
- Purge Cache button 289
- global variables 421
 - in Variables pane 405
 - placing in separate windows 422
 - viewing in debugger 470
- Global Variables window 421
- Global Variables Window command 470, 649
- globals
 - auto-completion of names 247
- Glossary command 652
- glue code 502
- GNU, converting make files to projects 53
- Go Back command 161, 631
 - limitations of 247
- Go Forward command 161, 631
 - limitations of 247
- Go To Line command 631
- going
 - back and forward 161
- going to a line number 161
- grep 198–206
- Group command 544, 650
- Group pop-up menu 113
- groups
 - and Debug Info marker 106
 - creating 87
 - expanding and collapsing 75
 - moving 86
 - naming 88
 - removing 87
 - removing with drag and drop 88
 - removing with menu commands 87
 - selecting 77
- guided tour
 - of build target settings 321–324
 - of IDE preferences 253–255
- H**
- Hard Reset command (Embedded) 638
- Has Data Member section, of New Property dialog box 586
- header file 33
- header files
 - opening 117
 - precompiling 371–378
- Height field, of Java Applet wizard 520
- Help Menu 652–653
 - About Metrowerks command (Windows) 653
- help viewers
 - QuickHelp (Mac OS) 164
 - QuickView (Mac OS) 165
 - THINK Reference (Mac OS) 166
 - WinHelp (Windows) 164
- (Helper Apps) folder 261
- hexadecimal
 - entering data as 475
 - viewing data as 473
- Hexadecimal command 644
- Hide Breakpoints command 460, 640
- Hide Floating Toolbar command 316, 654
- Hide Main Toolbar command (Windows) 316
- Hide Non-Debugging Windows radio button
 - of Windowing preference panel 286
- Hide Window Toolbar command 317, 403, 654
- hiding
 - editor window toolbars 146
 - toolbars 315
- hierarchy expansion triangle 225
- Hierarchy view, browser 211
- History (VCS operation) 608
- Home key 43, 150, 151, 153
- host
 - defined 18
- Host Application For Libraries And Code Resources field
 - of Runtime Settings panel 339
- Host Flags pop-up menu
 - All option 333
 - in Access Paths settings panel 333
 - None option 333
- host terminology conventions 21
- hosts
 - opening project files 70
- How To command 652
- HSpace field, of Java Applet wizard 520
- HTML

specifying page in Java Applet wizard 518
HTML Page section, of Java Applet wizard 518
Hypertext Markup Language. *See* HTML. 518
hypertext navigation 21

I

IDE 31

- about 17
- adding menu commands 300
- and VCS 38
- available tools 18
- browser 207, 579
- choosing preferences 256–296
- configuring options 253
- configuring target options 321
- converting multiple IDE 1.7 projects to latest version 71
- converting single IDE 1.7 project to latest version 71
- customizing 38, 296–320
- debugging source code 395
- default key bindings 660–677
- defined 17
- getting started 29
- installation 31
- introduction 17, 35–39
- Linux system requirements 31
- Mac system requirements 30
- menu reference 617–660
- online documentation 162
- opening projects from earlier versions 71
- preference panel dialog box buttons 254
- preference panels 254
- programming concepts 31–35
- RAD components 569
- RAD designs and layouts 509
- RAD Layout editor 537
- RAD tools defined 37
- reference information 617
- removing menu commands 303
- scripting 38
- Scripts menu (Mac OS) 657
- settings panel dialog box buttons 323
- software generation 34
- Solaris system requirements 31
- system requirements 29–31
- Windows system requirements 30
- working with projects 41

IDE command 652

IDE Extras preference panel 259

- Enable Automatic Toolbar Help checkbox (Mac OS) 262

- Find Reference Using pop-up menu (Mac OS) 263

- Launch Editor field 260

- Launch Editor W/ Line # field 260

- Recent Projects field 259

- Use BBETM Extensions checkbox (Mac OS) 262

- Use External Editor checkbox 261

- Use Multiple Document Interface checkbox (Windows) 260

- Use Script Menu checkbox (Mac OS) 262

- Use Third Party Editor checkbox (Windows) 260

- Use ToolServer Menu checkbox (Mac OS) 262

- Zoom Windows To Full Screen checkbox 261

IDE preferences

- guided tour 253–255

IDE Preferences window

- Browser Display panel 272

- Build Settings panel 256

- Debugger Preferences 283

- dialog box buttons 254

- discarding changes 255

- Editor Preferences 272

- Font & Tabs panel 277

- General Preferences 256

- IDE Extras panel 259

- Plugin Settings panel 263

- preference panels 254

- Source Trees panel 266

- Syntax Coloring panel 279

IDE User Guide

- overview 19–20

identifier icon, in browser 222

identifying

- symbols in browser database 245

Ignore Case checkbox 184

Ignore Traps option

- of MetroNub Settings preference panel (Mac OS) 295

Ignored By Make flag

- of Flags pop-up menu 345

Implements List section

- of Java Bean wizard 531

Index

- of New Event Set wizard 596
 - Import button 310
 - Import Components button 552
 - Import Components command 621
 - Import Project command 103, 621
 - Import Weak command 392
 - Importable XML option (Mac OS) 103
 - importing
 - commands 310
 - custom keywords 282
 - key bindings 310
 - projects 103–104
 - In Variable Panes Show All Locals By Default checkbox
 - of Display Settings preference panel 285
 - In Variable Panes Show Variable Types By Default checkbox
 - of Display Settings preference panel 284
 - inactive
 - breakpoints 423
 - include file 33
 - Include File Automatically Added For Member Type field
 - of New Data Member wizard 244
 - Include File Cache field
 - of Build Settings preference panel 258
 - #include files 235
 - Include Files Automatically Added For Return Type And Parameters field
 - of New Member Function wizard 240
 - Include Files section, of New Class wizard 235
 - Include Files That Will Automatically Be Added For Base Classes field
 - of New Class wizard 235
 - Include Insert Template Commands In Context Menu checkbox
 - of Browser Display preference panel 273
 - indenting
 - lines of text automatically 279
 - infinite loops
 - escaping from (Mac OS) 451
 - Info Menu (Solaris) 659
 - information
 - examining in a project 100–101
 - Information section, of Java Applet wizard 522, 526
 - inheritance
 - analyzing in browser 250
 - Inherited access icon 222
 - Initialize Before command 393
 - Initializer field
 - of New Data Member wizard 242
 - Initializer field, in New Property dialog box 586
 - Inline checkbox
 - of New Member Function wizard 239
 - input files
 - creation of 32
 - Insert Information In Every File checkbox
 - of Java Applet wizard 523
 - of Java Application wizard 527
 - Insert Reference Template command (Mac OS) 625
 - Insert Template command 657
 - Insert Template contextual menu command 273
 - Inside option, of New Event Set wizard 594
 - installation
 - of CodeWarrior IDE 31
 - Instruction Scheduling 356
 - Integrated Development Environment 17
 - interface file 33
 - Interface Pop-up Menu 139
 - Interface pop-up menu 49, 89
 - Interfaces pop-up menu 114
 - Internet Protocol. *See* IP.
 - Interpret DOS And Unix paths checkbox
 - in Access Paths settings panel (Mac OS) 332
 - Intersolv
 - PVCS plug-in 602
 - introduction to IDE 17
 - IP 292, 295, 296
- ## J
- Java 28
 - AWT RAD components 569–570
 - Swing RAD components 570–572
 - Java Applet wizard 516
 - absbottom option 520
 - absmiddle option 520
 - Add button 522
 - Align pop-up menu 520
 - Applet Class section 516
 - Applet Name field 519
 - Applet Parameters section 521
 - Applet Title field 523
 - AWT Applet radio button 518

-
- baseline option 520
 - bottom option 520
 - Can Run As A Standalone Application checkbox 518
 - Class Name field 517
 - Codebase/Archive field 519
 - Default applet parameter 521
 - Description applet parameter 521
 - Description field 523
 - Generate HTML Page checkbox 518
 - Generate Standard Methods checkbox 518
 - Height field 520
 - HSpace field 520
 - HTML Page section 518
 - Information section 522, 526
 - Insert Information in Every File checkbox 523
 - left option 520
 - Location field 517
 - middle option 520
 - Name applet parameter 521
 - Package Name field 517
 - Remove button 522
 - right option 520
 - Swing Applet radio button 518
 - texttop option 520
 - Title field 519
 - top option 520
 - Type applet parameter 521
 - Use Archive (JAR) field 520
 - Variable applet parameter 521
 - VSpace field 520
 - Width field 520
 - Java Applet Wizard, of New window 512
 - Java applets
 - describing class of 516
 - Java Application wizard 523
 - Application Class section 523
 - Application Title field 526
 - AWT Frame radio button 526
 - Class Name field 523, 525
 - Create A Frame On Startup checkbox 524
 - Description field 526
 - Frame Class section 525
 - Insert Information in Every File checkbox 527
 - Location field 524, 526
 - Package Name field 524, 526
 - Swing Frame radio button 526
 - Java Application Wizard, of New window 512
 - Java applications
 - describing class of 523
 - Java Bean wizard 527
 - Access pop-up menu 529
 - Base Class and Interfaces section 529
 - Base Class section 530
 - Choose button 529
 - Class Is A Bean checkbox 528
 - Class Name field 528
 - File field 529
 - Implements List section 531
 - Modifiers section 529
 - Name and Location section 527
 - New File option 529
 - Non-Visual Bean/Custom option 530
 - Package field 529
 - Specifier pop-up menu 529
 - Visual Bean option 530
 - Java Bean Wizard, of New window 512
 - Java Debugging preference panel 291
 - JDK Version pop-up menu 292
 - Port ID field 292
 - Protocol pop-up menu 292
 - Remote Debugging checkbox 292
 - Remote IP Address field 292
 - Timeout field 292
 - Java Exceptions Submenu 653
 - All Exceptions command 653
 - Exceptions in Targeted Classes command 653
 - No Exceptions command 653
 - Java Frame wizard 545
 - AWT Frame radio button 546
 - Class Name field 545
 - Frame Class section 545
 - Location field 546
 - Package Name field 545
 - Swing Frame radio button 546
 - Java frames
 - describing class of 545
 - Java Settings preference panel 292
 - Class For Debugging field (Windows) 293
 - JView Arguments field (Windows) 293
 - Program Arguments field (Windows) 293
 - JButton component 571
 - JCheckBox component 571
 - JComboBox component 571
 - JDK Version pop-up menu
 - of Java Debugging preference panel 292
 - JEditorPane component 571
-

Index

- JLabel component 571
- JList component 571
- JMenuBar component 571
- JOptionPane component 571
- JPanel component 571
- JPasswordField component 571
- JPopupMenu component 571
- JProgressBar component 571
- JRadioButton component 571
- JScrollBar component 572
- JScrollPane component 572
- JSlider component 572
- JSplitPane component 572
- JTabbedPane component 572
- JTable component 572
- JTextArea component 572
- JTextField component 572
- JTextPane component 572
- JToggleButton component 572
- JToolBar component 572
- JTree component 572
- jumping to
 - markers 160
- JView Arguments field
 - of Java Settings preference panel (Windows) 293

K

- K&R coding
 - relaxing C popup parsing 276
- Keep Program In Background While Stepping checkbox
 - of MetroNub Settings preference panel (Mac OS) 293
- key bindings
 - adding 308
 - customizing 304
 - defaults 660–677
 - deleting 309
 - exporting 310
 - importing 310
 - modifying 306
 - removing 309
 - restrictions in choosing 306
- Key Bindings section
 - of Customize IDE Commands window 300
- key combination conventions 23

- keyboard conventions 22
 - Alt/Option 23
 - Backspace/Delete 22
 - Ctrl/Command/Meta 23
 - Enter/Return 22
 - multiple modifier keys 23
 - Solaris 23
- keyboard navigation 150
- Keyboard Preferences... 660
- keyboard shortcuts
 - adding 308
 - deleting 309
 - modifying 306
 - removing 309
 - restrictions 306
- keys
 - modifier key legend (Mac OS, Solaris, and Linux) 661
 - prefix key timeout 312
 - prefix keys 310
 - Quote Key prefix 311
- keywords
 - coloring 280
- Kill 444
- Kill command 451, 638
 - in debugger toolbar 403
- Kill command (debugger) 445
- killing
 - execution of code 451
- killing execution 451
 - compared to stopping 452

L

- Label (VCS operation) 608
- Label component
 - components
 - Label 570
- Launch Editor field
 - of IDE Extras preference panel 260
- Launch Editor W/ Line # field
 - of IDE Extras preference panel 260
- Launchable flag
 - of Flags pop-up menu 345
- launching debugger 440
 - problems 494
- Layout editor 511, 537, 537–544
- Layout editor contextual menu 543

-
- Layout Menu 650–651
 - Align To Grid command 651
 - Bring To Front command 650
 - Display Grid command 650
 - Group command 650
 - Send To Back command 650
 - Snap To Grid command 650
 - Ungroup command 650
 - layout wizards 544–546
 - Java Frame Wizard 538
 - layouts
 - and use with Live view 554
 - creating 537
 - creating objects 540
 - limitations of 538
 - manipulating RAD objects 540
 - modifying 538
 - removing objects 541
 - working with 534–536
 - left margin
 - clicking to select a line 276
 - Left Margin Click Selects Line checkbox
 - of Editor Settings preference panel 276
 - left option, of Java Applet wizard 520
 - libraries
 - dynamic library 33
 - static library 33
 - library file 33
 - Lifetime Based Register Allocation 356
 - limitations
 - on choosing key bindings 306
 - Line button in hierarchy window 224
 - %line command-line string 260
 - Line Number Button 144
 - line number, going to 161
 - linear code navigation 452
 - linear navigation through code 452
 - Link Order view 42
 - of Project window 51
 - link order, setting 365
 - linker 34
 - Linker pop-up menu
 - in Target Settings panel 328
 - linker, shared libraries 391
 - Linking 361–391
 - linking
 - projects 363–371
 - linking code 37
 - Linux
 - system requirements 31
 - List button, in browser 220
 - List component 570
 - List view
 - contrasted with Live view 554
 - List View button 553
 - Live Range Splitting 355
 - Live view
 - and use with RAD layouts 554
 - contrasted with List view 554
 - Live View button 553
 - Local Path (VCS option) 605
 - local variables 477
 - viewing in debugger 469
 - Location field
 - of Java Applet wizard 517
 - of Java Application wizard 524, 526
 - of Java Frame wizard 546
 - Location Of Relocated Libraries, Code Resources Or Remote Debugging Folder field 357
 - Location text field, of File panel 560
 - Location text field, of Project panel 513, 532
 - Lock command 558
 - Lock item
 - in Component Information Bar 556
 - Log All Statements Bypassed checkbox 68
 - Log Build Rules Discarded checkbox 68
 - Log DebugStr Messages checkbox
 - of MetroNub Settings preference panel (Mac OS) 295
 - Log System Messages checkbox
 - of Debugger Settings panel 359
 - Log Targets Bypassed checkbox 68
 - Log window 434
 - logical expressions 488
 - Lookup Symbol command 657
 - Loop Transformations 355
 - Loop Unrolling 356
 - Loop-Invariant Code Motion 355
 - loops and conditional breakpoints 462
 - loops, infinite
 - escaping from (Mac OS) 451
 - lvalue
 - defined 438
-

M

- Mac OS 28, 30
 - system requirements 30
- macros
 - auto-completion of names 248
- MacsBug Handles Traps option
 - of MetroNub Settings preference panel (Mac OS) 295
- Main Text color setting 274
- Mainsoft
 - SourceSafe plug-in 602
- Maintain Files In Cache For field
 - of Global Settings preference panel 289
- Make command 257, 366, 371, 373, 400, 635
- Makefile Importer 53, 65–68
- Makefile Importer wizard
 - Cancel button 66
 - Finish button 66
 - Generate button 66
 - options 66
- Makefile Location option 66
- makefiles
 - converting to projects 65–68
 - creating new projects from 65
- managing
 - files in a project 75–90
- manipulating
 - objects in RAD layouts 540
- manual conventions 23
- manual style 20
- manuals
 - platform targets 27
 - QuickStart and tutorials 27
- .MAP extension 369
- Mapping Info section
 - of File Mappings settings panel 342
- margins
 - clicking to select a line 276
- Marker Pop-up Menu 141
- marker, Debug Info 106
- markers
 - adding 158
 - Dirty File 143
 - jumping to 160
 - removing 160
- .mcp extension 56, 513
- MDI 260, 287
 - meaning of
 - bit values in register 428
- Member Function Declaration section, of New Member Function wizard 238
- member functions
 - specifying file locations for 239, 243
 - viewing in browser 218
- Member Functions pane, in browser 222
- memory
 - changing 439
 - viewing memory at an address 478
 - viewing raw memory 477
- memory dump 437, 471, 643
- Memory window 437
 - and memory dump 471
 - changing base address 438
 - changing contents of 439
- memory window 437
- menu commands
 - creating 301
 - creating groups for 300
 - customizing 300, 303
 - deleting 304
 - to remove files and groups 87
- Menu editor 573
 - creating menu items 574
 - modifying menu items 575
 - removing menu items 576
- menu reference
 - for IDE 617–660
- MenuBar component 570
- menus
 - creating with Menu editor 574
 - modifying with Menu editor 575
 - removing with Menu editor 576
- Merge Into Output command 393
- Merge shared libraries option 393
- Message List Pane 382
- Message Window
 - command 648
 - correcting compiler errors 385
 - error and warning messages 383
 - stepping through messages 384
 - using 382–391
- messages
 - stepping through 384
- Method option 605
- methods

-
- alphabetizing C++ and Object Pascal methods 413
 - methods (C++)
 - alphabetizing 285
 - Methods tab view, in browser (RAD) 582
 - MetroNub Settings panel
 - Remote IP/
 - Port field (Mac OS) 295
 - MetroNub Settings preference panel 293
 - Always Use File Mapping For Symbolics checkbox (Mac OS) 294
 - Debugger Trap Settings (Mac OS) 294
 - Ignore Traps option (Mac OS) 295
 - Keep Program In Background While Stepping checkbox (Mac OS) 293
 - Log DebugStr Messages checkbox (Mac OS) 295
 - MacsBug Handles Traps option (Mac OS) 295
 - PowerPC trap handling 295
 - Remote Debugging checkbox (Mac OS) 295
 - Stop For Traps option (Mac OS) 295
 - Metrowerks
 - CVS plug-in 602
 - Projector plug-in 602
 - SourceSafe plug-in 602
 - Metrowerks Tool Set pop-up menu 67
 - MFC
 - seeing classes in browser 251
 - Microsoft
 - SourceSafe plug-in 602
 - Microsoft Windows 30
 - middle option, of Java Applet wizard 520
 - Minimize Non-Debugging Windows radio button
 - of Windowing preference panel (Windows) 287
 - Minimize Window command (Windows) 646
 - MIPS 28
 - mixed
 - viewing 409, 415
 - .mkb extension 310
 - modification dates
 - in debugger 289
 - synchronizing 90, 369
 - Modification dates, synchronizing 90
 - Modified item
 - in Component Information Bar 556
 - modifier key conventions 22
 - modifier key legend (Mac OS, Solaris, and Linux) 661
 - Modifiers section
 - of Java Bean wizard 529
 - of New Data Member wizard 242
 - of New Event dialog box 599
 - of New Event Set wizard 595
 - of New Member Function wizard 239
 - of New Method dialog box 588
 - modifying
 - environment variables 341
 - existing commands 299
 - key bindings 306
 - keyboard shortcuts 306
 - layouts 538
 - menu items in Menu editor 575
 - projects 59
 - Monitor For Debugging pop-up menu
 - of Windowing preference panel (Mac OS) 287
 - monitors
 - debugging on multiple 287
 - Move Open Windows To Debugging Monitor
 - When Debugging Starts checkbox
 - of Windowing preference panel (Mac OS) 288
 - moving
 - files and groups 86
 - projects 102–103
 - MPW 124
 - MSL C Reference command 652
 - MSL C++ Reference command 652
 - multi-class hierarchy, browser 224–226
 - Multi-file searches 175–179
 - multiple data types 477
 - Multiple Document Interface. *See* MDI. 260
 - multiple IDE 1.7 projects
 - converting to latest IDE version 71
 - multiple modifier keys
 - documentation conventions 23
 - multiple RAD objects, selecting 540
 - multiple redo 624
 - multiple undo 624
 - multiple-monitor debugging 287
 - Mutable option, of Specifier pop-up menu 242
 - MW Perl 692
 - MW Perl.dll 691
 - MWPerl Panel.dll 691
-

Index

N

- Name and Location section
 - of Java Bean wizard 527
 - of New Event Set wizard 592
- Name and Location section, of New Class wizard 231
- Name applet parameter, of Java Applet wizard 521
- Name column
 - in Source Trees preference panel 267
 - in Source Trees settings panel 347
- Name field
 - in Source Trees preference panel 267
 - in Source Trees settings panel 347
 - of Customize IDE Commands window 299, 301
 - of New Data Member wizard 241
 - of New Event dialog box 599
 - of New Member Function wizard 239
 - of New Method dialog box 587, 589
 - of New Property dialog box 584, 586
 - of Source Trees preference panel 268
 - of Source Trees settings panel 347
- Name item
 - in Component Information Bar 556
- Namespace field
 - of New Class wizard 233
- Namespaces Required For Base Classes And Constructor Parameters field, of New Class wizard 235
- Namespaces Required For Parameters field
 - of New Member Function wizard 239
- Namespaces Required For Type field
 - of New Data Member wizard 241
- naming
 - projects 55
- Native checkbox, of New Method dialog box 589
- navigating
 - code 452
 - Project window 43
 - through code in the browser 246
- navigating code
 - by call chain 453
 - in browser window 454
 - linear 452
 - using a contextual menu 246
 - using the Go Back and Go Forward commands 246
- navigation
 - basic concepts 149
 - using hypertext 21
 - via scrollbars 149
 - via the keyboard 150
- navigation through code
 - linear 452
 - via routine call chain 453
 - via Source Code Browser window 454
 - with the Files pane 455
- New button
 - of New Class wizard 233
 - of New Member Function wizard 240
- New Catalog button 551
- New Class Browser command 647
- New Class command 229, 645
- New Class wizard 231
 - Access pop-up menu 235
 - Add Files To Targets list 237
 - Additional Header Include Files 235
 - After option 233
 - Base Classes and Methods section 234
 - Base Classes field 234
 - Before option 233
 - Class field 232
 - Class Name field 231
 - Constructor Parameters field 235
 - Declaration File field 231
 - Existing button 233
 - Generate Constructors and Destructors checkbox 235
 - Include Files section 235
 - Include Files That Will Automatically Be Added For Base Classes field 235
 - Name and Location section 231
 - Namespace field 233
 - Namespaces Required For Base Classes And Constructor Parameters field 235
 - New button 233
 - New File option 232
 - Project field 236
 - Relative To Class option 232
 - Targets section 236
 - Use separate file for member definitions checkbox 233
 - Virtual destructor checkbox 235
- New command 53, 511, 531, 538, 558, 619
- New Command button 301

-
- New Custom Event Set option, in New Event Set dialog box 590
 - New Data Member command 645
 - commands
 - New Data Member 230
 - New Data Member wizard 241
 - Additional Header Include Files 244
 - Const checkbox 242
 - Data Member Declaration section 241
 - Declaration field 244
 - Definition field 244
 - Include File Automatically Added For Member Type field 244
 - Initializer field 242
 - Modifiers section 242
 - Name field 241
 - Namespaces Required For Type field 241
 - Type field 241
 - Volatile checkbox 242
 - New Event command 597, 645
 - New Event dialog box
 - Add button 599
 - Cancel button 599
 - Declaration section 599
 - dialog boxes
 - New Event 597
 - Modifiers section 599
 - Name field 599
 - Optional Throws field 599
 - Package Required For Parameters field 599
 - Parameters field 599
 - Synchronized checkbox 599
 - New Event Set command 589, 645
 - New Event Set dialog box
 - dialog boxes
 - New Event Set 589
 - Existing Custom Set option 590
 - New Custom Event Set option 590
 - New Event Set wizard 591
 - Access pop-up menu 595
 - Add File To Targets list 596
 - After option 594
 - Base Class and Interfaces section 595
 - Base Class section 596
 - Before option 594
 - Class field 594
 - Class Is A Bean checkbox 592
 - Class Name field 592
 - Custom option 596
 - File field 592
 - Implements List section 596
 - Inside option 594
 - Modifiers section 595
 - Name and Location section 592
 - New File option 593
 - Package field 594
 - Relative To Class option 594
 - Set button 232, 593, 594
 - Specifier pop-up menu 595
 - Targets section 596
 - Visual Component option 596
 - New Expression command 643
 - New File option
 - of Java Bean wizard 529
 - of New Class wizard 232
 - of New Event Set wizard 593
 - New Folder button 551
 - New Folder command 557
 - New Group button 301
 - New Member Function command 230, 645
 - New Member Function wizard 237
 - Additional Header Include Files 241
 - Const checkbox 239
 - Declaration field 239
 - Definition field 240
 - Existing button 240
 - File Locations section 239, 243
 - Include Files Automatically Added For Return Type And Parameters field 240
 - Inline checkbox 239
 - Member Function Declaration section 238
 - Modifiers section 239
 - Name field 239
 - Namespaces Required For Parameters field 239
 - New button 240
 - Parameters field 239
 - Return Type field 239
 - New Method command 586, 645
 - New Method dialog box
 - Add button 589
 - Cancel button 589
 - Declaration section 589
 - dialog boxes
 - New Method 586
 - Modifiers section 588
 - Name field 587, 589
-

Index

- Native checkbox 589
 - Optional Throws field 587
 - Package Required For Parameters field 587
 - Parameters field 587
 - Return Type field 587
 - Synchronized checkbox 589
 - New Project dialog box 58
 - New Property command 584, 645
 - New Property dialog box
 - Accessors section 586
 - Add button 586
 - dialog boxes
 - New Property 584
 - Has Data Member section 586
 - Initializer field 586
 - Name field 584, 586
 - Package Required For Type field 586
 - Summary section 586
 - Transient checkbox 586
 - Type field 585, 586
 - Volatile checkbox 586
 - New Text File command 373, 375, 377, 619
 - New window
 - Add Design To Project checkbox of Project panel 532
 - Component Catalog File option 559
 - Design Name text field of Project panel 532
 - Design pop-up menu 538
 - Empty Project option 55
 - File Name text field in File panel 559
 - File tab 558
 - Java Applet Wizard 512
 - Java Application Wizard 512
 - Java Bean Wizard 512
 - Location text field of File panel 560
 - Location text field of Project panel 513, 532
 - Object tab 538
 - Project Name text field in Project panel 512
 - Project pop-up menu 538
 - Project Stationery options 53
 - Project tab 53, 511, 531
 - Set button 56, 513, 560
 - Next button
 - of Browser wizards 230
 - of RAD wizards 515, 544, 591
 - No Exceptions command 653
 - None option
 - in Host Flags pop-up menu 333
 - of Bitfield Name pop-up menu 427
 - of Plugin Diagnostics 264
 - None option, of Specifier pop-up menu 239, 242, 529, 589, 595
 - Non-Visual Bean/Custom option, of Java Bean wizard 530
 - note conventions 20
 - notification of completed build 371
 - Nucleus 28
 - number, going to line 161
 - Numeric Keypad Bindings checkbox
 - of Customize IDE Commands window 306, 308, 312
- ## O
- object code 34
 - removing 370
 - Object Inspector 562–567
 - contextual menu 566
 - Events tab 565
 - navigating properties with the keyboard (Windows) 564
 - Object pop-up menu 563
 - Properties tab 564
 - Object Inspector command 562, 649
 - Object Master 90
 - Object Pascal
 - methods, alphabetizing 413
 - Object pop-up menu
 - and keyboard navigation (Windows) 563
 - in Object Inspector 563
 - Object tab, of New window 538
 - objects
 - creating in RAD layouts 540
 - manipulating in RAD layouts 540
 - removing from RAD layouts 541
 - odoc AppleEvent 261
 - OK button 336
 - online documentation viewers 27
 - online help and references 162
 - online references
 - QuickHelp (Mac OS) 164
 - QuickView (Mac OS) 165
 - THINK Reference (Mac OS) 166
 - WinHelp (Windows) 164
 - Open Catalog button 548, 552
 - Open command 69, 619
 - Open File icon, in browser 223

-
- Open Recent command 69, 259, 619
 - Open Scripts Folder command 658
 - Open Windows On Debugging Monitor During
 - Debugging checkbox
 - of Windowing preference panel (Mac OS) 288
 - opening
 - existing projects 68–72
 - project files from other hosts 70
 - projects from earlier IDE versions 71
 - related files 160
 - source files with the browser 248
 - Opening a Related File 116
 - Opening Files from the File Menu 110
 - Opening Files from the Project Window 112
 - Optimization Level Slider
 - of Global Optimizations settings panel 353
 - optimizations
 - Arithmetic Optimizations 354
 - Branch Optimizations 354
 - Common Subexpression Elimination 354
 - Copy And Expression Propagation 354
 - Copy Propagation 354
 - Dead Code Elimination 354
 - Dead Store Elimination 355
 - Expression Simplification 354
 - Global Register Allocation 354
 - Global Register Allocation Only For
 - Temporary Values 354
 - Instruction Scheduling 356
 - Lifetime Based Register Allocation 356
 - Live Range Splitting 355
 - Loop Transformations 355
 - Loop Unrolling 356
 - Loop-Invariant Code Motion 355
 - Peephole Optimization 354
 - Repeated 356
 - Strength Reduction 355
 - Vectorization 356
 - Optimize For section
 - of Global Optimizations settings panel 353
 - optimizing code
 - and impact on breakpoints 463
 - Optional Throws field
 - of New Event dialog box 599
 - Optional Throws field, of New Method dialog box 587
 - options
 - Access paths 329
 - activating the browser 338
 - Add default access path 333
 - adding access paths 334
 - advanced compile options 371
 - Auto-target libraries 358
 - Browser Display 272
 - Build Extras 337
 - build settings 256
 - caching subprojects 338
 - Caching symbolics between runs 359
 - changing access paths 336
 - choosing host application for non-executables 339
 - commands 298
 - Custom keywords 356
 - Debugger settings 357
 - Editor settings 273
 - file mappings 341, 342
 - fonts and tabs 277
 - general runtime settings (Windows) 340
 - Global optimizations 352
 - global source trees 266
 - Host flags for access paths 333
 - IDE extras 259
 - key bindings 304
 - Log system messages 359
 - Plugin Settings 263
 - preferences 256–296
 - project-specific source trees 345
 - removing access paths 336
 - runtime environment settings (Windows) 340
 - Runtime settings 339
 - setting in browser 245
 - syntax coloring 279
 - System Paths 333
 - System paths 332
 - Target settings 327
 - target settings 324–360
 - toolbars 313
 - User paths 332
 - user paths 333
 - using modification date caching 337
 - Options Pop-up Menu 142
 - Other command 652
 - Other Settings
 - of Editor Settings preference panel 275
 - Others Button 179
 - Output Directory field
 - in Target Settings panel 329
-

Index

output file 34, 41
Overlays view 42

P

Package field
 of Java Bean wizard 529
 of New Event Set wizard 594
Package Name field
 of Java Applet wizard 517
 of Java Application wizard 524, 526
 of Java Frame wizard 545
Package Required For Parameters field
 of New Event dialog box 599
Package Required For Parameters field, of New
 Method dialog box 587
Package Required For Type field, of New Property
 dialog box 586
Page Down key 43, 151, 153
Page Setup command 622
Page Setup command (Mac OS) 622
Page Up key 43, 151, 153
Palm OS 28
PalmQuest reference 263
pane
 active 402
 selecting items in 402, 412
Pane Resize Bar 382
Pane Resize boxes 148
Pane Splitter Controls 144
Pane Splitter controls 147
pane zoom box, of Browser window 219
Panel component 570
panels
 for IDE preferences 254
 in Target Settings window 322
panes
 creating 147
 removing 148
 resizing 148, 402, 412
Panes, in editor window 147–148
Parameters field
 of New Event dialog box 599
 of New Member Function wizard 239
Parameters field, of New Method dialog box 587
parsing
 K&R-styled C code 276
Pascal
 correcting circular references 389
Pascal string
 entering data as 475
 viewing data as 473
Pascal String command 644
Pascal, spurious C code in 503
Password field 605
Paste command 154, 624
Path pop-up menu (Mac OS) 145
Peephole Optimization 354
PEF container 393
Perforce
 Perforce plug-in 602
Perforce VCS plugin 602
Perl 692
Perl Panel 692
Perl plug-ins
 installation on Mac OS 692
 installation on Windows 691
Perl script example (Mac OS) 694
Perl scripting 691
 special considerations 695
 with IDE plug-in API 693–695
Perl software plug-ins
 installation 691–692
Perl target settings panel 693
 configuring 692–693
PerlDLL.dll 691
platform target
 defined 18
platform targets 35
 documentation 27
Play Sound After ‘Bring Up To Date’ & ‘Make’
 checkbox, of Build Settings preference panel
 (Mac OS) 258
PlayStation 28
Plugin Diagnostics
 All Info option 264
 Errors Only option 264
 None option 264
 of Plugin Settings preference panel 263
Plugin Settings preference panel 263
 Disable Third Party Plugins 265
 Plugin Diagnostics 263
pointer types
 viewing 474
Pop-up Menu editor 577

-
- pop-up menus
 - Path (Mac OS) 145
 - sorting functions in 276
 - PopupMenu component 570
 - Port ID field
 - of Java Debugging preference panel 292
 - of x86 Settings preference panel (Mac OS) 296
 - Post-Linker pop-up menu
 - in Target Settings panel 329
 - PowerPC 30, 393
 - PowerPC Embedded 28
 - PowerPC trap handling
 - in MetroNub Settings preference panel 295
 - PowerPlant
 - seeing classes in browser 251
 - #pragma, adding markers by using 159
 - Precompile command 257, 371, 373, 374, 376, 378, 634
 - precompile_target 373
 - Precompiled flag
 - of Flags pop-up menu 345
 - precompiled headers
 - automatic updating of 373
 - creating 372
 - precompiling 371–378
 - preference panels 254
 - Browser Display 272
 - Build Settings 256
 - Display Settings 283
 - Editor Settings 273
 - Font & Tabs 277
 - Global Settings 288
 - IDE Extras 259
 - Java Debugging 291
 - Java Settings 292
 - MetroNub Settings 293
 - Plugin Settings 263
 - Source Trees in IDE Preferences window 266
 - Syntax Coloring 279
 - Windowing 286
 - x86 Settings 295
 - preferences 253–255
 - Activate browser coloring 273
 - Auto indent 279
 - Auto target libraries 290
 - Background color 275
 - Balance while typing 275
 - Build before running 256
 - Cache edited files between debug sessions 289
 - changing syntax highlighting colors 280
 - choosing 256–296
 - Close non-debugging windows 287
 - Collapse non-debugging windows 287
 - commands 298
 - Compiler thread stack 258
 - Context popup delay 277
 - controlling syntax highlighting within a window 280
 - Default Size For Unbounded Arrays 286
 - Disable third-party COM plugins 265
 - Display Settings 283
 - Do nothing to non-debugging windows 286
 - Don't step into runtime support code 290
 - Drag and Drop editing 276
 - Dynamic scrolling 275
 - Enable automatic toolbar help (Mac OS) 262
 - enabling remote Java debugging 292
 - Find Reference Using (Mac OS) 263
 - Flashing delay 277
 - fonts and tabs 277
 - for main text color 274
 - global settings for debugger 288
 - Hide non-debugging windows 286
 - IDE extras 259
 - importing or exporting custom keywords 282
 - In variable panes, show all locals by default 285
 - In variable panes, show variable types by default 284
 - Include file cache (Mac OS) 258
 - Include insert template commands in context menu 273
 - Java debugging 291
 - Java settings for debugger 292
 - key bindings 304
 - Left margin click selects line 276
 - MetroNub settings 293
 - MetroNub symbolics settings (Mac OS) 294
 - Minimize non-debugging windows 287
 - Monitor for debugging (Mac OS) 287
 - plugin diagnostics 263
 - Recent Documents 259
 - Recent Projects 259
 - remembering for fonts 275
 - Save open files before build 257
 - Select stack crawl window when task is stopped 290
-

Index

- selection position 275
- setting Java class for debugging (Windows) 293
- setting Java program arguments (Windows) 293
- setting JDK version for remote Java debugging 292
- setting JView arguments (Windows) 293
- setting port ID for remote debugging (Mac OS) 296
- setting port ID for remote Java debugging 292
- setting protocol for remote Java debugging 292
- setting remote IP address for debugging (Mac OS) 296
- setting remote IP address for Java debugging 292
- setting timeout for Java debugging 292
- setting up remote debugging (Mac OS) 296
- Show Message After Building Up-To-Date Project 257
- Show tasks in separate windows 285
- show variable types by default 472
- Show variable values in source code 286
- Sort function popup 276
- Sort functions by method name in browser 285, 413
- Source Trees 266
- Tab indents selection 279
- Tab inserts spaces 279
- Tab size 279
- toolbars 313
- Use BBEdit™ Extensions (Mac OS) 262
- Use External Editor 261
- Use Multiple Document Interface (Windows) 260
- Use Script Menu (Mac OS) 262
- Use ToolServer Menu (Mac OS) 262
- using color for custom keywords 281
- Variable Change Hilite 284
- Watchpoint Hilite 284
- window position and size 275
- Windowing 286
- x86 settings 295
- Zoom windows to full screen 261
- Preferences command 253, 626
- Prefix Key Timeout field
 - of Customize IDE Commands window 312
- prefix keys 310
- Quote Key 311
- timeout of 312
- Pre-Linker pop-up menu
 - in Target Settings panel 329
- Preparing 397
- preparing
 - for debugging 397–400
- Preprocess command 257, 379, 634
- preprocessing code 378
- preprocessor 378
- preprocessor symbols
 - adding to projects 106–107
- Print command 622
- Print Selection Only 126
- Print Setup command (Windows) 622
- Print using Syntax Highlighting option 126
- printing
 - window 126
 - with syntax coloring 126
- Private option, of Access pop-up menu 239, 242, 588, 595
- Process pane 419
- Processes window 417
- Processes Window command 417, 648
- Processes window toolbar 418
- processor registers
 - viewing 480
- Program Arguments field
 - of Java Settings preference panel (Windows) 293
 - of Runtime Settings panel (Windows) 340
- program counter. *See* current-statement arrow.
- Program window 402
 - compared to Source Code Browser window 410
 - debugger toolbar 403
 - defined 402
 - Source pane 406
 - Stack Crawl pane 404
 - Variables pane 404
- programming concepts 31–35
- Project (VCS operation) 607
- Project Checkout Status icon 50
- Project field
 - of New Class wizard 236
- project file 33
- Project Headers checkbox 179

-
- Project Information Caption 381
 - Project Inspector 99, 391
 - Project Inspector command 648
 - Project Menu 632–638
 - Add File command 632
 - Add Files command 632
 - Add Window command 632
 - Bring Up To Date command 634
 - Check Syntax command 633
 - Compile command 634
 - Create New Group command 632
 - Create New Segment command (Mac OS) 633
 - Create New Target command 633
 - Disable Debugger command 637
 - Disassemble command 634
 - Enable Debugger command 636
 - Make command 635
 - Precompile command 634
 - Preprocess command 634
 - Remove Binaries command 635
 - Remove Object Code & Compact command 635
 - Remove Object Code command 635
 - Remove Selected Items command 633
 - Re-Search for Files command 635
 - Reset File Paths command 633
 - Reset Project Entry Paths command 636
 - Resume command (Mac OS) 637
 - Run command 637
 - Set Default Project command 638
 - Set Default Target command 638
 - Stop Build command 635
 - Synchronize Modification Dates command 636
 - Project Name text field, of Project panel 512
 - Project panel
 - Design Name text field 532
 - Location text field 513, 532
 - Project Name text field 512
 - Project Panel
 - Add Design To Project checkbox 532
 - Project pop-up menu, in New window 538
 - Project Relative Path 335
 - project stationery 52, 60, 61
 - automatic setup of 58
 - creating 61
 - folder 61
 - requirements 61
 - Project Stationery options, of New window 53
 - Project tab, in New window 53, 511, 531
 - Project window 612
 - Checkout Status column 50
 - Code column 45
 - Data column 46
 - Debug column 46
 - debug column 399
 - Design tab 533
 - Design view 42, 50
 - File column 44
 - File Control pop-up menu 49
 - Files view 42, 44
 - guided tour 42–51
 - Interface pop-up menu 49
 - Link Order view 42, 51
 - navigating 43
 - Project Checkout Status icon 50
 - sorting items in Files view 47
 - Target column 47
 - Targets view 43, 51
 - toolbar 43
 - Touch column 47
 - Projector VCS plugin 602
 - projects
 - about project stationery 60
 - activating debugging for 104
 - adding designs 531
 - adding files 78
 - adding files (Macintosh) 81
 - adding files (Solaris) 82
 - adding files (Windows) 80
 - adding files with drag and drop 83
 - adding preprocessor symbols to 106–107
 - backing up 73
 - build targets 18
 - building 59, 363–371
 - choosing a default project 74–75
 - choosing stationery 53
 - closing 74
 - compiling 363–371
 - complex project strategy 94
 - controlling debugging in 104–106
 - converting from 1.7 IDE 71
 - converting from IDE 1.7 format 71
 - converting makefiles 65–68
 - creating 52–59
 - creating from stationery 54
 - creating groups 87
-

- creating stationery 61
- creating subprojects within 99
- debug enabling 367
- debugging 37, 368
- debugging setup 104–106
- defined 18
- embedded debugging start-up process 443
- empty project 53
- examining information 100–101
- expanding and collapsing groups 75
- general debugging start-up process 440
- importing and exporting 103–104
- items saved with 73
- linking 363–371
- making 366
- managing files 75–90
- modifying 59
- moving 102–103
- moving files and groups 86
- naming 55
- navigation 43
- new 53
- New Project dialog box 58
- opening existing 68–72
- opening files prior to IDE 1.7 (Mac OS) 72
- opening project files from other hosts 70
- opening projects from earlier IDE versions 71
- opening subprojects 69
- platform target 18
- project file 33
- RAD wizards 53
- removing files and groups 87
- renaming groups 88
- revision control 50
- running 367
- saving 72–73
- saving a copy 73
- selecting files 77
- selecting groups 77
- setting link order 365
- settings 324–360
- stationery 36, 52, 60–65
- stationery folder 61
- subproject defined 92
- switching between 68
- synchronizing modification dates 90
- targets 35
 - defined 18
- touching and untouching files 89

- types 52
- updating 366
- working with complex projects 91–99
- Properties (VCS operation) 608
- Properties command 544
 - commands
 - Properties 558
- Properties tab
 - in Object Inspector 564
- Properties tab view, in browser (RAD) 581
- Protected option, of Access pop-up menu 239, 242, 588, 595
- Protocol pop-up menu
 - of Java Debugging preference panel 292
- Public option, of Access pop-up menu 239, 242, 529, 588, 595
- punctuation balancing 154
 - automatic 155
- Pure Virtual option, of Specifier pop-up menu 239
- Purge Cache button
 - of Global Settings preference panel 289
- PVCS VCS plugin 602

Q

- question mark in variable names 500
- QuickHelp 263
- QuickHelp (Mac OS) 164
- QuickStart 27, 31
- QuickStart and tutorials 27
- QuickStart resources 27
- QuickView 263
- QuickView (Mac OS) 165
- Quit command 623
- Quote Key prefix 311
 - assigning 312

R

- RAD
 - about 509–510
 - adding designs to projects 531
 - additional browser features 579–583
 - additional information for applets 522, 526
 - and CodeWarrior architecture 509
 - assigning build targets to custom event sets 596
 - benefits of 509
 - browser tab control 579

-
- Catalog pane 552
 - Catalog pop-up menu 548
 - catalogs 511
 - class authoring 510
 - Component Catalog toolbar 551
 - Component Catalog window 550–562
 - component editors 572–577
 - Component Information Bar 555
 - component model 510
 - Component Palette 546–550
 - Component Palette toolbar 548
 - Component pane 553
 - component tools 548
 - components 510, 569
 - Content View buttons 553
 - creating component catalogs 558
 - creating layout objects 540
 - creating layouts 537
 - creating parameters for applets 521
 - describing an application's frame class 525
 - describing class of Java applets 516
 - describing class of Java applications 523
 - describing class of Java frames 545
 - design defined 93
 - Design view 533
 - dialog boxes for RAD browser 584–600
 - Events tab view 583
 - Layout editor 511, 537, 537–544
 - layout wizards 544–546
 - manipulating layout objects 540
 - Methods tab view 582
 - modifying layouts 538
 - modifying object events 565
 - modifying object properties 564
 - Object Inspector 562–567
 - Properties tab view 581
 - removing layout objects 541
 - setting base class and interfaces of beans 529
 - setting base class and interfaces of custom event sets 595
 - setting name and location of beans 527
 - setting name and location of custom event sets 592
 - sort order of components 557
 - sorting components 557
 - specifying an HTML page for applets 518
 - wizards 512
 - Java Applet Wizard 531
 - Java Application Wizard 531
 - Java Bean Wizard 531
 - working with designs 531–534
 - working with layouts 534–536
 - RAD components
 - Java AWT 569–570
 - Java Swing 570–572
 - RAD objects
 - examining 541
 - selecting multiple 540
 - RAD projects
 - creating 511–515
 - RAD tools
 - defined 37
 - in CodeWarrior 510–511
 - RAD wizards 53, 515–531
 - Back button 515, 544, 591
 - Cancel button 515, 544, 591
 - Finish button 515, 544, 591
 - Generate button 516, 545
 - Java Applet Wizard 512
 - Java Application Wizard 512
 - Java Bean Wizard 512
 - Java Frame Wizard 538
 - Next button 515, 544, 591
 - RAM Doubler 466
 - Ram Doubler 495
 - Rapid Application Development. *See* RAD.
 - Rational
 - ClearCase plug-in 602
 - raw memory
 - viewing 477
 - Read button
 - of Register Details window 429
 - Recent Projects field
 - of IDE Extras preference panel 259
 - Recent Strings pop-up menu 182, 186
 - recompiling 365
 - recompiling files 365
 - Recursion (VCS operation) 607
 - recursive search of access paths 330
 - enabling and disabling 330
 - redefined functions
 - setting breakpoints 462
 - redo and undo
 - series of actions 276
 - Redo command 624
 - reference information
-

Index

- for IDE 617
- for IDE menus 617–660
- refining code 34
- register coloring 499
- Register Description option
 - of Text View pop-up menu 429
- Register Details option
 - of Text View pop-up menu 429
- Register Details window 424
 - Address field 427
 - Auto text view option 429
 - Bit Value field 427
 - Bit Value Modifier pop-up menu 428
 - Bitfield Description text view option 429
 - Bitfield Name pop-up menu 427
 - Browse button 426
 - changing bit values 428
 - Description field 428
 - Description File field 426
 - Format pop-up menu 427
 - Read button 429
 - Register Description text view option 429
 - Register Details text view option 429
 - Register Value view 427
 - Reset Value button 429
 - Revert button 428
 - Text View pop-up menu 429
 - using 426
 - Write button 429
 - XML files 425
- Register Details Window command 424
- Register Value view
 - of Register Details window 427
- Register Window command 430
- register windows 406, 408
- registers
 - changing bit values 428
 - changing values 430, 431, 432
 - determining meaning of bit values 428
 - in expressions 485
 - Register Details window 424
 - viewing 406, 408, 430, 431, 432
 - viewing details of 424
 - viewing memory pointed to by an address 478
- Registers folder 425
- Registers Windows command 649
- Registry Key option
 - of Source Trees preference panel 268
 - of Source Trees settings panel 347
- Registry Key option, in Type pop-up menu 268, 347
- regular expressions 198–206
- related files
 - opening 160
- Relative To Class option, of New Class wizard 232
- Relative To Class option, of New Event Set wizard 594
- Relaxed C Popup Parsing checkbox
 - of Editor Settings preference panel 276
- Remember Password (VCS option) 605
- Remember Password option 605
- Remember section
 - of Editor Settings preference panel 275
- remembering
 - insertion point location 275
 - selection position 275
- Remote Debugging checkbox
 - of Java Debugging preference panel 292
 - of MetroNub Settings preference panel (Mac OS) 295
 - of x86 Settings preference panel (Mac OS) 296
- Remote IP Address field
 - of Java Debugging preference panel 292
 - of x86 Settings preference panel (Mac OS) 296
- Remote IP/
Port field
 - of MetroNub Settings preference panel (Mac OS) 295
- Remove a file set command 195
- Remove Binaries command 635
- Remove button
 - in Access Paths settings panel 336
 - of Source Trees preference panel 272
 - of Source Trees settings panel 351
- Remove button, of Java Applet wizard 522
- Remove Files command 88
- Remove Object Code & Compact command 635
- Remove Object Code command 370, 635
- Remove Selected Items command 633
- Remove Toolbar Item command 319
- removing
 - all elements from toolbars 319
 - binaries 369, 370
 - custom keywords 282
 - elements from toolbars 319

-
- environment variables 341
 - files and groups 87
 - key bindings 309
 - keyboard shortcuts 309
 - markers 160
 - menu items in Menu editor 576
 - object code 370
 - panes in editor window 148
 - RAD objects in layouts 541
 - Rename commands 558
 - Renaming and saving a file 118
 - reordering
 - expressions 420
 - Repeated optimizations 356
 - Replace & Find Next command 629
 - Replace & Find Previous command 629
 - Replace All command 187, 629
 - Replace command 629
 - require directive 692
 - requirements
 - for creating stationery 61
 - Re-Search for Files command 635
 - Reset command (Embedded) 638
 - Reset File Paths command 633
 - Reset Floating Toolbar command 320, 655
 - Reset Main Toolbar command (Windows) 320
 - Reset Project Entry Paths command 636
 - Reset Value button
 - of Register Details window 429
 - Reset Window Toolbar command 320, 654
 - resize bar, in browser 219
 - Resize command 543
 - resizing
 - columns in Component Information Bar 557
 - panes 412
 - panes in editor window 148
 - resizing panes 402
 - resource file 32
 - Resource File flag
 - in Flags pop-up menu 345
 - resources
 - QuickStart and tutorials 27
 - Restart command 638
 - restarting
 - execution of code 452
 - Restore Window command (Windows) 646
 - restoring
 - default toolbars 320
 - restrictions
 - on choosing key bindings 306
 - Resume command (Mac OS) 637
 - Return Type field
 - of New Member Function wizard 239
 - Return Type field, of New Method dialog box 587
 - revert
 - to last saved version 157
 - Revert button
 - of Register Details window 428
 - Revert command 157, 621
 - Revert Panel button 255, 298, 324
 - revision control 651
 - revision control systems 50
 - see Version Control System (VCS) 601
 - right option, of Java Applet wizard 520
 - Routine Pop-up Menu 158
 - routine pop-Up menu 140
 - Routine pop-up menu. *See* Function pop-up menu.
 - routines
 - seeing definitions in browser 249
 - stepping into 447
 - stepping out of 447
 - Run 444
 - Run App/Script field
 - of Customize IDE Commands window 302
 - Run command 257, 366, 367, 400, 445, 450, 465, 494, 637
 - in debugger toolbar 403
 - running
 - code in debugger 445
 - running code 443
 - running debugger
 - problems 496
 - running debugger. *See* launching debugger.
 - Runtime Settings panel 339
 - Clear button 340
 - Environment Settings section (Windows) 340
 - General Settings section (Windows) 340
 - Host Application For Libraries And Code Resources field 339
 - Program Arguments field (Windows) 340
 - Working Directory field (Windows) 340
- S**
- Sample field
-

Index

- of Font & Tabs preference panel 278
- Save A Copy As command 73, 120, 621
- Save All command 620
- Save As command 118, 435, 621
- Save button 255, 297, 298, 323
- Save buttons 324
- Save command 376, 378, 620
- Save Default Window command 647
- Save Open Files Before Build checkbox
 - of Build Settings preference panel 257
- Save Project Entries Using Relative Paths checkbox
 - in Target Settings panel 329
- Save this File Set command 194
- saving
 - a copy of a project 73
 - files in various text formats 277
 - items saved with projects 73
 - projects 72–73
 - window position and size 275
- Saving all files 118
- Saving files automatically 118
- Saving one file 117
- scripting
 - IDE 38
 - using Perl 691
- scripts
 - IDE scripting 38
- Scripts Menu 657
 - Open Scripts Folder command 658
- Scrollbar component 570
- scrollbar navigation 149
- scrolling
 - setting dynamic scrolling (Mac OS) 275
- ScrollPane component 570
- Search Menu 626–632
 - Compare Files command 631
 - Enter ‘Find’ String command 628
 - Enter ‘Replace’ String command 628
 - Find command 627
 - Find Definition 630
 - Find Definition & Reference command 630
 - Find in Next File command 627
 - Find in Previous File command 628
 - Find Next command 627
 - Find Previous command 627
 - Find Previous Selection command 628
 - Find Reference command (Mac OS) 630
 - Find Selection command 628
 - Go Back command 631
 - Go Forward command 631
 - Go To Line command 631
 - Replace & Find Next command 629
 - Replace & Find Previous command 629
 - Replace All command 629
 - Replace command 629
- Search Menu Find Selection command 628
- search of access paths
 - enabling and disabling 331
- searching 180
 - find and replace 189–197
 - for selection 181
 - multi-file 175–179
 - selected text 180–181
- Segments view 42
- Select All command 567, 625
- Select Stack Crawl Window When Task Is Stopped checkbox 290
 - of Global Settings preference panel 290
- selected text search 180
- selecting
 - lines by clicking in left margin 276
 - multiple RAD objects 540
- selecting items in a pane 402, 412
- selection
 - by keyboard 78
 - by mouse-clicking 77
 - finding 181
 - printing 126
- Selection Position checkbox
 - of Editor Settings preference panel 275
- Send To Back command 544, 650
- Set Breakpoint command 639
- Set button
 - of New Event Set wizard 232, 593, 594
 - of New window 56, 513, 560
- Set Default Project command 638
- Set Default Target command 638
- set up
 - for debugging 397–400
- Set Watchpoint command 466, 641
- Set/Clear Breakpoint At command 458
- Set/Clear Breakpoint dialog box 458
- setting
 - a temporary breakpoint 449

-
- Background Color 275
 - breakpoints 458
 - breakpoints for redefined functions 463
 - breakpoints for templated functions 462
 - font 278
 - font size 278
 - main text color 274
 - options in browser 245
 - tab size 279
 - the current build target 96
 - watchpoints 466
 - setting breakpoints 458
 - setting link order 365
 - Setting print options 125
 - setting up
 - build target for debugging 397
 - files for debugging 398
 - settings
 - for build targets 321–324
 - Source Trees 345
 - Settings option for makefiles 67
 - settings panels 322
 - Access Paths 329
 - Build Extras 337
 - Custom Keywords 356
 - Debugger Settings 357
 - File Mappings 341
 - Global Optimizations 352
 - Runtime Settings 339
 - Source Trees in Target Settings window 345
 - Target Settings 327
 - setup
 - project debugging 104–106
 - shared libraries 393
 - linker options 391
 - sharing files between projects 503
 - Shift Left command 155, 625
 - Shift Right command 155, 625
 - shifting text 155
 - shortcut conventions 22
 - Solaris 23
 - shortcuts
 - entering values in Type field of Java Applet wizard 521
 - keyboard shortcut restrictions 306
 - Show Breakpoints command 459, 640
 - Show Catalog Window command 647
 - Show Floating Toolbar command 316, 654
 - Show Global Toolbar 654
 - Show Inherited checkbox 222, 250
 - Show Main Toolbar command (Windows) 316
 - Show Message After Building Up-To-Date Project checkbox
 - of Build Settings preference panel 257
 - Show Private command 219
 - Show Processes command (debugger) 417
 - Show Protected command 219
 - Show Public command 219
 - Show Tasks In Separate Windows checkbox
 - of Display Settings preference panel 285
 - Show Types command 472, 500, 642
 - show variable types by default 472
 - Show Variable Values In Source Code checkbox
 - of Display Settings preference panel 286
 - Show Window Toolbar command 317, 403, 654
 - showing
 - editor window toolbars 146
 - toolbars 315
 - signed decimal
 - viewing data as 473
 - Signed Decimal command 643
 - single IDE 1.7 projects
 - converting to latest IDE version 71
 - single-class hierarchy, in browser 226
 - Size pop-up menu
 - of Font & Tabs preference panel 278
 - skipping
 - statements 448
 - skipping statements 448
 - avoiding run-time corruption 449
 - consequences of 448
 - Smaller Code Size radio button
 - of Global Optimizations settings panel 353
 - Snap To Grid command 544, 650
 - software
 - generation of 34
 - Solaris 28
 - Copy Files... 660
 - File Systems... 660
 - Info Menu (Solaris) 659
 - keyboard conventions 23
 - Keyboard Preferences... 660
 - system requirements 31
 - Sort Function Pop-Up checkbox
 - of Editor Settings preference panel 276
-

Index

- Sort Functions By Method Name In Browser
 - checkbox
 - of Display Settings preference panel 285
- sort order
 - RAD components 557
- Sort Order button 48
- sorting
 - functions in pop-up menus 276
 - RAD components 557
- sorting items in Files view of Project window 47
- source code
 - activating debugging for 104
 - disassembling 379
 - editing and browsing 36
 - editing in debugger 481
 - font and color 457
 - viewing 502
- Source Code Browser
 - Source pane 414
- Source Code Browser window 410, 454
 - compared to Program window 410
 - compared to Stack Crawl window 410
 - Files pane 412
 - Functions pane 413
 - setting breakpoints in 415
- Source Code Disclosure Triangle 382
- source code file 32
- Source Code Pane 382
- source files
 - compiling 365
 - opening with the browser 248
 - precompiling 371–378
- Source pane
 - disclosure triangle 406
 - of Program window 406
 - of Source Code Browser 414
- Source pane, in browser 223
- source paths 266, 345
- Source pop-up menu 412, 415
- Source Tree Relative 336
- Source Trees
 - difference between panels in IDE Preferences window and Target Settings window 267, 346
- source trees 266, 345
- Source Trees list
 - in Source Trees preference panel 267
 - in Source Trees settings panel 346
- Source Trees preference panel
 - Absolute Path option 267
 - Add button 268
 - Change button 271
 - Choose button 268
 - Environment Variable option 267
 - Name column 267
 - Name field 267, 268
 - of IDE Preferences window 266
 - Registry Key option 268
 - Remove button 272
 - Source Trees list 267
 - Type field 267
- Source Trees settings panel
 - Absolute Path option 347
 - Add button 347
 - Change button 350
 - Choose button 347
 - Environment Variable option 347
 - Name column 347
 - Name field 347
 - of Target Settings window 345
 - Registry Key option 347
 - Remove button 351
 - Source Trees list 346
 - Type field 347
- Sources checkbox 178
- SourceSafe VCS plugin 602
- special features
 - of expressions 485
- Specifier pop-up menu
 - Abstract option 529, 589, 595
 - Final option 529, 589, 595
 - Mutable option 242
 - None option 239, 242, 529, 589, 595
 - of Java Bean wizard 529
 - of New Event Set wizard 595
 - Pure Virtual option 239
 - Static option 239, 242, 589, 595
 - Virtual option 239
- Speed Doubler 466
- stack
 - viewing routine calls 404
- stack contents
 - viewing 479
- Stack Crawl pane 404, 453
- Stack Crawl window 402

-
- compared to Source Code Browser window 410
 - debugger toolbar 403
 - defined 402
 - Source pane 406
 - Stack Crawl pane 404
 - Variables pane 404
 - Stack Editor Windows command 645
 - Start ToolServer command 656
 - starting up
 - debugging for most projects 440
 - debugging for some embedded projects 443
 - statements
 - skipping 448
 - static constructors
 - debugging 290
 - static library 33
 - Static option, of Specifier pop-up menu 239, 242, 589, 595
 - static variables 421
 - stationery 60
 - about project stationery 60
 - and creating projects 54
 - automatic setup of 58
 - creating 61
 - folder 61
 - project 60–65
 - project stationery 53
 - projects 52
 - requirements 61
 - stationery projects 36
 - Status (VCS operation) 608
 - status area
 - of browser 224
 - Step Into 444
 - Step Into command 447, 639
 - in debugger toolbar 403
 - Step Out 444
 - Step Out command 447, 639
 - in debugger toolbar 403
 - Step Over 444
 - Step Over command 446, 638
 - in debugger toolbar 403
 - stepping
 - into routines 447
 - out of routines 447
 - through a single line of code 446
 - through code 446
 - Stepping Buttons 382
 - stepping through code 443
 - Stop 444
 - Stop at EOF option 177, 196
 - Stop At Temp Breakpoint On Application Launch checkbox
 - of Debugger Settings panel 358
 - Stop At Watchpoints checkbox
 - of Debugger Settings panel 359
 - Stop Build command 635
 - Stop command 445, 450, 639
 - in debugger toolbar 403
 - Stop For Traps option
 - of MetroNub Settings preference panel (Mac OS) 295
 - Stop ToolServer command 656
 - stopping
 - execution of code 450
 - stopping code 443
 - stopping execution 450
 - compared to killing 452
 - strategy
 - for creating complex projects 94
 - Strength Reduction 355
 - strings
 - coloring 280
 - style conventions
 - for beginners 21
 - notes 20
 - tips 21
 - typefaces 21
 - warnings 20
 - subproject
 - defined 92
 - subprojects
 - browsing across 247
 - creating within projects 99
 - opening 69
 - Success pop-up menu
 - of Build Settings preference panel 259
 - Summary section, of New Property dialog box 586
 - Swing
 - RAD components 570–572
 - Swing Applet radio button
 - of Java Applet wizard 518
 - Swing Frame radio button
 - of Java Application wizard 526
-

Index

- of Java Frame wizard 546
- Switch To Monitor command 642
- Symantec THINK Reference 263
- symbol definitions, finding 163
- symbol documentation
 - looking up 162
- Symbol hint 415
- Symbol window, in browser 227
- symbolics file 35
 - and debugging 397
 - multiple open files 402, 412
- symbolics files 396–397
 - and location in which saved 397
- symbolics information
 - generating 399
- symbols
 - completing 247
 - defining for C/C++ 375
 - defining for Pascal 377
 - identifying in the browser database 245
- Symbols pane, in Contents window 216
- Synchronize Modification Dates command 369, 636
- Synchronize Status (VCS operation) 607
- Synchronized checkbox
 - of New Event dialog box 599
 - of New Method dialog box 589
- synchronizing
 - modification dates 90, 369
- Synchronizing modification dates 90
- Synergex
 - PVCS plug-in 602
- syntax
 - of expressions 489
- Syntax Coloring
 - and printing 126
- syntax coloring
 - changing 280
- Syntax Coloring command 280
- Syntax Coloring preference panel 279
- Syntax coloring, table of 280
- syntax highlighting
 - controlling within a window 280
- System Headers checkbox 178
- System Paths pane
 - in Access Paths settings panel 333
- System Paths radio button

- in Access Paths settings panel (Windows) 332
- System Relative Path 269, 336, 349
- System Requirements 29
- system requirements 29–31
 - for Linux IDE 31
 - for Mac IDE 30
 - for Solaris IDE 31
 - for Windows IDE 30

T

- tab control
 - displaying a tab view 580
- tab control, in browser (RAD) 579
- Tab Indents Selection checkbox
 - of Font & Tabs preference panel 279
- Tab Inserts Spaces checkbox
 - of Font & Tabs preference panel
 - Font & Tabs preference panel
 - Tab Inserts Spaces checkbox 279
- Tab Settings section
 - of Font & Tabs preference panel 278
- Tab Size field
 - of Font & Tabs preference panel 279
- tab view
 - displaying 580
 - limitations of 580
- tabs
 - inserting spaces instead of 279
 - setting size of 279
- target
 - build target defined 91
 - changing the build target name 95
 - changing the build target settings 96
 - defined 18
- Target column 47, 98
- Target Name field
 - in Target Settings panel 328
- Target Settings command 375, 377, 378, 626
- Target Settings panel 327
 - Choose button 329
 - Clear button 329
 - Linker pop-up menu 328
 - Output Directory field 329
 - Post-Linker pop-up menu 329
 - Pre-Linker pop-up menu 329
 - Save Project Entries Using Relative Paths
 - checkbox 329

-
- Target Name field 328
 - Target Settings window
 - build target configurations 326
 - choosing settings 324–360
 - code generation configurations 352
 - debugger configurations 357
 - dialog box buttons 323
 - discarding changes 323
 - editor configurations 356
 - settings panels 322
 - Source Trees panel 345
 - Targets
 - IDE 28
 - targets
 - assigning files to build targets 98
 - creating build target dependencies 97
 - creating build targets 94
 - platform target documentation 27
 - setting the build target 96
 - Targets section
 - of New Class wizard 236
 - Targets section, of New Event Set wizard 596
 - Targets tab, in Project window
 - Project window
 - Targets tab 533
 - Targets view 43
 - and designs 533
 - of Project window 51
 - Task pane 419
 - tasks
 - viewing 417
 - templated functions
 - setting breakpoints 462
 - templates
 - auto-completion of names 248
 - temporary breakpoints 449, 460
 - effect on regular breakpoints 460
 - setting 460
 - temporary variables 500
 - Text
 - drag and drop of 154
 - text
 - adding in editor 151
 - deleting in editor 151
 - indenting automatically 279
 - shifting left and right 155
 - Text Editing Area 139
 - Text File List option (Mac OS) 103
 - text formats
 - saving files in 277
 - text replace
 - Replace All 187
 - replacing found text 185
 - selective replace 186
 - single file 182–188
 - text replacing
 - in multiple files 189–197
 - text search 198–206
 - activating multi-file 189
 - Batch search 187
 - choosing file sets 193
 - choosing files 190
 - controlling range 184
 - controlling search parameters 184
 - controlling search range 196
 - finding selection 181
 - finding text 182
 - for selection 181
 - multi-file 175–179
 - multiple file 189–197
 - regular expressions 198–206
 - removing file sets 197
 - saving file sets 194
 - selected text search 180
 - single file 182–188
 - Text View pop-up menu
 - Auto option 429
 - Bitfield Description option 429
 - of Register Details window 429
 - Register Description option 429
 - Register Details option 429
 - TextArea component 570
 - TextField component 570
 - texttop option, of Java Applet wizard 520
 - THINK Reference 166, 263
 - THINK Reference (Mac OS) 166
 - third-party
 - disabling and enabling plugins 265
 - third-party editor support 260
 - third-party editors 90
 - third-party text editors
 - Emacs 260
 - third-party tools 39
 - threads
 - viewing 417
 - __throw() 642
-

Index

- Tile Editor Windows command 646
- Tile Editor Windows Vertically command 646
- Tile Vertical command 646
- Timeout field
 - of Java Debugging preference panel 292
- tip conventions 21
- Title field, of Java Applet wizard 519
- Toggle Index View button 552
- Tool Set Used In Makefile pop-up menu 67
- toolbar
 - in browser 218
 - of Component Catalog window 551
 - of Component Palette 548
 - of debugger 403
- Toolbar Disclosure Button 144
- Toolbar Submenu 647, 653–655
 - Anchor Floating Toolbar command (Mac OS) 655
 - Clear Floating Toolbar command 655
 - Clear Window Toolbar command 654
 - Hide Floating Toolbar command 654
 - Hide Window Toolbar command 654
 - Reset Floating Toolbar command 655
 - Reset Window Toolbar command 654
 - Show Floating Toolbar command 654
 - Show Window Toolbar command 654
 - Unanchor Floating Toolbar command (Mac OS) 655
- toolbars
 - adding elements 318
 - anchoring floating toolbar 320
 - customizing 313
 - elements 315
 - floating toolbar (Mac OS) 316
 - main toolbar (Windows) 316
 - modifying 318
 - Project window toolbar 43
 - removing all elements 319
 - removing elements 319
 - restoring default settings 320
 - showing and hiding 315
 - showing and hiding in editor windows 146
 - types 314
 - window toolbar 317
- Toolbox Assistant, Macintosh Programmer's* 165
- tools
 - third-party 39
- tools for RAD 510–511
- Tools Menu (Mac OS) 655–657
 - Commando command 656
 - Execute as ToolServer Script command 656
 - Insert Template command 657
 - Lookup Symbol command 657
 - Start ToolServer command 656
 - Stop ToolServer command 656
- Tools Submenu 653–655
- ToolServer
 - tools 657
- ToolServer Worksheet command (Mac OS) 648
- top option, of Java Applet wizard 520
- Touch column 47, 89
- Touch command 89
- touching
 - files 89
- Transient checkbox, in New Property dialog box 586
- Treat `#include <.../>` as `#include "..."` option 332
- troubleshooting
 - breakpoints 496, 497
 - bus error 497
 - changing variable values 498
 - Debug command 495
 - debugger 493–508
 - Enable Debugging 494
 - error on launch 495
 - launching debugger 494
 - no source code 502
 - outdated source files 503
 - strange data types 500
 - strange variable names 500
 - undefined identifier 501
 - unrecognized data types 501
 - variable does not change 498
 - variables 498
- tutorial resources 27
- Type applet parameter, of Java Applet wizard 521
- Type field
 - in Source Trees preference panel 267
 - in Source Trees settings panel 347
 - of New Data Member wizard 241
- Type field shortcut, in Java Applet wizard 521
- Type field, in New Property dialog box 586
- Type field, of New Property dialog box 585
- Type pop-up menu
 - Absolute Path option 267, 347

Environment Variable option 267, 347
 Registry Key option 268, 347
 type. *See* data types.
 typedefs
 auto-completion of names 248
 typeface conventions 21
 types
 of project files 52
 typing
 balancing parentheses, brackets, and
 braces 275
 typographical conventions 20

U

Unanchor Floating Toolbar command 320
 Unanchor Floating Toolbar command
 (Mac OS) 655
 undefined identifier 501
 understanding
 plugin compilers 362
 undo and redo
 series of actions 276
 Undo Checkout (VCS operation) 608
 Undo command 623, 624
 undoing
 multiple actions 156
 the previous action 156
 Ungroup command 544, 650
 UNI Software
 Voodoo plug-in 602
 unicode string
 viewing data as 473
 Unicode String command 644
 UNIX text files 121
 Unlock command 558
 Unselect tool 549
 unsigned decimal
 viewing data as 473
 Unsigned Decimal command 643
 untouching
 files 89
 updating projects 365, 366
 Use Archive (JAR) field, of Java Applet wizard 520
 Use BBEdit™ Extensions checkbox
 of IDE Extras preference panel (Mac OS) 262
 Use External Editor checkbox
 of IDE Extras preference panel 261
 Use Modification Date Caching checkbox
 of Build Extras settings panel 337
 Use Multiple Document Interface checkbox
 of IDE Extras preference panel (Windows) 260
 Use Multiple Undo checkbox
 of Editor Settings preference panel 276
 Use Script Menu checkbox
 of IDE Extras preference panel (Mac OS) 262
 Use separate file for member definitions checkbox
 of New Class wizard 233
 Use Third Party Debugger checkbox
 of Build Extras settings panel (Windows) 339
 Use Third Party Editor 260
 Use Third Party Editor checkbox
 of IDE Extras preference panel (Windows) 260
 Use ToolServer Menu checkbox
 of IDE Extras preference panel (Mac OS) 262
 Use Version Control option 604
 User Paths pane
 in Access Paths settings panel 333
 User Paths radio button
 in Access Paths settings panel (Windows) 332
 user-defined roots 266, 345
 Username field 605
 using
 Browser menu 229
 expressions 485
 Register Details window 426
 the debugger 400–401

V

values
 changing variable value 475
 Variable applet parameter, of Java Applet
 wizard 521
 Variable Change Hilite option
 of Display Settings preference panel 284
 Variable window 422, 435
 automatic closing of 435
 variables
 changing values 475
 creating environment 340
 data formats 475
 deleting environment 341
 expanding 405, 436, 469
 global 421, 470

Index

- in Expressions window 476
 - in separate windows 471
 - local 469, 477
 - meaning of \$ 500
 - modifying environment 341
 - opening a window for 422
 - static 421
 - strange names 500
 - temporary 500
 - troubleshooting 498
 - viewing information while debugging 415
- Variables (VCS operation) 608
- Variables pane 404, 469
 - Variables:All listing 405
 - Variables:Auto listing 405
- Variables/
 - All 285
 - Auto 285
- Variables:All 405
- Variables:Auto 405
- VCS 38
- VCS menu 606
- VCS Message window 613
- VCS Pop-up menu 609, 611
- VCS pop-up menu 223
- VCS Setup panel 603, 604
- Vectorization 356
- Version Control Login window 609
- Version Control Settings (VCS option) 604
- Version Control Settings command 626
- Version Control System (VCS)
 - activation 603
 - file permissions
 - Checked out 610
 - Locked 611
 - Modify Read-Only 610
 - Read-Only 610
 - Unlocked 611
- IDE plugins
 - ClearCase 602
 - CVS 602
 - Perforce 602
 - Projector 602
 - PVCS 602
 - SourceSafe 602
 - VOODOO 602
- installation 603
- Menu 651
- menus
 - VCS menu 606
 - VCS Pop-up menu 609
- operations
 - About 608
 - Add 608, 611
 - Checkin 608, 612
 - Checkout 608, 611
 - Comment 608
 - Connect 608
 - Disconnect 608
 - Get 607, 611
 - History 608
 - Label 608
 - Make Writable 612
 - Project 607
 - Properties 608
 - Recursion 607
 - Status 608
 - Synchronize Status 607
 - Undo Checkout 608, 611
 - Unlock 611
 - Variables 608
- windows
 - Project window 612
 - VCS Message Window 613
 - Version Control Login 609
- view
 - variables while debugging 415
- View Alphabetical command 567
- View Array command 422, 436, 643
- View As command 643
- View As Implementor command 219
- View As Subclass command 219
- View As User command 219
- View command 557
- View In Groups command 567
- View Memory As command 436, 438, 478, 643
- View Memory command 437, 478, 643
- View Variable command 422, 643
- viewers, documentation 27
- viewing
 - breakpoints 460
 - call chain 404
 - code as assembly language 396, 407, 415
 - code as mixed 409, 415
 - data as different types 474
 - data as multiple types 477

-
- data in debugger 469
 - data in different formats 473
 - data types 472
 - debugger toolbar 403
 - expressions 420
 - global variables 421, 470
 - local variables 404, 469
 - member functions and data members in
 - browser 218
 - memory
 - viewing memory dump 471
 - memory at an address 478
 - pointer types 474
 - processor registers 480
 - raw memory 477
 - register details 424
 - registers 406, 408, 430, 431, 432
 - stack contents 479
 - threads and tasks 417
 - watchpoints 468
 - Virtual destructor checkbox
 - of New Class wizard 235
 - Virtual option, of Specifier pop-up menu 239
 - Visual Bean option, of Java Bean wizard 530
 - Visual C, converting nmake files to projects 53
 - Visual Component option, of New Event Set wizard 596
 - Visual Sourcesafe, see SourceSafe 602
 - Volatile checkbox
 - of New Data Member wizard 242
 - Volatile checkbox, in New Property dialog box 586
 - VOODOO VCS plugin 602
 - VSpace field, of Java Applet wizard 520
- W**
- Warning Button 380
 - warning conventions 20
 - Warning Messages 383
 - watchpoint
 - defined 465
 - restrictions on 467
 - setting 466
 - viewing 468
 - Watchpoint Hilite option
 - of Display Settings preference panel 284
 - watchpoints
 - clearing 424
 - limitations (Mac OS) 466
 - on 68K machines 466
 - setting and clearing 466
 - viewing 468
 - Watchpoints window 424
 - Watchpoints window 424
 - Watchpoints Window command 424, 468, 649
 - “Weak” link 392
 - Width field, of Java Applet wizard 520
 - wildcard searching 198–206
 - Win32/x86 28
 - Window Menu 645–650
 - Breakpoints Window command 649
 - Browser Contents command 647
 - Build Progress Window command 648
 - Class Hierarchy Window command 647
 - Collapse Window command (Mac OS) 646
 - Component Catalog command 649
 - Component Palette command 649
 - Errors & Warnings Window command 648
 - Expand Window command (Mac OS) 646
 - Expressions Window command 648
 - Global Variables Window command 649
 - Minimize Window command (Windows) 646
 - New Class Browser command 647
 - Object Inspector command 649
 - Processes Window command 648
 - Project Inspector command 648
 - Registers Windows command 649
 - Restore Window command (Windows) 646
 - Save Default Window command 647
 - Show Catalog Window command 647
 - Stack Editor Windows command 645
 - Tile Editor Windows command 646
 - Tile Editor Windows Vertically command 646
 - Tile Vertical command 646
 - Toolbar submenu 647
 - ToolServer Worksheet command (Mac OS) 648
 - Watchpoints Window command 649
 - Zoom Window command 646
 - Window Position And Size checkbox
 - of Editor Settings preference panel 275
 - Windowing preference panel 286
 - Close Non-Debugging Windows radio button 287
 - Collapse Non-Debugging Windows radio button 287
-

Index

- Do Nothing To Non-Debugging Windows
 - radio button 286
 - Hide Non-Debugging Windows radio
 - button 286
 - Minimize Non-Debugging Windows radio
 - button (Windows) 287
 - Monitor For Debugging pop-up menu
 - (Mac OS) 287
 - Move Open Windows To Debugging Monitor
 - When Debugging Starts checkbox
 - (Mac OS) 288
 - Open Windows On Debugging Monitor
 - During Debugging checkbox (Mac OS) 288
 - Windows
 - system requirements 30
 - windows
 - class browser 216–224
 - controlling syntax highlighting within 280
 - creating panes 147
 - Expressions window 420
 - Global Variables window 421
 - Program window 402
 - Project window Design view 42, 50
 - Project window Files view 42, 44
 - Project window Link Order view 42, 51
 - Project window navigation 43
 - Project window Targets view 43, 51
 - Project window toolbar 43
 - removing panes 148
 - resizing panes 148, 402
 - Source Code Browser 410
 - Stack Crawl window 402
 - WinHelp (Windows) 164
 - wizards
 - for Browser 230–244
 - for RAD 515–531
 - for RAD layouts 544–546
 - for RAD projects 512
 - Java Applet 516
 - Java Applet Wizard 512, 531
 - Java Application 523
 - Java Application Wizard 512, 531
 - Java Bean 527
 - Java Bean Wizard 512, 531
 - Java Frame 545
 - Java Frame Wizard 538
 - New Class 231
 - New Data Member 241
 - New Event Set 591
 - New Member Function 237
 - wizards, for RAD projects 53
 - working
 - with complex projects 91–99
 - Working Directory field
 - in Runtime Settings panel (Windows) 340
 - working with projects 41
 - Write button
 - of Register Details window 429
- ## X
- x86 Settings preference panel 295
 - Port ID field (Mac OS) 296
 - Remote Debugging checkbox (Mac OS) 296
 - Remote IP Address field (Mac OS) 296
 - XML
 - files for Register Details window 425
- ## Y
- Y2K. *See* Year 2000.
 - Year 2000 (Y2K)
 - additional information 24
 - and CodeWarrior compliance issues 24
- ## Z
- Zoom Window command 646
 - Zoom Windows To Full Screen checkbox
 - of IDE Extras preference panel 261

CodeWarrior

IDE User Guide

Credits

writing lead:	Derek Saldaña
other writers:	Caresse Bennett, David Blache, Alisa Dean, Chris Magnuson, Roopa Malavally, Marc Paquette, Jim Trudeau, L. Frank Turovich
engineering:	Lubo Antonov, Kevin Bell, Greg Bolsinga, Kenny Drycksback, David Dubrow, Matt Henderson, Michael Marks, Glenn Meter, Mathavi Paramasivam, Dan Podwall, Max Requesnes, Rob Vaterlaus
quality assurance	Joe Hayden, Patrick Sullivan, Isaac Wankerl, Josef Wankerl
frontline warriors:	Richard Atwell, John Cortell, Ron Liechty, Todd McDaniel, Jim Trudeau, Roger Wong, Ben Kenobi, and CodeWarrior users everywhere

