

CodeWarrior™

Development Studio

IDE 5.7 User's Guide

Revised 25 May 2005

metrowerks

Metrowerks and the Metrowerks logo are registered trademarks of Metrowerks Corporation in the United States and/or other countries. CodeWarrior is a trademark or registered trademark of Metrowerks Corporation in the United States and/or other countries. All other trade names and trademarks are the property of their respective owners.

Copyright © 2004 Metrowerks Corporation. ALL RIGHTS RESERVED.

No portion of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Metrowerks. Use of this document and related materials are governed by the license agreement that accompanied the product to which this manual pertains. This document may be printed for non-commercial personal use only in accordance with the aforementioned license agreement. If you do not have a copy of the license agreement, contact your Metrowerks representative or call 1-800-377-5416 (if outside the U.S., call +1-512-996-5300).

Metrowerks reserves the right to make changes to any product described or referred to in this document without further notice. Metrowerks makes no warranty, representation or guarantee regarding the merchantability or fitness of its products for any particular purpose, nor does Metrowerks assume any liability arising out of the application or use of any product described herein and specifically disclaims any and all liability. **Metrowerks software is not authorized for and has not been designed, tested, manufactured, or intended for use in developing applications where the failure, malfunction, or any inaccuracy of the application carries a risk of death, serious bodily injury, or damage to tangible property, including, but not limited to, use in factory control systems, medical devices or facilities, nuclear facilities, aircraft navigation or communication, emergency systems, or other applications with a similar degree of potential hazard.**

How to Contact Metrowerks

Corporate Headquarters	Metrowerks Corporation 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	http://www.metrowerks.com
Sales	United States Voice: 800-377-5416 United States Fax: 512-996-4910 International Voice: +1-512-996-5300 E-mail: sales@metrowerks.com
Technical Support	United States Voice: 800-377-5416 International Voice: +1-512-996-5300 E-mail: support@metrowerks.com

Table of Contents

I Introduction

1 IDE User's Guide Overview	17
Release Notes	17
Licensing	17
CodeWarriorU.com	18
Documentation Structure	20
Documentation Formats	20
Documentation Types	21
Manual Conventions	21
Figure Conventions	22
Keyboard Conventions	22
Special note for Solaris and Linux users	22

2 CodeWarrior IDE Overview	23
Development Cycle	23
CodeWarrior IDE Advantages	25
IDE Tools Overview	26

II Projects

3 Working with Projects	31
About Projects	31
Project Manager	31
Build Targets	33
Managing Projects	35
Advanced Projects	41
Custom Project Stationery	41

Table of Contents

Subprojects	42
Strategies	43
4 Project Window	45
About the Project Window	45
Project Window Pages	47
Files Page	47
Link Order Page	51
Targets Page	52
File, Group, Layout, and Target Management	53
Build-Target Management	58
5 Working with Files	61
Managing Files	61
6 Dockable Windows	67
About Dockable Windows	67
Working with Dockable Windows	70
Dock Bars	76
7 Workspaces	79
About Workspaces	79
Using Workspaces	79
8 Creating Console Applications	83
About Console Applications	83
Creating Console Applications	84
III Editor	
9 The CodeWarrior Editor	89
Editor Window	89
Editor Toolbar	92

Table of Contents

Interfaces Menu	92
Functions Menu	92
Markers Menu	93
Document Settings Menu	93
Version Control System Menu	93
Other Editor Window Components	94
Path Caption	94
File Modification Icon	94
Breakpoints Column	95
Text Editing Area	95
Line and Column Indicator	95
Pane Splitter Controls	95
10 Editing Source Code	97
Text Manipulation	97
Symbol Editing Shortcuts	100
Punctuation Balancing	100
Code Completion	102
Code Completion Configuration	103
Code Completion Window	105
11 Navigating Source Code	113
Finding Interface Files, Functions, and Lines	113
Finding Interface Files	114
Locating Functions	114
Going Back and Forward	116
Using Markers	116
Remove Markers Window	117
Symbol Definitions	119
Reference Templates (Macintosh)	120
12 Finding and Replacing Text	123
Single-File Find	123
Single-File Find and Replace	126
Multiple-File Find and Replace	129

Table of Contents

In Folders	131
In Projects	134
In Symbolics	136
In Files	138
Search Results Window	141
Text-Selection Find	143
Regular-Expression Find	145
Using the Find String in the Replace String	147
Remembering Sub-expressions	148
Comparing Files and Folders	148
Comparison Setup	149
File Comparison	152
Folder Comparison	155

IV Browser

13 Using the Browser	161
Browser Database	161
Browser Data	162
Browser Symbols	164
Browser Contextual Menu	164
14 Using Class Browser Windows	167
Class Browser window	167
Classes pane	174
Member Functions pane	175
Data Members pane	176
Source pane	177
Status Area	178
15 Using Other Browser Windows	179
Multiple-Class Hierarchy Window	179
Single-Class Hierarchy Window	182

Table of Contents

Browser Contents window	183
Symbols window	185
Symbols toolbar	187
Symbols pane	187
Source pane	187
16 Using Browser Wizards	189
The New Class Wizard	189
The New Member Function Wizard	194
The New Data Member Wizard	197
V Debugger	
17 Working with the Debugger	203
About the CodeWarrior Debugger	203
About Symbolics Files	204
Thread Window	204
Common Debugging Actions	208
Symbol Hint	211
Contextual Menus	212
Multi-core Debugging	214
Data Viewer Plug-ins	214
External Builds Support	215
External Build Wizard	218
18 Manipulating Program Execution	221
Breakpoints	222
Breakpoints Window	222
Working with Breakpoints	226
Working with Breakpoint Templates	231
Eventpoints	233
Log Point	235
Pause Point	236

Table of Contents

Script Point	237
Skip Point	238
Sound Point (Windows OS)	239
Trace Collection Off	240
Trace Collection On	241
Working with Eventpoints	242
Watchpoints	244
Special Breakpoints	248
19 Working with Variables	249
Global Variables Window	249
Variable Window	251
Expressions Window	254
20 Working with Memory	259
Memory Window	259
Array Window	264
Registers Window	266
General Registers	267
FPU Registers	267
Host-specific Registers	267
Register Details Window (Windows OS)	270
Description File	273
Register Display	274
Text View	274
21 Working with Debugger Data	275
Symbolics Window	275
System Browser Window	278
Log Window	281
22 Working with Hardware Tools	285
Flash Programmer Window	285
Target Configuration	287
Flash Configuration	289

Table of Contents

Program / Verify	291
Erase / Blank Check	294
Checksum	295
Hardware Diagnostics Window	297
Configuration	299
Memory Read / Write	300
Scope Loop	301
Memory Tests	303
Working with a Logic Analyzer	309
Configuring the Project	309
Using the Logic Analyzer	312
Trace Window	314
Cache Window	315
Profile Window	316
Command Window	317

23 Profiler 319

Overview	319
What Is a Profiler?	320
Types of Profilers	320
A Profiling Strategy	322
Profiling Code	322
Using the Profiler	323
What It Does	323
How It Works	324
Profiling Made Easy	325
Configuring	329
Profiler Libraries and Interface Files	329
Profiling Special Cases	329
Viewing Results	335
What It Does	335
How It Works	335
Finding Performance Problems	340
Troubleshooting	341
Reference	343

Table of Contents

Compiler Directives	343
Memory Usage	344
Time and Timebases	345
Profiler Function Reference	345

VI Compilers and Linkers

24 Compilers 353

Choosing a Compiler	353
Compiling Projects	354

25 Linkers 359

Choosing Linkers	359
Linking Projects	360

VII Preferences and Target Settings

26 Customizing the IDE 363

Customizing IDE Commands	363
Commands Tab	365
Pre-defined Variables in Command Definitions	368
Customize Toolbars	373
Kinds of Toolbars	374
Toolbar Elements	375
Modify a Toolbar	375
Customize Key Bindings	379

27 Working with IDE Preferences 385

IDE Preferences Window	385
General Panels	387
Build Settings	387

Table of Contents

Concurrent Compiles	389
IDE Extras	390
Help Preferences	393
Plugin Settings	394
Shielded Folders	394
Source Trees	396
Editor Panels	399
Code Completion	400
Code Formatting	401
Editor Settings	403
Font & Tabs	406
Text Colors	408
Debugger Panels	412
Display Settings	412
Window Settings	414
Global Settings	416
Remote Connections	418

28 Working with Target Settings 423

Target Settings Window	423
Target Panels	426
Target Settings	426
Access Paths	427
Build Extras	430
Runtime Settings	432
File Mappings	434
Source Trees	437
Code Generation Panels	437
Global Optimizations	437
Editor Panels	440
Custom Keywords	441
Debugger Panels	442
Other Executables	443
Debugger Settings	445
Remote Debugging	447

Table of Contents

29 Preference and Target Settings Options	449
--------------------------------------------------	------------

VIII Menus

30 IDE Menus	493
---------------------	------------

Windows Menu Layout	493
File Menu	493
Edit Menu	495
View Menu	497
Search Menu	498
Project Menu	500
Debug Menu	502
Data Menu	504
Window Menu	506
Help Menu	506
Macintosh Menu Layout	507
Apple Menu	507
CodeWarrior Menu	507
File Menu	508
Edit Menu	510
Search Menu	511
Project Menu	513
Debug Menu	515
Data Menu	517
Window Menu	519
VCS Menu	521
Tools Menu	521
Scripts Menu	522
Help Menu	522

Table of Contents

31 Menu Commands	523
A Register Details Window XML Specification	559
Register Details Window XML Specification	559
Accessing the XML Files from CodeWarrior	568
A Sample XML File	569
Creating the New XML File	569
Adding Multiple BITFIELD Attributes	570
Adding BFVALUE Attributes	572
Completing the New XML File	574
References	577
Index	579

Table of Contents



Introduction

This section contains these chapters:

- [IDE User's Guide Overview](#)
- [CodeWarrior IDE Overview](#)

IDE User's Guide Overview

This chapter of the *CodeWarrior™ IDE User's Guide* is a high-level description of documentation and training resources for learning to use the IDE.

- [“CodeWarriorU.com” on page 18](#)—free, Internet-based instruction for CodeWarrior products. Use this resource to learn more about the CodeWarrior Integrated Development Environment (IDE) and computer programming.
- [“Documentation Structure” on page 20](#)—a guide to the various CodeWarrior manuals available. This guide notes the location of generic and specific product documentation.
- [“Manual Conventions” on page 21](#)—some common typographical conventions used in this manual and other Metrowerks documentation.

Release Notes

Please read the release notes. They contain important last-minute additions to the documentation. The Release Notes folder is located on the CodeWarrior CD.

Licensing

Web-based licensing is available. It is a server licensing solution that generates FlexLM v8 or later based license keys automatically over the world wide web through a registration/activation process. You can register and activate permanent, node-locked license keys.

Metrowerks products are shipped to customers with registration cards that contain a unique registration number. Products that ship with a one year annual support certificate will also have a unique registration number.

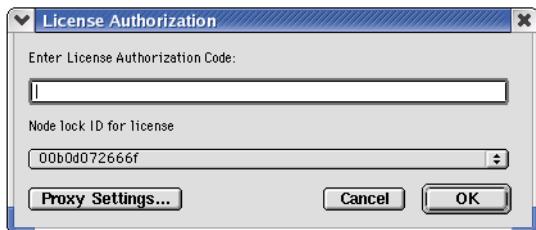
IDE User's Guide Overview

CodeWarriorU.com

During product installation you will be instructed to register at `http://www.metrowerks.com/mw/register`. You can also reach the registration website by selecting the **Help > Register Product** menu command from the IDE's main menu. Registration from the website collects the registration code and verifies it against the correct product and gathers contact information.

An email will be sent to you with the License Authorization Code and instructions. In the IDE you can select **Help > License Authorization...** to display the License Authorization dialog box. [Figure 1.1](#) shows the License Authorization dialog box.

Figure 1.1 License Authorization



Enter the License Authorization Code and select an ethernet address from the Node lock ID for license dropdown list, if one exists. After entering the authorization code, the CodeWarrior IDE will make an HTTP call to the Metrowerks licensing server with the activation code and generate the permanent license keys. The resulting license keys are automatically updated into the license.dat text file of the CodeWarrior product executing the authorization. You can also manually edit the license.dat file per instructions provided in the License_Install.txt file in the root folder of your CodeWarrior installation path. If the IDE evaluation period expires prior to activation, you will have to manually edit the license.dat file.

CodeWarriorU.com

CodeWarriorU.com offers a wide range of free, Internet-based courses in a wide variety of computer programming topics. Use this supplement to the

CodeWarrior documentation to acquire more experience using CodeWarrior products.

IDE User's Guide Overview

Documentation Structure

CodeWarriorU.com courses include:

- Text-based instruction
- Expert instructors
- A variety of self-assessment and study materials
- Interactive message boards for communicating with instructors and students

CodeWarriorU offers many courses, such as:

- Learn Programming in C
 - For beginning programmers.
- Introduction to Java
 - For beginning and experienced programmers. Take this course to learn how to create Java software.
- Introduction to C++
 - For beginning and experienced programmers. Take this course to learn how to create C++ software.
- Intermediate C++
 - For programmers who completed the Introduction to C++ course and have basic C++ programming knowledge. Take this course to learn the foundation needed to create more sophisticated C++ software.

To find out more, visit this web site:

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

Documentation Structure

CodeWarrior products include an extensive documentation library of user guides, targeting manuals, and reference manuals. Take advantage of this library to learn how to efficiently develop software using the CodeWarrior programming environment.

Documentation Formats

CodeWarrior documentation presents information in various formats:

- **Print**—Printed versions of CodeWarrior manuals, including the *IDE User's Guide*, *MSL C Reference*, *C/C++ Reference*, and product-focused Targeting manuals.

- **PDF** (Portable Document Format)—Electronic versions of CodeWarrior manuals. The CodeWarrior CD Documentation folder contains the electronic PDF manuals.
- **HTML** (Hypertext Markup Language)—HTML or Compressed HTML (.CHM) versions of CodeWarrior manuals.

Documentation Types

Each CodeWarrior manual focuses on a particular information type:

- **User guides**—User guides provide basic information about the CodeWarrior user interface. User guides include information that supports all host platforms on which the software operates, but do not include in-depth platform-specific information.
- **Targeting manuals**—Targeting manuals provide specific information required to create software that operates on a particular platform or microprocessor. Examples include the *Targeting Windows*, *Targeting Java*, and *Targeting DSP56800* manuals.
- **Reference manuals**—Reference manuals provide specialized information that supports coding libraries, programming languages, and the IDE. Examples include the *C Compiler Reference*, *MSL C Reference*, and *Extending the CodeWarrior IDE* manuals.
- **Core manuals**—Core manuals explain the core technologies available in the CodeWarrior IDE. Examples include:
 - *IDE User's Guide*
 - *C/C++ Compilers Reference*
 - *MSL C Reference and MSL C++ Reference*
 - *Extending the CodeWarrior IDE*
 - *Command-Line Tools Reference*

Manual Conventions

This section explains conventions in the *IDE User's Guide*.

Figure Conventions

The CodeWarrior IDE employs a virtually identical user interface across multiple hosts. For this reason, illustrations of common interface elements use images from any host. However, some interface elements are unique to a particular host. In such cases, clearly labelled images identify the specific host.

Keyboard Conventions

The CodeWarrior IDE accepts keyboard shortcuts, or *key bindings*, for frequently used operations. For each operation, this manual lists corresponding key bindings by platform. Hyphens separate multiple keystrokes in each key binding.

Special note for Solaris and Linux users

The Solaris and Linux IDE use Macintosh symbols to represent modifier keys in key bindings. [Table 1.1](#) shows the relationship between the Macintosh symbols and the equivalent modifier keys on Solaris and Linux computers. Solaris and Linux computers can map a modifier key to any key on the keyboard. The preceding table reflects the default modifier key configuration for these computers. Remember that custom mappings supersede the default configuration noted in the table.

Table 1.1 Macintosh modifier-key equivalents for Solaris and Linux

Symbol	Macintosh Name	Solaris Equivalent	Linux Equivalent
	Control	Control	Ctrl
	Option	Alt	Alt
	Command	Meta	Alt
	Shift	Shift	Shift

CodeWarrior IDE Overview

The CodeWarrior™ Integrated Development Environment (IDE) provides an efficient and flexible software-development tool suite. This chapter explains the advantages of using the CodeWarrior IDE and provides brief descriptions of the major tools that make up the IDE.

This chapter contains these sections:

- [“Development Cycle” on page 23](#)
- [“CodeWarrior IDE Advantages” on page 25](#)
- [“IDE Tools Overview” on page 26](#)

Development Cycle

A software developer follows a general development process:

- Begin with an idea for new software
- Implement new idea in source code
- Have the IDE compile source code into machine code
- Have the IDE link machine code and form an executable file
- Correct errors (debug)
- Compile, link, and release a final executable file.

The stages of the development cycle correspond to one or more chapters in this manual.

[Figure 2.1 on page 24](#) depicts the development cycle as a flowchart. [Table 2.1 on page 25](#) details the different stages and their corresponding sections in this manual.

CodeWarrior IDE Overview

Development Cycle

Figure 2.1 The Development Cycle diagram

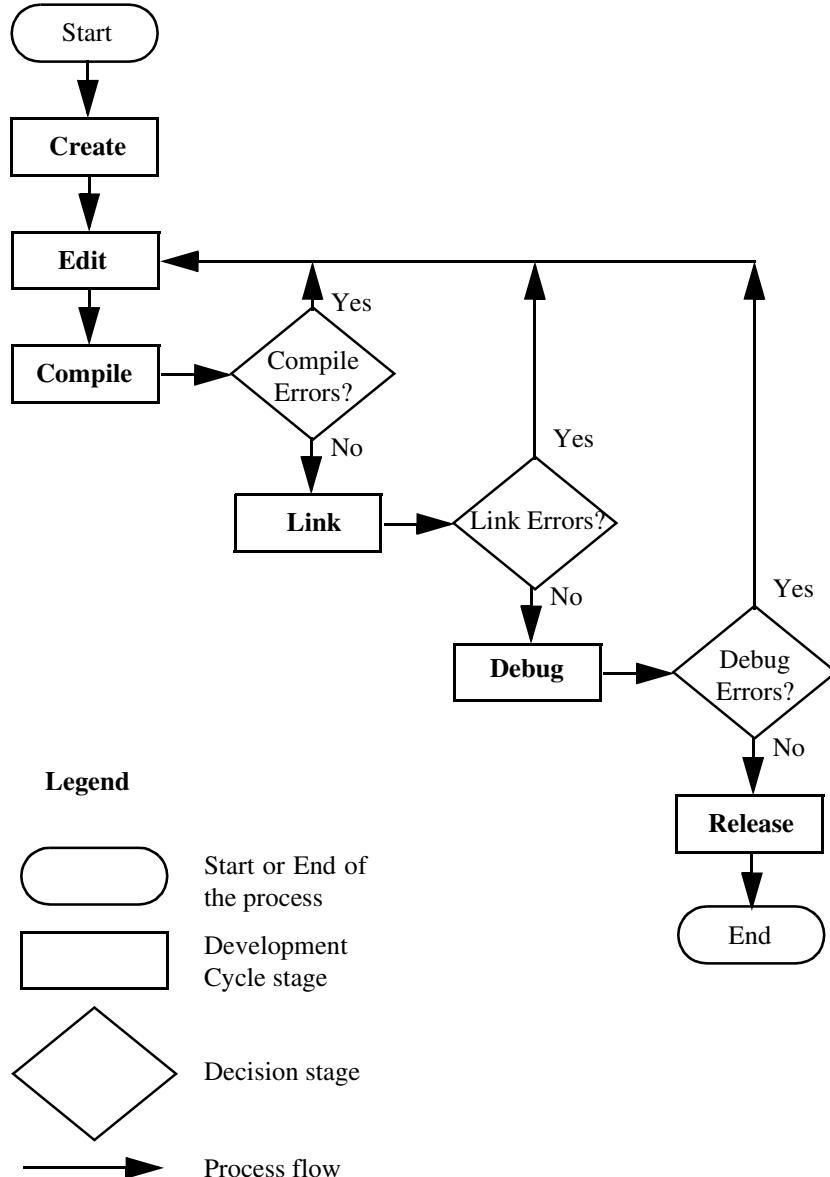


Table 2.1 Stage descriptions and related sections in the IDE User's Guide

Stage	Description	Related Sections
Create	Create the initial project, source files, and build targets.	<ul style="list-style-type: none"> • “Projects” on page 29 • “Preferences and Target Settings” on page 361 • “Menus” on page 491
Edit	Transform your project into working source code, organize interface elements, and correct errors.	<ul style="list-style-type: none"> • “Editor” on page 87 • “Browser” on page 159
Compile	Compile the source code into machine format that operates on the target host.	“Compilers and Linkers” on page 351
Link	Link the separate compiled modules into a single binary executable file.	“Compilers and Linkers” on page 351
Debug	Find and resolve all coding and logic errors that prevent the program from operating as designed.	“Debugger” on page 201
Release	Release for public use.	Beyond the scope of this manual.

CodeWarrior IDE Advantages

- Cross-platform development

Develop software to run on multiple operating systems, or use multiple hosts to develop the same software project. The IDE runs on popular operating systems, including Windows, Macintosh, Solaris, and Linux. The IDE uses virtually the same graphical user interface (GUI) across all hosts.
- Multiple-language support

Choose from multiple programming languages when developing software. The IDE supports high-level languages, such as C, C++, and Java, as well as in-line assemblers for most processors.

- Consistent development environment

Port software to new processors without having to learn new tools or lose an existing code base. The IDE supports many common desktop and embedded processor families, including x86, PowerPC, MIPS, and many others.

- Plug-in tool support

Extend the capabilities of the IDE by adding a plug-in tool that supports new services. The IDE currently supports plug-ins for compilers, linkers, pre-linkers, post-linkers, preference panels, version controls, and other tools. Plug-ins make it possible for the CodeWarrior IDE to process different languages and support different processor families.

IDE Tools Overview

The CodeWarrior IDE is a tool suite that provides sophisticated tools for software development. This section explains the standard tools available in the IDE:

- a project manager
- an editor
- a search engine
- a source browser
- a build system
- a debugger

[Table 2.2 on page 27](#) explains the purpose of these tools and lists corresponding CodeWarrior IDE features.

Table 2.2 IDE tools and features

Tool	Purpose	CodeWarrior IDE Features
Project Manager	Manipulate items associated with a project	<ul style="list-style-type: none"> Handles top-level file management for the software developer Organizes project items by major group, such as files and targets Tracks state information (such as file-modification dates) Determines build order and files to be included in each build Coordinates with plug-ins to provide version-control services
Editor	Create and modify source code	<ul style="list-style-type: none"> Uses color to differentiate programming-language keywords Allows definition of custom keywords for additional color schemes Automatically verifies parenthesis, brace, and bracket balance Allows use of menus for navigation to any function or into the header files used by the program
Search Engine	Find and replace text	<ul style="list-style-type: none"> Finds a specific text string Replaces found text with substitute text Allows use of regular expressions Provides file-comparison and differencing functionality
Source Browser	Manage and view program symbols	<ul style="list-style-type: none"> Maintains a symbolics database for the program. Sample symbols include names and values of variables and functions. Uses the symbolics database to assist code navigation Links every symbol to other locations in the code related to that symbol Processes both object-oriented and procedural languages
Build System	Convert source code into an executable file	<ul style="list-style-type: none"> Uses compiler to generate object code from source code Uses linker to generate final executable file from object code
Debugger	Resolve errors	<ul style="list-style-type: none"> Uses symbolics database to provide source-level debugging Supports symbol formats such as CodeView, DWARF (Debug With Arbitrary Records Format), and SYM (SYMbolic information format)



Projects

This section contains these chapters:

- [Working with Projects](#)
- [Project Window](#)
- [Working with Files](#)
- [Dockable Windows](#)
- [Workspaces](#)
- [Creating Console Applications](#)

Working with Projects

This chapter explains how to work with projects in the CodeWarrior™ IDE. Projects organize several file types associated with a computer program:

- Text files—files that contain any kind of text. Sample text files include Read Me files and source files.
- Source files—files that contain source code only. Sample source files include C++ files and Java files.
- Library files—files that contain special code designed to work together with a particular programming language or operating environment.
- Generated files—files created by the IDE while building or debugging the project.

This chapter contains these sections:

- [“About Projects” on page 31](#)
- [“Managing Projects” on page 35](#)
- [“Advanced Projects” on page 41](#)

About Projects

The IDE uses build targets and a Project Manager to organize source code and support files. This section explains both components.

Project Manager

The IDE gathers source, library, resource, and other files into a *project*. The Project Manager manipulates the information stored in the project.

[Figure 3.1 on page 32](#) diagrams Project Manager interactions with IDE tools. [Table 3.1 on page 32](#) explains the interactions.

Working with Projects

About Projects

Figure 3.1 Project Manager

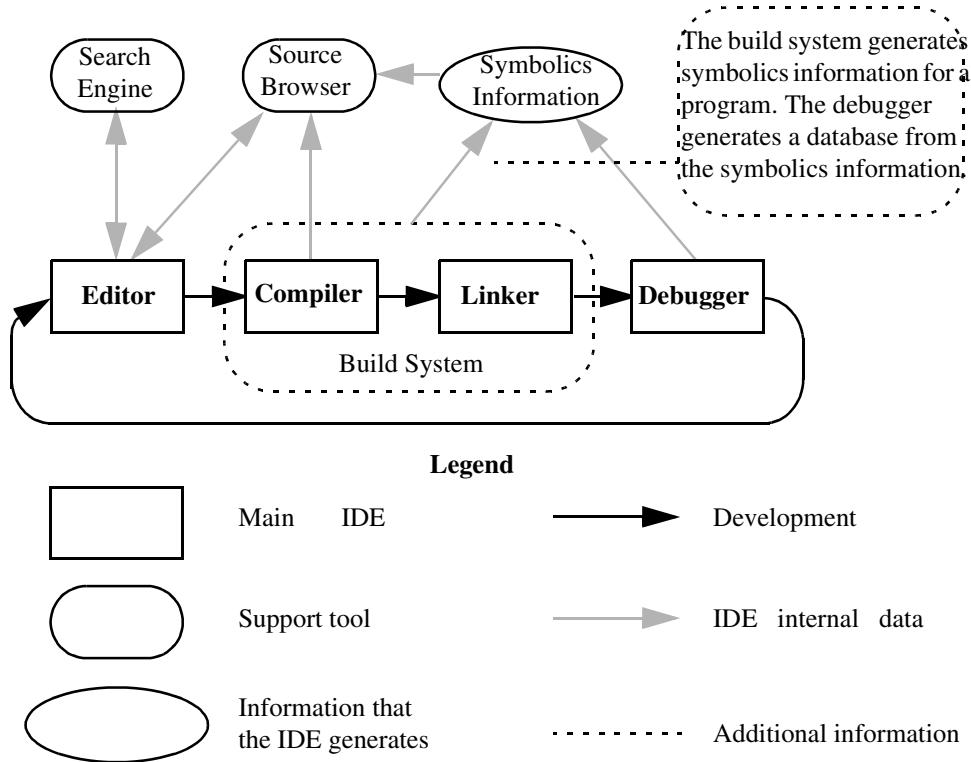


Table 3.1 Project Manager interactions

IDE Tool	Project Manager Interactions
Editor	<ul style="list-style-type: none">Coordinates internal data flow among editor windows, search engine, and source browserMatches find-and-replace results between related header files and source filesAssociates functions and variables with corresponding source code
Compiler	<ul style="list-style-type: none">Synchronizes a symbolics database of program functions, variables, and values with source codeCoordinates internal data flow between symbolics database and source browserDetermines files to include in build process

Table 3.1 Project Manager interactions (*continued*)

IDE Tool	Project Manager Interactions
Linker	<ul style="list-style-type: none"> Sends compiled object code to linker for conversion to executable code Sets the link order for processing compiled object code
Debugger	<ul style="list-style-type: none"> Matches debugging data to source code Updates symbolics database to reflect changing values during a debug session

Build Targets

For any given build, the project manager tracks:

- files and libraries
- link order
- dependencies
- compiler, linker, and other settings

The IDE stores this information in a *build target*. As the project changes, the project manager automatically updates the build target. The project manager also coordinates program builds, using the build-target information to call the appropriate tools in the correct order with the specified settings.

For example, the project manager directs the build system to compile only those source files that rely on information in a modified file.

Note that all of this operation happens automatically. The software developer does not need to remember makefile syntax or semantics, and never has to debug makefile syntax errors. The IDE simplifies the process, making it easier to develop software.

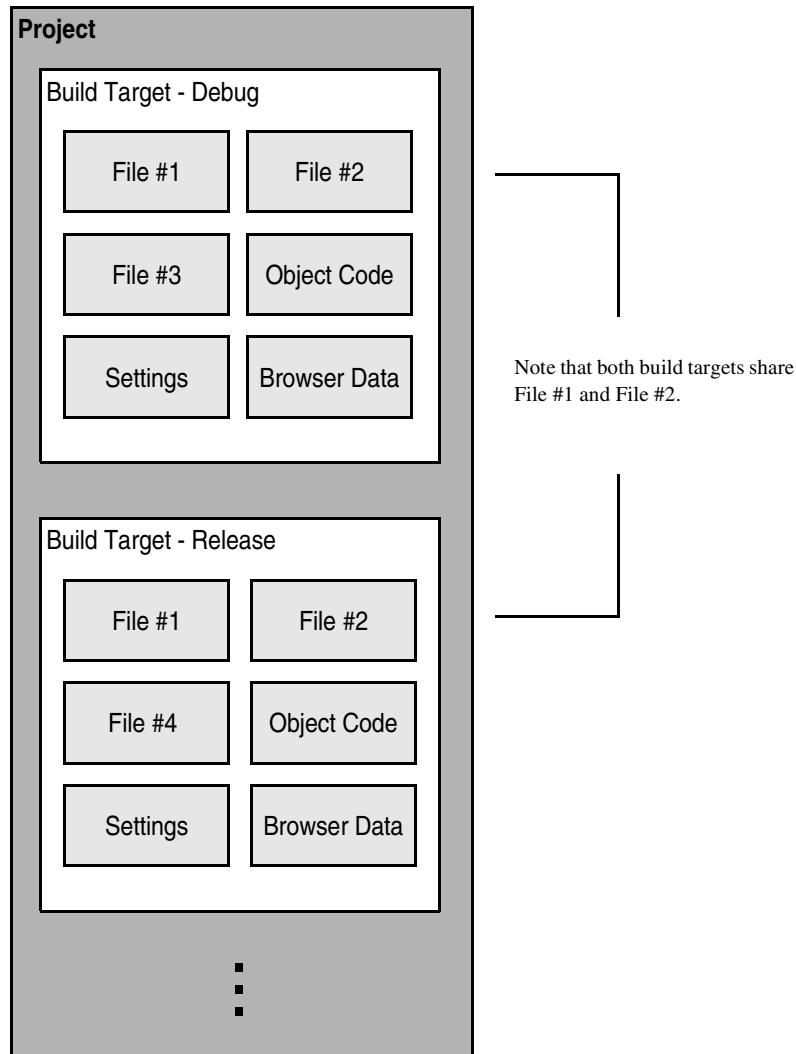
The project manager also supports multiple build targets within the same project file. Each build target can have its own unique settings, and even use different source and library files. For example, it is common to have both debug and release build targets in a project.

[Figure 3.2 on page 34](#) shows a sample project with debug and release build targets.

Working with Projects

About Projects

Figure 3.2 Project with multiple build targets



Managing Projects

Use these tasks to manage projects:

- Create a new project
- Open existing project
- Save project
- Close project
- Inspect an open project
- Print an open project

Creating New Projects using Project Stationery

Use the project stationery provided with the IDE to quickly create new projects. The stationery contains everything needed for a minimal, ready-to-run project. Use project stationery as a foundation upon which to add features for each new project.

1. Choose **File > New**.
2. Click the **Project** tab and select a project type.
3. Enter a project name (include the `.mcp` extension) in the **Project Name** field and set the **Location** for the new project.
4. Click **OK** in the **New** window.
5. Select the appropriate project stationery from the **New Project** window.
6. Click **OK** in the **New Project** window.

The IDE uses the selected stationery as a template to create a new project.

Creating New Projects from Makefiles

Some CodeWarrior products have a Makefile Importer wizard to convert most Visual C nmake or GNU make files into projects. The wizard performs these tasks:

- Parses the makefile to determine source files and build targets
 - Creates a project
 - Adds the source files and build targets determined during parsing
 - Matches makefile information, such as output name, output directory, and access paths, with the newly created build targets.
 - Selects a project linker
1. Choose **File > New**.
 2. Click the **Project** tab.
 3. Select **Makefile Importer Wizard**.
 4. Enter a project name (include the .mcp extension) in the **Project Name** field and set the **Location** for the new project.
 5. Click **OK** in the **New** window.
 6. Enter the path to the makefile in the **Makefile location** field or click **Browse** to navigate to the makefile.
 7. Choose the tool set used for makefile conversion and linker selection.
 - **Tool Set Used In Makefile**—Choose the tool set whose build rules form the basis of the makefile.
 - **Metrowerks Tool Set**—Choose the linker tool set to use with the generated project.
 8. Select the desired diagnostic settings.
 - **Log Targets Bypassed**—Select to log information about makefile build targets that the IDE fails to convert to project build targets.
 - **Log Build Rules Discarded**—Select to log information about makefile rules that the IDE discards during conversion.
 - **Log All Statements Bypassed**—Select to log targets bypassed, build rules discarded, and other makefile items that the IDE fails to convert.

9. Click **Finish**, then **Generate**.

The Makefile Importer wizard performs the conversion process and displays additional information.

Creating Empty Projects

Unlike project stationery, empty projects do not contain a pre-configured collection of template source files, library files, or build targets. Empty projects allow advanced software engineers to custom-build new projects from scratch.

TIP If working with an empty project is too intimidating, try modifying a project created with project stationery instead. Project stationery pre-configures complicated settings to quickly get started.

1. Choose **File > New**.
2. Click the **Project** tab and select **Empty Project**.
3. Enter a project name (include the **.mcp** extension) in the **Project Name** field and set the **Location** for the new project.
4. Click **OK** in the **New** window.

The IDE creates an empty project. Add files and libraries, create build targets, and choose the appropriate target settings to complete the new project.

Opening Projects

Use the IDE to open previously saved projects. CodeWarrior projects normally end in the *Metrowerks CodeWarrior Project* extension of **.mcp**. Open projects to add, remove, or modify files to enhance the capabilities of the final executable file.

1. Choose **File > Open**.
2. Find and select the project to open.
3. Click **Open**.

The IDE opens the project and displays its Project window.

Working with Projects

Managing Projects

NOTE The IDE prompts you for confirmation to update projects created in older CodeWarrior versions.

Opening Projects Created on Other Hosts

CodeWarrior projects whose names end in .mcp are cross-platform. However, the object code stored inside each project folder is not cross-platform. Use these procedures to properly open the project on a different host computer.

1. If not present, add the .mcp filename extension to the project name.
 2. Copy the project folder from the original host to the new host.
 3. Delete the Data folder inside the newly copied project folder.
 4. Open the newly copied project on the new host IDE.
 5. Recompile the project to generate new object code.
-

Saving Projects

The IDE automatically saves projects and updates project information after performing these actions:

- Closing the project
 - Applying or saving a preference or target-setting option
 - Adding, deleting, or compiling a file
 - Editing group information
 - Removing or compacting object code
 - Quitting the IDE
-

Inspecting Project Files

Use the **Project Inspector** command to review and configure source-file attributes and target information in the Project Inspector window.

1. Select a file in the Project window.
 2. Open the **Project Inspector** window, as explained in [Table 3.2](#).
-

Table 3.2 Opening the Project Inspector window

On this host...	Do this...
Windows	Select View > Project Inspector .
Macintosh	Select Window > Project Inspector .
Solaris	Select Window > Project Inspector .
Linux	Select Window > Project Inspector .

3. Examine the source-file attributes and target settings.
 - Click the **Attributes** tab to view the file attributes.
 - Click the **Targets** tab to view the build targets that use the file.
-

Printing Projects

The Project Manager can print a complete listing of the **Files**, **Designs**, **Link Order**, or **Targets** tab currently displayed in the Project window.

1. Select the Project window.
2. Click the **Files**, **Designs**, **Link Order**, or **Targets** tab.
3. Choose **File > Print**.
4. Set the print options in the print dialog.
5. Print the Project window contents.

The IDE prints the contents of the selected tab.

Choosing a Default Project

The IDE allows multiple open projects at the same time. However, a given source file can belong to more than one open project, making it ambiguous as to which project a source-file operation applies.

To resolve ambiguity, choose the default project to which the IDE applies operations.

1. If only one project is open, it automatically becomes the default project.

Working with Projects

Managing Projects

2. If more than one project is open, choose **Project > Set Default Project** to select the desired default project.

In ambiguous situations, the IDE applies operations to the selected default project.

Exporting Projects to XML Files

The IDE can export a project to an Extensible Markup Language (XML) file. Use this capability to store projects in text-oriented environments, such as a version control system.

1. Bring the project to export forward (in focus).
2. Choose **File > Export Project**.
3. Name the exported XML file and save it in the desired location.

The IDE converts the project to an XML file.

Importing Projects Saved as XML Files

The IDE can import a project previously saved in Extensible Markup Language (XML) format. Use this capability to recreate projects stored in text-oriented environments, such as a version control system.

1. Choose **File > Import Project**.
2. Create a new folder in which to save the converted project and all of its generated files.
3. Find the XML file that you want to import.
4. Save the XML file in the newly created folder.

The IDE converts the XML file to a project.

Closing Projects

Use the **Close** command to close a CodeWarrior project file at the end of a programming session. The IDE automatically saves changes to a closed project.

1. Select the Project window to close.
2. Close the project.

- Choose **File > Close**.
- Click the close box in the Project window.

Advanced Projects

Advanced projects deal with these topics:

- Custom project stationery—modified project stationery tailored to advanced programming needs.
- Subprojects—projects within projects.
- Strategies—obtaining the maximum benefit from advanced projects.

Custom Project Stationery

You can create your own project stationery templates to meet your advanced programming needs:

- Pre-configure new project stationery to include often-used files, libraries, and source code
- Configure build targets and options to any desired state
- Set up a reusable template to use for creating projects

NOTE Custom project stationery requires in-depth knowledge about project structure and operation. Before creating custom stationery, be sure to fully understand existing project stationery included with the CodeWarrior product.

Creating Custom Project Stationery

Use custom project stationery to develop a convenient template for creating new projects. An efficient way to develop custom stationery is to save one of the existing projects in the stationery folder under a different name and then modify the copy.

- Stationery folder for Windows
{CodeWarrior}\Stationery
- Stationery folder for Mac OS
{CodeWarrior}\(Project Stationery)

Working with Projects

Advanced Projects

1. Follow the usual process for creating a project from project stationery.
See [“Creating New Projects using Project Stationery” on page 35](#) for more information.
2. Choose **File > Save A Copy As**.
3. Find the stationery folder in the CodeWarrior installation.
4. Create a folder inside the stationery folder to store the newly created project.
5. **Save** the project to its new folder. Use a descriptive project name with the **.mcp** extension.
6. Customize the newly saved project so that it becomes a template for creating other projects:
 - Add source files to the project. Save these files in the same folder as the project itself.
 - Add build targets for building the project with frequently used settings.
 - Configure other project preferences as desired.
7. Close the customized project to save it.
8. Open the customized project folder inside the stationery folder.
9. Find and delete the **_Data** folder.

The IDE now treats the customized project as project stationery. The descriptive name appears in the **Project** tab of the **New** window.

Subprojects

A subproject is a project nested inside a parent project. Subprojects organize source code for the IDE to build prior to building the parent project. For example, the IDE builds subprojects for an application’s plug-ins before building the parent project for the application itself.

Adding Subprojects to a Project

Use a subproject to organize a separate set of source files and build targets inside a parent project.

1. Open the parent project in which to add a subproject.
2. Click the **Files** tab in the Project window.

3. If the parent project has more than one build target, use the build-target list box in the Project window toolbar to choose the desired build target.
4. Add a separate project to the Project window:
 - Drag and drop the .mcp file of the separate project into the Project window, or
 - Choose **Project > Add Files** to add the .mcp file of the separate project.

The IDE treats the added project as a subproject. The subproject appears in the **Files** view of the parent Project window.

Opening Subprojects

The IDE can open a subproject from the parent Project window. Use this feature to more conveniently open the subproject.

1. Double-click the subproject in the **Files** view of the parent Project window.
2. The IDE opens the subproject in its own Project window.

Strategies

Projects can organize files into build targets or subprojects. Each of these structures has its own advantages. Choose the structure best suited to the programming need.

Build Targets

Build targets organize collections of files inside a project. Build targets have these advantages:

- Using multiple build targets inside a single project allows access to all source code for that project.
- Build targets organize different collections of build settings for a single project.
- Each project accommodates up to 255 build targets.

Subprojects

Subprojects incorporate separate, standalone projects into parent projects. Subprojects have these advantages:

Working with Projects

Advanced Projects

- Subprojects separate distinct parts of a complex program, such as an application and its various plug-ins.
- Using subprojects streamlines a complicated build. For example, create a project that builds all plug-ins for an application. Add this project as a subproject of the main application. The IDE then builds all plug-ins before building the main application.
- Use subprojects to break down a complicated project that approaches the 255 build-target limit. Organize related build targets into different subprojects to improve build speed.

Project Window

This chapter explains how to work with the Project window in the CodeWarrior™ IDE. The Project window provides these features:

- view and modify all files created for use with a computer program.
- manipulate files arranged by type.
- control the way the IDE handles files.

This chapter contains these sections:

- [“About the Project Window” on page 45](#)
- [“Project Window Pages” on page 47](#)
- [“File, Group, Layout, and Target Management” on page 53](#)
- [“Build-Target Management” on page 58](#)

About the Project Window

The Project window organizes files in a computer program. Use this window to control various aspects of each file. The window includes these items:

- Project window toolbar
- Tabs
- Columns

[Figure 4.1 on page 46](#) shows a sample Project window. [Table 4.1 on page 46](#) explains the items in the Project window.

NOTE The number and names of the tabs in the Project window depend on the current build target and on the installed IDE plug-ins.

Project Window

About the Project Window

Figure 4.1 Project window

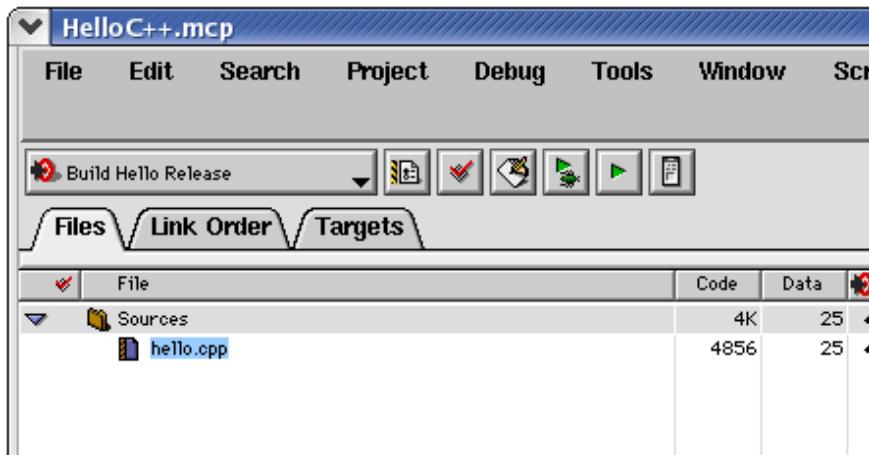


Table 4.1 Project window—items

Item	Icon	Explanation
Current Target		Use to specify the build target that you want to modify.
Target Settings		Click to view and edit the settings for the current build target. You can also display settings for a target selected in Targets tab.
Synchronize Modification Dates		Click to check the modification dates of each project file and mark those files that need compilation.
Make		Click to compile and link all modified and manually selected (touched) project files.
Debug		Click to debug the current build target.

Table 4.1 Project window—items (continued)

Item	Icon	Explanation
Run		Click to compile and link the current build target, then run the program.
Project Inspector		Click to view project information and edit file-specific information.
Files		Click to display the Files page. This page shows a list of files in the project and their associated properties.
Link Order		Click to display the Link Order page. This page shows the link order of files in the current build target.
Frameworks (Mac OS only)		Click to display the Frameworks page. This page shows available programming frameworks to link against. The Frameworks tab appears only for projects that support frameworks.
Targets		Click to display the Targets page. This page shows a list of all build targets, sub-projects, and target-linking information.

Project Window Pages

The Project window uses pages to organize items:

- Files
- Link Order
- Targets
- Frameworks (for projects supporting code frameworks - Mac OS only)

Files Page

The Files page shows information about individual files in a project. The Files page shows information about these file types:

Project Window

Project Window Pages

- Text files—files that contain any type of text. Sample text files include Read Me files and source files.
- Source files—files that contain source code only. Sample source files include C++ files and Java files.
- Library files—files that contain special code designed to work together with a particular programming language or operating environment.

[Table 4.2](#) explains the items in the Files page.

Table 4.2 Files page—items

Item	Icon	Explanation
Touch		Indicates the touch status of each file. Click in this column to toggle touching a file. Touching a file manually selects it for compilation during the next build. Click the Touch icon to sort files by touch status.
File		Displays a hierarchical view of the file and group names used by the project. Click the column title to sort files by name. Double-click a file to open it. Use the hierarchical controls to display and hide group contents.
Code		Displays the size, in bytes or kilobytes, of the compiled executable object code for files and groups. Click the column title to sort files by code size.
Data		Displays the size, in bytes or kilobytes, of non-executable data in the object code for files in the project. Click the column title to sort files by data size.
Target		Indicates whether each file belongs to the current build target. Click in this column to toggle inclusion status. A black dot indicates that a file is included with current build target. Click the Target icon to sort files by inclusion status. The Target column appears only when the project has more than one build target.
Debug		Displays debugging status. Click in this column to toggle generation of debugging information for a file or group. Click the Debug icon to sort files by debugging status.
Checkout Status		Displays icons representing the current file status in a version-control system. The Checkout Status column appears only when the project uses a version-control system to manage files.

Table 4.2 Files page—items (*continued*)

Item	Icon	Explanation
Interfaces		Click to display a list of files inside a group or a list of #include files inside a source file. Choose a file to open it.
Sort Order		Click to toggle sorting between ascending and descending order for the active column. The icon indicates the current sort order.

Viewing a File Path

To distinguish between two files that have identical names but reside in different folders, examine the file path.

To view the complete path of a file, perform the task explained in [Table 4.3](#).

Table 4.3 Viewing a file path

On this host...	Do this...
Windows	Right-click the filename and select Open in Windows Explorer
Macintosh	Control-click the filename and select File Path .
Solaris	Click and hold on the filename, then select File Path .
Linux	Click and hold on the filename, then select File Path .

The File Path submenu shows the path to the file.

File Management

The project window lists all files found for all targets. If access paths are different for each target and a file with the same name exists in each path, the project window will list the occurrence of each file.

Project Window

Project Window Pages

For example, if two header files named `example.h` are used with two targets (TargetA and TargetB) and exist in separate locations for each target, you will see two entries of `example.h` in the project window. If both targets use the same file in one location, then a single entry will appear in the project window.

Select a file in the Files tab of the project window and view the Project Inspector window to reveal the path for the selected file, and which targets use the file. You can also select a file and click the right mouse button to display a context menu. Select Open in Windows Explorer (Windows) or File Path (Linux/Solaris/Mac) to display the path.

If a black dot is present in the target column for a listed file, then it is in the current target. You can select this dot to toggle whether or not to include this file with the current target. Double-click a source file to open it in the editor.

If you enable the **Save project entries using relative paths** option in the Target Settings panel, file locations will be stored using a relative path from the access paths defined in the Access Paths panel. If disabled, the IDE remembers project entries only by name. This can cause unexpected results if two or more files share the same name. In this case, re-searching for files by selecting the Project > Research for Files menu command could cause the IDE to find the file in a different access path.

NOTE If you use source files with the same name in different locations, you should enable the **Save project entries using relative paths** option.

Duplicate file names can also appear in the Files tab of the project window if a file is not found on one of the access paths. This can happen if an access path has been removed from the User Paths group in the Access Paths target settings panel. When the access path is removed, a duplicate appears in the project window. The duplicate entry remains displayed until the access path is restored.

If a project with several targets (for example Debug and Release target) uses the same file, that file is shown as a single entry. If you remove the access path for that file, then a duplicate entry will appear in the file list. This duplicate represents a missing file for the current target. The second file entry is still available for the other target. Restore the access path and choose **Project > Research for Files** to remove the duplicate entry in the list.

The **Project > Re-search for Files** command speeds up builds and other project operations. The IDE caches the location of project files after finding them in the access paths. **Re-search for Files** forces the IDE to forget the cached locations

and re-search for them in the access paths. This command is useful if you moved several files and you want the IDE to find the files in their new locations.

If the **Save project entries using relative paths** option is enabled, the IDE does not reset the relative-path information stored with each project entry, so re-searching for files looks for source files in the same location. If the files are not there, the IDE only re-searches for header files. To force the IDE to also re-search for source files, choose the **Project > Reset Project Entry Paths** menu command. If the **Save project entries using relative paths** option is disabled, the IDE re-searches for both header files and source files.

The **Reset Project Entry Paths** command resets the location information stored with each project entry and forces the IDE to re-search for project entries in the access paths. This command does nothing if the **Save project entries using relative paths** option is disabled.

NOTE If the IDE is unable to locate or resolve the location of project files, a Rescued items folder will appear. The IDE tries to locate the missing files and creates new references. This can happen when project data information, access paths, or other location settings in target settings panels are missing or have been compromised, for example, if the location of a project and related data directory have changed. One way this can happen is if a project has been committed to a source repository by one person and checked out to a different location by another person and a new project data folder is created.

Link Order Page

The Link Order page shows information about the order in which the IDE links project files. Manipulate the files in this page to change the link order. For example, if file B depends on file A in order to function, move file B below file A in the Link Order page.

[Table 4.4 on page 52](#) explains the items in the Link Order page.

Project Window

Project Window Pages

Table 4.4 Link Order page—items

Item	Explanation
Synchronize Modification Dates	To update the modification dates of files stored in a project, click the checkmark icon. Use the Synchronize Modification Dates command to update files modified outside of the CodeWarrior IDE, perhaps by a third-party editor that cannot notify the CodeWarrior IDE of changes.
Synchronize Status	To update version-control status information, click the Pencil icon.

Targets Page

The Targets page presents information about the build targets in a project. Use this page to create, manage, or remove build targets. Different build targets can store different IDE settings. For example, two build targets can handle the same project. One build target handles debugging the software, while the other build target handles building the software for final release.

[Table 4.5](#) explains items in the Targets page.

Table 4.5 Targets page—items

Item	Explanation
Targets	Displays all build targets and subprojects that the IDE processes to create a binary file. These icons denote build-target status: <ul style="list-style-type: none">•  active build target•  inactive build target
Link	Indicates the dependencies between build targets and subprojects.

File, Group, Layout, and Target Management

Use these tasks to manage files, groups, layouts, and targets:

- Create an item.
- Delete an item.
- Move an item.
- Rename an item.
- Touch an item.
- Manage items.
- Set default items.
- Configure item settings.

Removing Files/Groups/Layouts/Targets

The **Remove** command deletes files, groups, layouts, and build targets from the Project window. Removing files from the **Files** tab removes them from the project itself and from all build targets that use the files. Removing a file from the **Link Order**, **Segments**, or **Overlays** tab only removes the file from the current build target.

Removing files/groups/layouts/targets from a project

1. Click the **Files**, **Designs**, or **Targets** tab in the Project window.
2. Select the item to remove.
3. Remove the selected item from the project, as explained in [Table 4.6](#).

Table 4.6 Removing a selected item from a project

On this host...	Do this...
Windows	Select Edit > Delete
Macintosh	Select Edit > Clear

Project Window

File, Group, Layout, and Target Management

Table 4.6 Removing a selected item from a project (continued)

On this host...	Do this...
Solaris	Select Edit > Remove
Linux	Select Edit > Remove

The IDE removes the selected item from the project. For deleted files, the IDE updates all build targets that formerly used the file. For deleted build targets, the IDE deletes build-target information and leaves files intact.

Removing files from a build target

1. Click the **Link Order, Segments, or Overlays** tab in the Project window.
2. Select the item to remove.
3. Remove the selected item from the active build target, as explained in [Table 4.7](#).

Table 4.7 Removing a selected item from the active build target

On this host...	Do this...
Windows	Select Edit > Delete
Macintosh	Select Edit > Clear
Solaris	Select Edit > Remove
Linux	Select Edit > Remove

The IDE removes the file from the build target, but leaves the file itself intact. The file can be re-assigned to other build targets in the project.

Moving Files/Groups/Layouts/Targets

Reposition files, groups, layouts, or build targets in the **Files, Design, Link Order, or Targets** pages with the cursor.

1. Select one or more files, groups, layouts, or build targets to move with the pointer.
2. Drag the selected items to a new position in the current page, using the focus bar as a guide.
3. Release the mouse button.

The IDE repositions the selected files, groups, layouts, or build targets to the new location.

NOTE In the **Link Order** page, repositioning files changes the link order that the **Make** command uses to build the final executable file.

Renaming Files/Groups/Targets

The **Rename** command renames files, groups, or build targets in the project.

Rename files

1. Open the file to rename.
2. Choose **File > Save As**.
3. Type a new filename in the **Name** text box.
4. Click **Save**.

The IDE saves the file under the new name. The new filename appears in the Project window. Subsequent modifications affect the renamed file, leaving the original file intact.

Rename one or more groups

1. Click the **Files** tab in the Project window.
2. Select the group(s) to rename.
3. Press the **Enter** key.
4. Type a new name into the **Enter Group Name** text box of the **Rename Group** window.

Project Window

File, Group, Layout, and Target Management

5. Click **OK**.

The IDE renames the group. For selections of more than one group, the **Rename Group** window appears for each group.

Rename build targets

1. Click the **Targets** tab in the Project window.
2. Choose **Edit > targetname Settings**.
3. Select **Target Settings** in the **Target Settings Panels** list.
4. Type a new name in the **Target Name** text box.
5. Click **Save**.

The Project window displays the new build target name.

Touching Files and Groups

The **Touch** command manually selects source files or groups for compilation during the next **Bring Up To Date**, **Make**, **Run**, or **Debug** operation. A red check mark in the **Touch** column of the Project window indicates a touched file.

1. Click the **Files** tab in the Project window.
2. Touch a source file or group for compilation.

Click the **Touch** column next to the file or group name.

OR

Choose **Touch** from the **Interface** menu for the file or group.

A red check mark appears in the Touch column next to the file or group name.

Touch all project files for recompiling

1. Perform the task explained in [Table 4.8 on page 57](#).

Table 4.8 Touching all project files for recompiling

On this host...	Do this...
Windows	Alt-click the Touch column.
Macintosh	Option-click the Touch column.
Solaris	Alt-click the Touch column.
Linux	Alt-click the Touch column.

2. Red check marks appear next to all files and groups.

Untouching Files and Groups

The **Untouch** command manually excludes source files or groups from compilation during the next **Bring Up To Date**, **Make**, **Run**, or **Debug** operation.

1. Click the **Files** tab in the Project window.
2. Untouch a source file or group to remove it from the compilation list.
Click the red check mark in the **Touch** column next to the file or group name.
OR
Choose **Untouch** from the **Interface** menu for the file or group.

The red check mark disappears from the **Touch** column next to the file or group name.

Untouch all project files

1. Perform the task explained in [Table 4.9](#).

Table 4.9 Untouching all project files

On this host...	Do this...
Windows	Alt-click a red checkmark in the Touch column.
Macintosh	Option-click a red checkmark in the Touch column.

Project Window

Build-Target Management

Table 4.9 Untouching all project files (*continued*)

On this host...	Do this...
Solaris	Alt-click a red checkmark in the Touch column.
Linux	Alt-click a red checkmark in the Touch column.

2. The red checkmarks next to all files and groups disappear.

Build-Target Management

These tasks help you manage build targets:

- Create a build target.
- Remove a build target.
- Set the default build target.
- Rename a build target.
- Configure build-target settings.

Creating Build Targets

The **Create Target** command adds new build targets to a project.

1. Open the **Project** window.
2. Click the **Targets** tab in the Project window.
3. Choose **Project > Create Target**.
4. Type a name in the **Name** text box of the **New Target** window.
5. Select the **Empty target** or **Clone Existing Target** radio button as desired.
 - **Empty Target**—create a new build target from scratch.
 - **Clone Existing Target**—duplicate an existing build target in the **New Target** window.
6. Click **OK**.

The IDE adds the new build target to the project.

Removing Build Targets from a Project

You can remove unneeded build targets from the Project window.

1. Click the **Targets** tab in the Project window.
2. Select the item to remove.
3. Remove the selected build target, as explained in [Table 4.10](#).

Table 4.10 Removing the selected build target

On this host...	Do this...
Windows	Select Edit > Delete
Macintosh	Select Edit > Clear
Solaris	Select Edit > Delete
Linux	Select Edit > Delete

The IDE removes the build target.

Setting the Default Build Target

The CodeWarrior Project Manager can handle up to 255 build targets in a single project. One build target must be defined as the default target when more than one project is open. The default target is the target affected by project commands, such as **Make** and **Run**.

The Project menu

1. Choose **Project > Set Default Target > buildtarget**.
2. A checkmark indicates the default target.

Using the Project window toolbar

1. Enable the **Project** window.
2. Choose the build-target name from the **Current Target** pop-up menu.

The Targets page

1. Enable the **Project** window.
2. Click the **Targets** tab.
3. Click the desired build-target icon.

The icon changes to indicate that the build target is now the default.

Renaming Build Targets

The **Rename** command renames build targets in a project.

1. Click the **Targets** tab in the Project window.
2. Choose **Edit > *targetname* Settings**.
3. Select **Target Settings** in the **Target Settings Panels** list.
4. Type a new name in the **Target Name** text box.
5. Save the new name.

The new build-target name appears in the Project window.

Configuring Build Target Settings

The **Target Settings** panel options determine:

- The compiler used to process the project and produce object code
- The linker used to combine object code and produce a binary file
- The pre-linker and post-linker options that further process the object code
- The name assigned to a build target

Follow these steps to configure build-target settings.

1. Choose **Edit > *targetname* Settings**.
2. Select **Target Settings** from the **Target Setting Panels** list.
3. Specify target options as desired.
4. Save the new options

The panels available in the **Target Settings Panels** list update to reflect the choices in the **Target Settings** panel.

Working with Files

This chapter explains how to work with files in the CodeWarrior™ IDE. Most computer programs use these file types:

- Text files—files that contain any type of text. Example text files include Read Me files and source files.
- Source files—files that contain source code only. Example source files include C++ files and Java files.

Managing Files

These tasks manage files:

- Create a new file.
- Open an existing file.
- Save a file.
- Close a file.
- Print a file.
- Revert a file to a previously saved state.

Creating Text Files (Windows)

The **New** command opens a window from which you create new text files. You can use new text files as source files in a project or as plain-text files.

1. Select **File > New**.

The **New** window appears.

2. Click the **File** tab in the New window.
3. Select **Text File** in the list.
4. Type a filename in the **File name** text box.

Working with Files

Managing Files

5. Click **Set** to specify the location to save the new file.
6. Click **OK**.

The IDE creates the new text file and displays its contents in a new editor window.

TIP Use the **Customize IDE Commands** window to add the **New Text File** menu command to the **File** menu. Adding this menu command reduces the process of creating a new text file to one step: select **File > New Text File**. See [“Customizing the IDE” on page 363](#) for more information about using the Customize IDE Commands window.

Creating Text Files (Macintosh, Solaris, Linux)

The **New Text File** command creates new text files. You can use new text files as source files in a project or as plain-text files.

Select **File > New Text File** to create a new text file. The IDE creates the new text file and displays its contents in a new editor window.

Opening Source Files

The **Open** command opens one or more editable source files. Each open file appears in its own editor window.

NOTE The CodeWarrior editor cannot open files that prohibit editing. For example, the editor cannot open library files.

From the File menu

1. Choose **File > Open**.
2. Windows: Use the **Files of type** pop-up menu to select **All Files**.
3. Select a file.
4. Click **Open**.

The IDE displays the file in an editor window.

From the Project window

1. Perform one of these:
 - Double-click a filename in the **Files** tab of the Project window, or
 - Select an interface filename from the Interface menu.
2. The IDE finds, opens, and displays the selected source file in an editor window.

From an editor window

1. Select an interface filename from the Interface menu.
2. The IDE selects, opens, and displays the source file in an editor window.

NOTE The menu does not show files that do not contain source code or are not yet compiled.

Using Find and Open Files

1. In an editor window, select the name of an interface file, for example `stdio.h`.
2. Choose **File > Find and Open File**.

The IDE finds, opens, and displays the source file in an editor window.

To open a recent file or project

1. Choose **File > Open Recent > *recentfilename | recentprojectname***.
2. The IDE finds and opens the selected source file or project.

Saving Files

Use the **Save** command to save source files to ensure their continued existence between development sessions.

1. Choose **File > Save**.

Working with Files

Managing Files

NOTE If the file has no title, a save dialog appears. Type a filename and specify a location for the file, then click **Save**.

2. The IDE saves the file.
-

Saving All Modified Files

Use the **Save All** command to save the contents of all modified files. This command is useful for saving all files at the same time, rather than saving each file individually.

1. Save all currently opened and modified files, as explained in [Table 5.1](#).

Table 5.1 Saving all currently opened and modified files

On this host...	Do this...
Windows	Select File > Save All
Macintosh	While pressing Option, select File > Save All
Solaris	Select File > Save All
Linux	Select File > Save All

2. The IDE saves the files.
-

Saving File Copies

Use the **Save a Copy As** command to save a back-up copy of a project or file before modifying the original. Working on a copy of the original file provides a way to return to the original copy should modifications fail.

1. Choose **File > Save A Copy As**.
2. Type a new filename in the **Name** text box.
3. Click **Save**.

The IDE creates a copy of the file under the new name, leaving the original file unchanged.

Closing Files

The **Close** command closes open source files. Close editor windows to close a file.

1. Select an editor window to close.
2. Close the file window.
 - Choose **File > Close**, or
 - Click the close box.

NOTE The IDE displays an alert if the file is modified. The alert asks whether to save changes to the file.

The IDE closes the file window.

Closing All Files

The **Close All** command closes all currently open files. This command is useful for closing all files at the same time, rather than closing each file individually.

1. Close all currently open files, as explained in [Table 5.2](#).

Table 5.2 Closing all currently open files

On this host...	Do this...
Windows	Select Window > Close All or Window > Close All Editor Windows .
Macintosh	While pressing Option, select File > Close All .
Solaris	Select File > Close All or File > Close All Editor Windows
Linux	Select File > Close All or File > Close All Editor Windows

2. The IDE closes the files.

Printing Source Files

The **Print** command prints the entire contents of a selected file window.

1. Activate the desired editor window to print.
2. Choose **File > Print**.
3. Set print options in the **Print** dialog.
4. Click **OK** or **Print** to print the file.

The IDE prints the selected file.

NOTE Use the same process to print the contents of a window, such as a Project window.

Printing Source-File Selections

The **Print** command prints the currently selected contents in an editor window.

1. Activate the desired editor window to print.
2. Select the portion of text to print.
3. Choose **File > Print**.
4. Set print options in the **Print** dialog.
5. Click **OK** or **Print**

The IDE prints the selected text in the file.

Reverting Files

Use the **Revert** command to replace the current file with its previously saved version.

1. Choose **File > Revert**.
2. Click **OK** in the **Revert changes to file** dialog.

Dockable Windows

This chapter explains how to work with dockable windows in the Windows-hosted CodeWarrior™ IDE.

NOTE Dockable windows is not available on Linux and Solaris platforms.

Use dockable windows to do these tasks:

- Organize—attach, or *dock*, various windows to the edges of the screen for quick access.
- Group—dock windows of the same type to create a single window with multiple tabs, where each tab represents one of the original docked windows.

NOTE The dockable windows feature is available in Multiple Document Interface (MDI) mode only. This feature is not available in Floating Document Interface (FDI) mode. Toggle the [Use Multiple Document Interface](#) option in the [IDE Extras](#) preference panel to change between these two modes.

This chapter contains these sections:

- [“About Dockable Windows” on page 67](#)
- [“Working with Dockable Windows” on page 70](#)
- [“Dock Bars” on page 76](#)

About Dockable Windows

You can dock certain windows to the edges of the main frame window of the IDE. [Table 6.1 on page 68](#) explains possible states for dockable windows. [Figure 6.1 on page 69](#) shows the different window states.

Dockable Windows

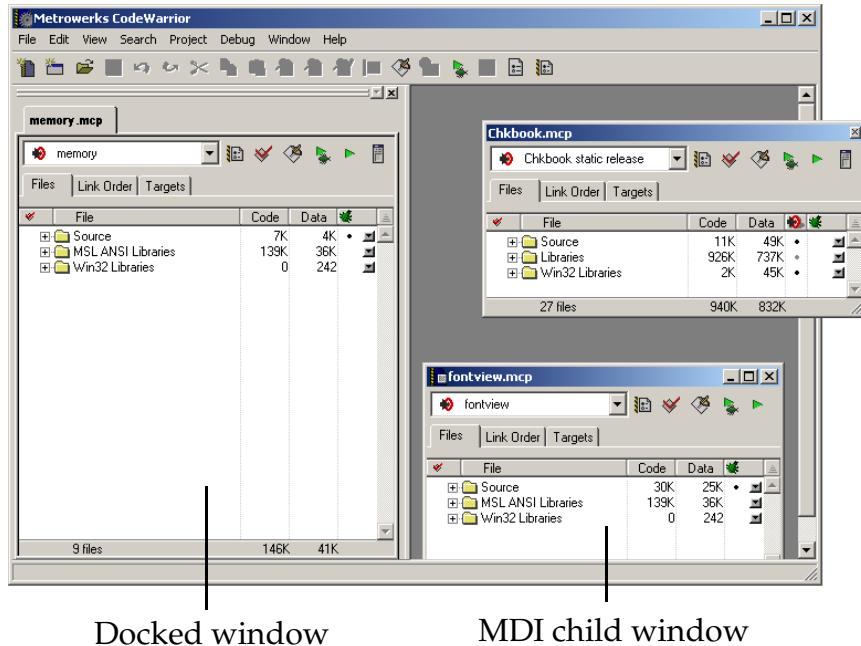
About Dockable Windows

In MDI mode, the IDE occupies a main window frame, or *client area*. IDE windows normally appear within this client area as you work. These windows are called *child windows* of the IDE's client area.

Table 6.1 Window states

State	Characteristics
Docked	<ul style="list-style-type: none">Attached to the left, right, top, or bottom edge of the client arearestricted to the client arearesizablehas a dock bar instead of a title bar
Floating	<ul style="list-style-type: none">Rests above all docked windows and MDI child windowsmovable outside the client area, like a floating palettehas a thin title bardoes not have Minimize or Maximize buttons
MDI Child	<ul style="list-style-type: none">Normal child window of the client area, when running in MDI moderestricted to the client area

Figure 6.1 Window states



Docked window

MDI child window

[Table 6.2](#) explains the difference between dockable windows and non-dockable windows. In this table, the term *non-modal* refers to a window that does not require your attention before allowing the IDE to proceed with other operations.

Table 6.2 Differences between dockable and non-dockable windows

Window Type	Required Criteria	Sample Windows
Dockable	All of these: <ul style="list-style-type: none"> • non-modal • resizable • maximizable 	<ul style="list-style-type: none"> • Thread • Project • Component Catalog
Non-dockable	Any of these: <ul style="list-style-type: none"> • modal • non-resizable • non-maximizable 	<ul style="list-style-type: none"> • IDE Preferences • Find • About Box

Dockable Windows

Working with Dockable Windows

NOTE The default setting for project windows is to dock to an edge of the client area. You can undock these windows.

Compound windows that have more than one pane dock as a group. You cannot separately dock individual panes from these windows. For example, you can dock the Thread Window, but you cannot dock the Stack Crawl pane separately from the Thread Window.

Working with Dockable Windows

You can dock windows in one of two ways:

- dragging a floating window to a docking position
- using a contextual menu to dock a window

You can resize docked windows and undock them to floating windows or MDI child windows.

This section explains how to perform tasks with dockable windows.

Docking a Window By Using a Contextual Menu

Use a contextual menu to dock a floating window or MDI child window to one of the four edges of the client area.

1. Right-click the window title bar.
A contextual menu appears.
2. Choose **Docked** from the contextual menu.

NOTE The **Docked** command appears in the contextual menu for dockable windows only.

The window docks to an edge of the client area. You can resize the docked window or move it to a different edge of the client area.

Docking a Window By Using Drag and Drop

You can drag a docked window or a floating window to one of the four edges of the client area to dock it.

1. Drag the window to one edge of the client area.

Drag a floating window by its title bar. Drag a docked window by its dock bar.

2. A window outline appears near the client-area edge, showing the final position after you release the window.

Use the outline as a visual cue that the IDE will dock the window. If an outline does not appear, you cannot dock the window.

3. Release the window to dock it to the edge.

The window appears in the position indicated by the window outline.

Docking Windows of the Same Kind

You can dock two or more windows of the same kind inside a single docked window. In this arrangement, tabs inside the single docked window represent each of the original docked windows. You can undock each tab individually from the single docked window.

1. Dock the first of two or more windows of the same kind to an edge of the client area.
2. Dock the second window to the same edge as the first window.

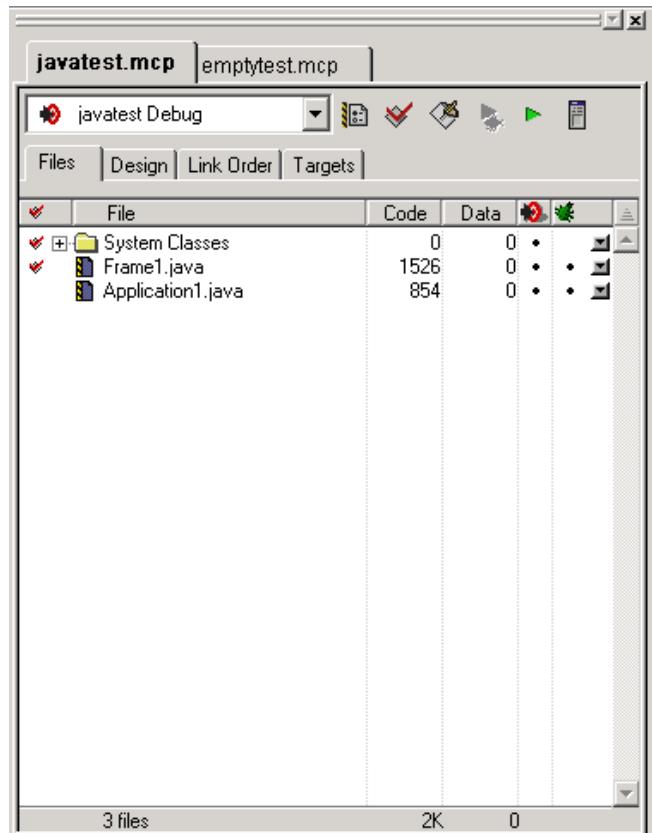
Use the window outline that appears as a visual cue that the IDE will dock the second window to the same edge as the first window.

3. Dock subsequent windows to the same edge as the first window.

Each additional docked window appears as a tab inside the first docked window. Click a tab to view its contents. The frontmost tab appears in bold font.

[Figure 6.2](#) shows two projects represented as tabs in a single docked window.

Figure 6.2 Two projects in a single docked window



Undocking a Window

Use a contextual menu to undock a window from an edge of the client area to a floating window or MDI child window.

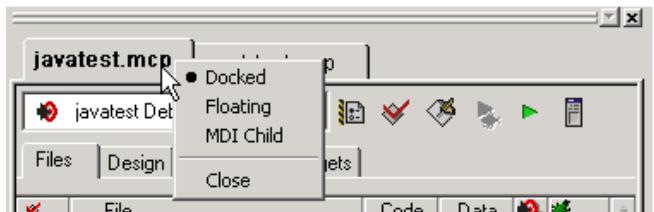
1. Right-click the tab inside the docked window that represents the window you want to undock.

A contextual menu appears.

Dockable Windows

Working with Dockable Windows

Figure 6.3 Contextual menu



2. Choose **Floating** or **MDI Child** from the contextual menu.
 - Floating—undock the window so that it becomes a floating window
 - MDI child—undock the window so that it becomes an MDI child window of the client area

The window undocks and becomes the chosen window type.

Alternately, double-click the tab to undock the corresponding window to a floating window.

Floating a Window

Use a contextual menu to float a docked window or MDI child window.

1. Right-click the tab in the docked window or the title bar of the MDI child window.

A contextual menu appears.
2. Choose **Floating** from the contextual menu.

NOTE The **Floating** command appears in the contextual menu for floatable windows only.

The window becomes a floating window (that you can drag outside the client area).

Alternately, double-click the tab in a docked window to float its corresponding window.

Unfloating a Window

Use a contextual menu to dock a floating window or make it an MDI child window.

1. Right-click the title bar of the floating window.
A contextual menu appears.
2. Choose **Docked** or **MDI Child** from the contextual menu.
 - **Docked**—dock the floating window
 - **MDI child**—unfloat the window so that it becomes an MDI child windowThe window unfloats and becomes the chosen window type.
Alternately, drag the floating window to an edge of the client area to dock it.

Making a Window an MDI Child

Use a contextual menu to make a docked window or floating window an MDI child window.

1. Right-click the tab in the docked window or the title bar of the floating window.
A contextual menu appears.
 2. Choose **MDI Child** from the contextual menu.
- The docked window or floating window becomes an MDI child window.
-

Suppressing Dockable Windows

Suppress dockable windows to drag a window to any location onscreen without docking it to an edge of the client area.

1. Hold down the Ctrl key while dragging or floating an MDI child window.
The thin window outline that normally indicates docked-window placement becomes a heavy window outline. Use this heavy outline as a visual cue that the IDE suppresses dockable windows.

Dockable Windows

Dock Bars

2. Release the window at its final position.

The window appears in the position indicated by the heavy window outline.

3. Release the Ctrl key.

Dock Bars

A docked window has a dock bar instead of a title bar. Use the dock bar to perform these tasks:

- move the docked window to a different edge of the client area
- collapse or expand view of the docked window
- close the docked window

[Figure 6.4](#) shows a dock bar.

Figure 6.4 Dock Bar



Collapsing a Docked Window

If two or more distinct docked windows occupy the same edge of the client area, you can collapse one docked window to view contents of other docked windows.

1. Dock two or more windows to the same edge of the client area.

The windows' contents must appear in separate docked windows, not as tabs in a single docked window.

2. Click the collapse button  on the dock bar of the docked window that you want to collapse.
3. The docked window collapses to hide its contents.

Expanding a Docked Window

If you previously collapsed a docked window, you can expand it and view its contents.

1. Click the expand button on the dock bar: 
 2. The docked window expands to restore its original view.
-

Moving a Docked Window

Use the gripper in a docked window's dock bar to move the docked window to a different edge of the client area.

1. Drag the docked window by the gripper in its dock bar: 
 2. Release the docked window at its new position.
-

Closing a Docked Window

Close a docked window directly from its dock bar.

1. Click the close button on the dock bar: 
 2. The docked window closes.
- Re-opening the window restores its docked position.

Dockable Windows

Dock Bars

Workspaces

This chapter explains how to work with workspaces in the CodeWarrior™ IDE. Use workspaces to do these tasks:

- Organize—save the state of all windows onscreen for later reuse
- Migrate across computers—transfer your workspace from one computer to another

This chapter contains these sections:

- [“About Workspaces” on page 79](#)
- [“Using Workspaces” on page 79](#)

About Workspaces

A workspace stores information about the current state of the IDE. This information consists of the size, location, and the docked state (Windows) of IDE windows. If you save a workspace during an active debugging session, the workspace also stores information about the state of debugging windows.

The IDE can use a default workspace, or it can use a workspace that you create. The IDE works with one workspace at a time. You can save and re-apply a workspace from one IDE session to the next.

Using Workspaces

Use menu commands to perform these workspace tasks:

- save a new workspace
- open an existing workspace
- close the current workspace

Using the Default Workspace

Use the default workspace to preserve IDE state from one session to the next. The IDE saves and restores the default workspace automatically.

1. Choose **Edit > Preferences**.

The IDE Preferences window opens.

2. Select **IDE Extras** in the **IDE Preference Panels** list.

The IDE Extras preference panel appears.

3. Enable the [**Use default workspace**](#) option.

- Checked—the IDE saves its state at the time you quit, then restores that state the next time you launch the IDE
- Unchecked—the IDE always launches with the same default state: no windows visible

Saving a Workspace

Save a workspace to store information about the current state of onscreen windows, recent items, and debugging.

1. Arrange your workspace.

Move windows to your favorite positions and start or finish a debugging session.

2. Choose **File > Save Workspace**.

A **Save** dialog box appears.

3. Enter a name for the current workspace

NOTE Add the extension .cww to the end of the workspace name, for example, `myworkspace.cww`. This extension helps you readily identify the workspace file. The Windows-hosted IDE requires this extension to recognize the file as a CodeWarrior workspace.

4. Save the workspace to a location on your hard disk.

The IDE now uses your saved workspace. In subsequent programming sessions, you can open the workspace.

Opening a Workspace

Open a workspace to apply its settings to the IDE.

1. Choose **File > Open Workspace**.

An **Open** dialog box appears.

2. Open the workspace.

Use this dialog box to navigate your hard disk and select a workspace file.

These files end in the **.cww** extension.

The IDE opens the selected workspace and applies its settings.

Saving a Copy of a Workspace

Save a copy of a current workspace under a different name.

1. Open an existing workspace.
2. Choose **File > Save Workspace As**.

A **Save As** dialog box appears.

3. Enter a name for the copy of the current workspace

NOTE Add the extension **.cww** to the end of the workspace name, for example, **myworkspace.cww**. This extension helps you readily identify the workspace file. The Windows-hosted IDE requires this extension to recognize the file as a CodeWarrior workspace.

4. Save the workspace to a location on your hard disk.

The IDE saves a copy of the current workspace under the name you specified.

Closing a Workspace

Close the current workspace after you finish working with it.

1. Choose **File > Close Workspace**.
2. The IDE closes the current workspace.

Workspaces

Using Workspaces

NOTE You cannot close the default workspace, however, the **IDE Extras** preference panel contains an option that determines whether the IDE uses the default workspace.

You can now open a different workspace or quit the IDE.

Opening a Recent Workspace

You can list recently used workspaces in the **Open Recent** submenu. The **IDE Extras** preference panel contains an option that determines the number of recent workspaces that the submenu will list.

1. Choose **File > Open Recent**.

A submenu appears. This submenu lists recently opened projects, files, and workspaces. A checkmark appears next to the active workspace.

2. Choose a recent workspace from the Open Recent submenu.

The IDE applies the workspace that you select.

Creating Console Applications

This chapter explains how to work with console applications in the CodeWarrior™ IDE. Console applications provide these benefits to novice programmers:

- Simplicity—console applications are computer programs that use a simple text-mode interface. The simplicity of console-mode applications free novice programmers to learn a programming language without having to learn graphical user interface programming at the same time.
- Foundation—understanding console applications provides the basis for more advanced computer programming. Advanced programmers readily understand console applications.

Read this chapter to learn more about typical tasks for working with console applications.

This chapter contains these sections:

- [About Console Applications](#)
- [Creating Console Applications](#)

About Console Applications

A console application is a simple, text-based computer program. Console applications do not usually employ a graphical user interface (GUI). Instead, the applications rely on plain-text input and output in a terminal window.

Console applications are ideal for novice programmers. The applications are easier to program because they lack a GUI. If problems arise, the programmer can use text-based feedback together with the debugger to correct problems.

Creating Console Applications

Creating Console Applications

Creating Console Applications

Create a console application to begin working with a text-based computer program. The CodeWarrior IDE provides pre-configured project stationery for creating console applications. Project stationery simplifies the project-creation process. This section explains how to create a console application.

Creating a Console Application

Use the **New** command to create a new project. The project stores information about the files in the console application.

1. Choose **File > New**.
The **New** window appears.
2. Click the **Project** tab.
3. Select a project stationery file.
4. Enter a project name in the **Project name** field and add the `.mcp` extension.
For example, name the project `test.mcp`.
5. Click **Set**.
Save the project in the desired location.
6. Click **OK**.
The **New Project** window appears.
7. Select a specific stationery file.
8. Click **OK**.
The IDE creates a console application from the selected stationery. The Project window for the console application appears.
9. Expand the **Sources** group.
This group contains placeholder source files.
10. Remove placeholder source files.
For example, select `main.c` and choose **Edit > Remove**.
11. Create a new source file, as explained in [Table 8.1](#).

Table 8.1 Creating a new source file

On this host...	Do this...
Windows	Press Ctrl-N
Macintosh	Press Command-N
Solaris	Press Meta-N
Linux	Press Meta-N (File > New Text File)

12. Enter source code.

For example, enter this source code shown in [Listing 8.1](#).

Listing 8.1 Sample source code

```
#include <stdio.h>
int main( void )
{
    printf("Hello World!");
    return 0;
}
```

13. Save the source file, as explained in [Table 8.2](#).

Table 8.2 Saving the source file

On this host...	Do this...
Windows	Press Ctrl-S (File > Save)
Macintosh	Press Command-S
Solaris	Press Meta-S
Linux	Press Meta-S (File > Save)

Enter a name for the source code. For example, enter Hello.c. Then click **Save**.

14. Choose **Project > Add Hello.c to Project...**

The **Add Files** window appears.

Creating Console Applications

Creating Console Applications

15. Add the file to all build targets in the project.

Select all checkboxes to add the file to all build targets, then click **OK**.

16. Drag the source file inside the **Sources** group.

17. Choose **Project > Run**.

The IDE compiles, links, then runs the console application.



Editor

This section contains these chapters:

- [The CodeWarrior Editor](#)
- [Editing Source Code](#)
- [Navigating Source Code](#)
- [Finding and Replacing Text](#)

The CodeWarrior Editor

This chapter explains how to work with the editor in the CodeWarrior™ IDE. Use the editor to perform these tasks:

- Manage text files—the editor includes common word-processing features for creating and editing text files. Sample text files include Read Me files and release notes.
- Manage source files—the editor includes additional features for creating and editing source files. The IDE processes source files to produce a program.

This chapter contains these sections:

- [“Editor Window” on page 89](#)
- [“Editor Toolbar” on page 92](#)
- [“Other Editor Window Components” on page 94](#)

Editor Window

Use the editor window to create and manage text files or source files. The window contains these major parts:

- Editor toolbar
- Text-editing area
- Line and column indicator
- Pane splitter controls

[Figure 9.1 on page 90](#) shows the editor window. [Table 9.1 on page 90](#) explains the items in the editor window.

The CodeWarrior Editor

Editor Window

Figure 9.1 Editor window

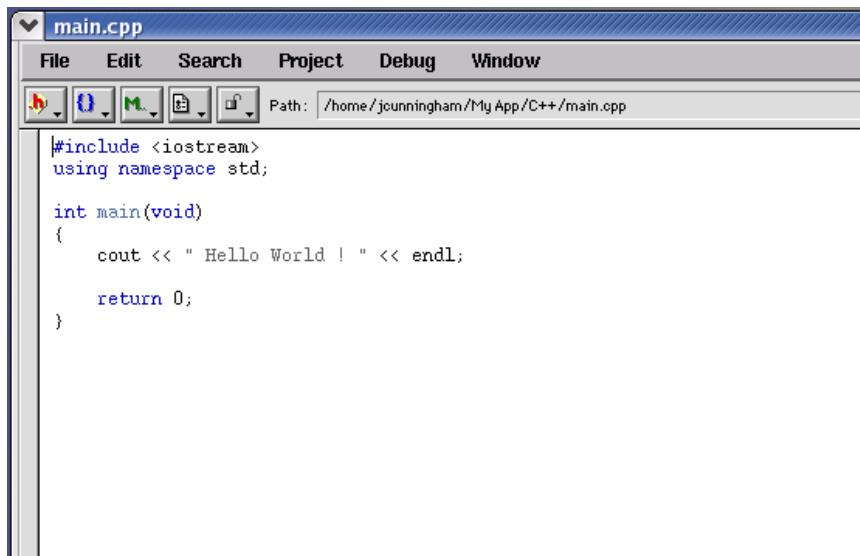


Table 9.1 Editor window—items

Item	Icon	Explanation
Interfaces Menu		Displays a list of referenced interface files or header files for the source file.
Functions Menu		Displays a list of functions defined in the source file.
Markers Menu		Displays a list of markers defined in the file.
Document Settings Menu		Displays file-format options and a syntax-coloring toggle.

Table 9.1 Editor window—items (*continued*)

Item	Icon	Explanation
Version Control System Menu		Displays a list of available Version Control System (VCS) commands. Choose a command to apply to the source file.
Path Caption		Displays the complete path to the file.
File Modification Icon	(top) and (bottom)	This icon indicates an unchanged file since the last save. This icon indicates a file with modifications not yet saved.
Breakpoints Column		Displays breakpoints for the file. Red dot indicates a user-specified breakpoint. Right-click on breakpoints column to bring up context menu.
Text Editing Area		Shows the text or source-code content of the file.
Line and Column Indicator	Line 1 Col 1	Displays the current line and column number of the text-insertion cursor
Pane Splitter Controls		Drag to split the window into panes.

The CodeWarrior Editor

Editor Toolbar

Editor Toolbar

Use the editor toolbar to complete these tasks:

- Open interface and header files
- Find function definitions
- Set and clear markers
- Modify file formats
- Control syntax coloring
- Execute version-control operations
- Determine a file's save state

This section explains how to expand and collapse the toolbar, and how to perform each toolbar task.

Expanding and Collapsing the Editor Window Toolbar

-  To expand the editor window toolbar, click this icon in the right-hand top corner of the editor window.
-  To collapse the Editor Window Toolbar, click this icon in the right-hand top corner of the Editor window.

Interfaces Menu

The Interfaces menu lists the source files included in the current source file.

See “[Finding Interface Files](#)” on page 114 for information on navigating source code with the Interfaces menu.

Functions Menu

The Functions menu lists the functions (routines) defined in the current file.

See “[Locating Functions](#)” on page 114 for information on navigating source code with the Functions pop-up.

Markers Menu

The Marker menu lists markers placed in the current file. Use markers to scroll to specific items in source code and find code segments by intuitive names.

See “[Using Markers](#)” on page 116 for information on navigating source code with Markers.

Document Settings Menu

The Document Settings menu shows whether the IDE applies syntax coloring to the window text, as well as the format in which the IDE saves the file.

Using the Document Settings Menu

Use the **Document Settings** pop-up to toggle syntax coloring on or off for the current file, and set the EOL (end-of-line) format for saving a text file.

The EOL formats are:

- Macintosh: <CR>
- DOS: <CR><LF>
- UNIX: <LF>

To toggle syntax coloring

- Choose **Document Settings > Syntax Coloring**.

The editor window updates to display the new syntax color setting.

To specify the EOL format for the file

- Choose the EOL format for the file.

The IDE applies the specified EOL format to the file the next time it gets saved.

Version Control System Menu

In editor windows, the version control pop-up menu lists options provided by a version control system (VCS) compatible with the IDE. Use a VCS to manage multiple versions of files. VCS packages are available separately for use with the IDE.

Using the Version Control System Menu

Use the **Version Control System** (VCS) pop-up menu to access version control commands related to the editor window's file. If a version control system is not enabled for a project, the only item on the VCS menu is **No Version Control Available**.

- Choose **VCS > VCScommand**

The IDE executes the VCS command.

Other Editor Window Components

Use other editor window components to perform these tasks:

- Determine the path to a file.
- Determine the modification status of a file.
- Set or clear breakpoints.
- Edit text or source code.
- Find the text-insertion point.

This section explains these additional editor window components.

Path Caption

The Path caption shows the path to the active file. The directory delimiters follow host conventions. For example, slashes separate directories for a path on a Windows computer and backslashes are used on Linux and Solaris.

File Modification Icon

The File Modification icon indicates the save status of the file:

- The  icon indicates an unchanged file since the last **Save**.
- The  icon indicates a file with modifications not yet saved.

Breakpoints Column

The Breakpoints column shows breakpoints defined in the current file. Each marker in the column indicates the line of source code at which the debugger suspends program execution.

Text Editing Area

The text editing area behaves the same way as it does in a word processor. Enter text or source code, perform edits, and copy or paste selections.

Line and Column Indicator

The Line and Column indicator shows the current position of the text-insertion point. Click the indicator to specify a line to scroll into view.

Pane Splitter Controls

Use the pane splitter controls to perform these tasks:

- Add panes to editor windows.
- Adjust pane size.
- Remove panes from editor windows.

This section explains how to perform each task.

Adding Panes to an Editor Window

Use the **Pane Splitter** controls to add additional view panes in an editor window and view two or more sections of a source file at the same time.

1. Click and drag a **Pane Splitter control** to add a view pane.
2. The IDE adds a new view pane to the editor window.

Resizing Panes in an Editor Window

Use the **Pane Resize** controls to resize the panes in an editor window.

1. Click and drag a vertical or horizontal **Pane Resize** control.

The CodeWarrior Editor

Other Editor Window Components

2. The IDE resizes the selected view pane.
-

Removing Panes from an Editor Window

Use the **Pane Resize** controls to remove additional view panes from an editor window.

1. Remove an editor window pane.
 - Double-click the **Pane Resize** control to remove the pane, or
 - Click and drag the **Pane Resize** control to the left or top edge of the editor window.
2. The IDE removes the view pane from the editor window.

Editing Source Code

This chapter explains how to edit source code in the CodeWarrior™ IDE. The IDE provides these features to help you edit source code:

- Select and indent text—the editor can select text by line, routine, or rectangular selection. The editor also handles text indentation.
- Balance punctuation—the editor can find matching pairs of parentheses, brackets, and braces. Most programming languages, such as C++, produce syntax errors for punctuation that lacks a counterpart.
- Complete code—the IDE can suggest ways to complete the symbols you enter in a source file

This chapter contains these sections:

- [“Text Manipulation” on page 97](#)
- [“Punctuation Balancing” on page 100](#)
- [“Code Completion” on page 102](#)

Text Manipulation

Use these tasks to manipulate text files:

- Select text
- Overstrike text
- Use virtual space
- Indent text

This section explains how to perform each task.

Selecting Text in Editor Windows

The editor lets you select text in several ways while you edit source files.

NOTE Enable the **Left margin click selects line** option in the **Editor Settings** preference panel to use the right-pointing arrow cursor.

Lines

Follow these steps to select a line of text:

- Triple-click anywhere on a line, or
- Click the right-pointing cursor in the left margin of the line.

Multiple lines

Follow these steps to select multiple lines of text:

- Drag the cursor over several lines of text and release, or
- Position the cursor at the beginning of a selection range, then Shift-click the end of the selection range to select all text between the two points, or
- Drag the right-pointing cursor to select lines of text.

Rectangular text selections

[Table 10.1](#) explains how to select rectangular portions of text.

Table 10.1 Selecting a rectangular portion of text

On this host...	Do this...
Windows	Alt-drag the cursor over the portion of text.
Macintosh	Command-drag the cursor over the portion of text.
Solaris	Alt-drag the cursor over the portion of text.
Linux	Alt-drag the cursor over the portion of text.

Entire routines

Follow these steps to select an entire routine:

1. Hold down the **Shift** key.
2. Choose a function name from the **Function** list menu.

Overstriking Text (Windows OS)

Use the Overstrike command to toggle between text insertion and text overwriting mode when entering text. Press the **Ins** key to toggle overstrike mode.

Using Virtual Space

Use the Virtual Space feature to place the cursor anywhere in the white space of a line of source code and enter text at that position.

For example, consider the line of C++ code shown in [Listing 10.1](#).

Listing 10.1 Sample C++ source code

```
void aFunction (const char * inMessage)           virtualspace
```

Toggling virtual space changes the cursor behavior:

- enabled—clicking in the *virtualspace* places the cursor at the location that you clicked. You can enter text at that location.
- disabled—clicking in the *virtualspace* places the cursor after the last character on the line (in the example, after the closing parenthesis). To place the cursor beyond this character, you must repeatedly press the space bar on your keyboard.

To use virtual space, follow these steps:

1. Select **Edit > Preferences**.

The **IDE Preferences** window opens.

2. Select **Editor Settings** in the IDE Preference Panels list.

The Editor Settings preference panel appears.

3. Select the **Enable Virtual Space** option:

4. Click **Apply** or **Save** to save your changes to the preference panel.

5. Close the IDE Preferences window.

Indenting and Unindenting Text Blocks

Use the **Shift Left** and **Shift Right** commands to shift a selected block of text to the left or right. You can indent or unindent one or more lines using these commands. The **Tab Size** option specifies the amount of indentation.

1. Select the text to be shifted.
2. Indent or unindent the selected text.
 - To unindent text: Choose **Edit > Shift-Left**.
 - To indent text: Choose **Edit > Shift-Right**.

Symbol Editing Shortcuts

You can use the browser contextual menu to enhance source-code editing in the IDE. Use this menu to streamline text entry in editor windows. You can enter the first few letters of a function name, then use the browser contextual menu to complete the entry.

The IDE also provides these keyboard shortcuts with the browser enabled:

- **Find symbols with prefix**—find symbols matching the selected prefix
- **Find symbols with substring**—find symbols matching the selected substring
- **Get next symbol**—obtain the next symbol from the browser database
- **Get previous symbol**—obtain the previous symbol from the browser database

See the *IDE Quick Reference* card for more information about these keyboard shortcuts.

Punctuation Balancing

Balance punctuation to ensure that each opening parenthesis, bracket, or brace has a corresponding closing counterpart. This section explains how to balance punctuation.

Balancing Punctuation

Use the **Balance** option when editing source code to make sure that every parenthesis (()), bracket ([]), and brace ({ }) has a mate.

1. Position the cursor between the suspect punctuation.
2. Check for the matching punctuation.
 - Choose **Edit > Balance**
 - OR
 - Double-click the parenthesis, bracket, or brace character to check for a matching character.

From a text insertion point, the editor searches forward until it finds a parenthesis, bracket, or brace, then it searches in the opposite direction until it finds the matching punctuation. When double-clicking on a parenthesis, bracket, or brace, the editor searches in the opposite direction until it finds the matching punctuation.

When it finds a match, it highlights the text between the matching characters. If the insertion point is not enclosed or if the punctuation is unbalanced, the computer beeps.

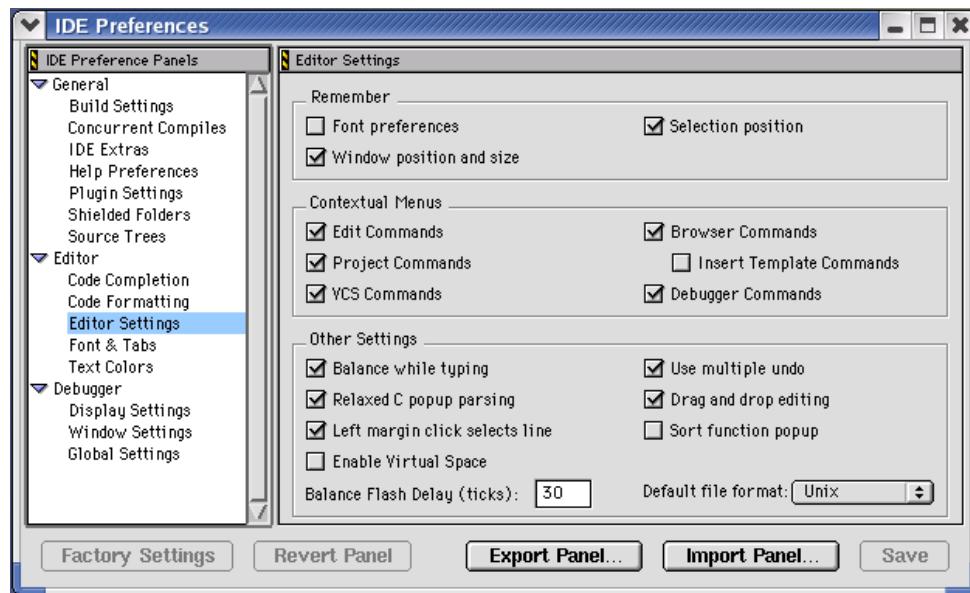
Toggling Automatic Punctuation Balancing

Use the **Editor Settings** to enable or disable the punctuation balancing feature.

Editing Source Code

Code Completion

Figure 10.1 Editor Settings (Balance While Typing)



To toggle automatic punctuation balancing, follow these steps:

1. Select **Edit > Preferences**.
This opens the **IDE Preferences** window.
2. In the **IDE Preference Panels** list, select **Editor Settings**.
3. In the **Other Settings** area of Editor Settings, select or clear the **Balance While Typing** checkbox.

Code Completion

Use code completion to have the IDE automatically suggest ways to complete the symbols you enter in a source file. By using code completion, you avoid referring to other files to remember available symbols.

C/C++ Code Completion will function more effectively when “Language Parser” is selected for the “Generate Browser Data From” option in the Build Extras target settings panel for a project. Java Code Completion is not affected by this setting.

Code Completion Configuration

You can activate, deactivate, and customize code-completion operation. These tasks are associated with code completion:

- Activate automatic code completion
- Trigger code completion from the IDE menu bar
- Trigger code completion from the keyboard
- Deactivate automatic code completion

Activating Automatic Code Completion

Activate automatic code completion to have the IDE display a Code Completion window that helps you complete the symbols you enter in source code. The **Code Completion** preference panel configures the Code Completion window behavior.

1. Choose **Edit > Preferences**.

The **IDE Preferences** window appears.

2. Select the **Code Completion** preference panel in the **IDE Preference Panels** list.

3. Select the [**Automatic Invocation**](#) option.

Selecting this option configures the IDE to automatically open the Code Completion window.

4. Enter a delay in the [**Code Completion Delay**](#) field.

This delay determines how long the IDE waits between the time you type a trigger character and the time the Code Completion window appears. If you perform any action during this delay time, the IDE cancels the Code Completion operation.

5. Save your preferences.

Click the **Save** or **Apply** button.

The Code Completion window now appears automatically to help you complete code in editor windows.

Editing Source Code

Code Completion

Triggering Code Completion from the IDE Menu

Trigger code completion from the main menu to open the Code Completion window.

1. Bring forward an editor window.
2. Begin typing or place insertion point at end of source code that you want to complete.
3. Choose **Edit > Complete Code**

The Code Completion window appears. Use it to complete the symbol at the insertion point.

Triggering Code Completion from the Keyboard

To open code completion from the keyboard:

1. Bring forward an editor window.
2. Begin typing or place insertion point at end of source code to complete.
3. Press the appropriate code completion shortcut key combination.

[Table 10.2 on page 104](#) lists the default code completion key bindings for each IDE host. Use the **Customize IDE Commands** panel to change these key bindings.

Table 10.2 Code Completion key bindings

Host	Get Next Completion	Get Previous Completion	Complete Code
Windows	Alt-/	Alt-Shift-/	Alt-.
Macintosh	Control-/	Control-Shift-/	Control-.
Linux/Solaris	Control-/	Control-Shift-/	Control-.

Deactivating Automatic Code Completion

Deactivate automatic code completion to prevent the IDE from displaying the Code Completion window as you edit source code. The **Code Completion** preference panel configures Code Completion window behavior.

You can still manually trigger code-completion functionality from the keyboard or from the main menu.

NOTE To dismiss the Code Completion window after it automatically opens, press the **Esc** key or click outside the active editor window.

1. Choose **Edit > Preferences**.
2. Select the **Code Completion** preference panel in the **IDE Preference Panels** list.
3. Disable the [Automatic Invocation](#) option.
Clearing this option prevents the IDE from automatically opening the Code Completion window.
4. Save your preferences.
Click the **Save** or **Apply** button.

Code Completion Window

The Code Completion window displays possible symbols based on the context of the insertion point. For example, in Java you can complete code for any Java class, method, and variable from any package that has been imported or is being used elsewhere in the project.

[Figure 10.2](#) shows the Code Completion window. [Table 10.3 on page 106](#) explains the items in the Code Completion window. [Table 10.4 on page 107](#) explains the icons that appear in the Code Completion list.

Editing Source Code

Code Completion

Figure 10.2 Code Completion window

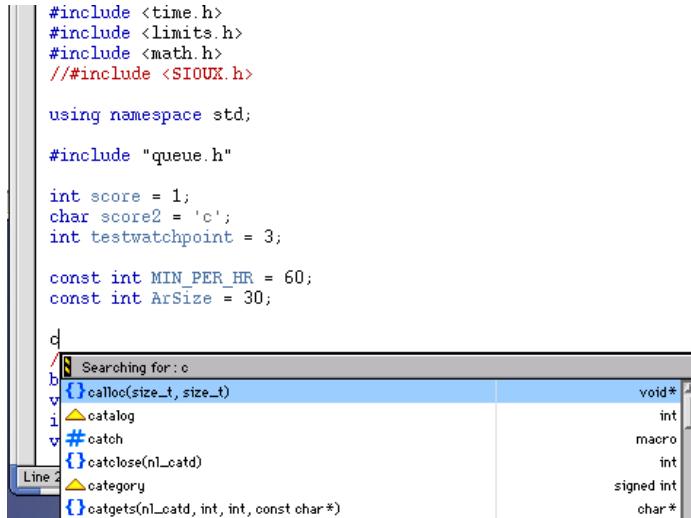


Table 10.3 Code Completion window—items

Item	Icon	Explanation
Code Completion list		Lists available variables and methods or functions along with their corresponding return types or parameters. This list changes based on the context of the insertion point in the active editor window. Icons help distinguish items in the list.
Disclosure Triangle	▼ ▶	Click to toggle display of Documentation pane for programming languages that support it.
Resize Bar	[horizontal bar]	Drag to resize the Code Completion list and the Documentation pane.
Documentation pane		Displays summary information or documentation for the selected item in the Code Completion list. This pane appears only for programming languages that support summary information or documentation.

Table 10.4 Code Completion window—icons

Icon	Code Type	Icon	Code Type
	Class		Method
	Function		Namespace
	Global Variable		None
	Language Keyword		Package
	Local Variable		Variable
	Constant		

Navigating the Code Completion Window

Navigate the Code Completion window by mouse or keyboard. You can perform these tasks:

- Resize the window
- Navigate the window by keyboard
- Refine the Code Completion list by keyboard

1. Bring forward an editor window.
2. Place the insertion point at the end of the source code to complete.
3. Choose **Edit > Complete Code** or use keyboard shortcut.
The Code Completion window appears.
4. Use the mouse to resize the Code Completion window (Mac and Windows).

The new window size remains in effect until you refine the Code Completion list or close the Code Completion window. You refine the Code Completion list by typing additional characters in the active editor window.

Editing Source Code

Code Completion

5. Use the keyboard to navigate the Code Completion list.

[Table 10.5](#) explains how to navigate the Code Completion list by keyboard.

Table 10.5 Navigating the Code Completion list by keyboard

Key	Action
Up Arrow	Select the previous item
Down Arrow	Select the next item
Page Up	Scroll to the previous page
Page Down	Scroll to the next page

6. Use the keyboard to refine the Code Completion list.

The Code Completion list updates as you add or delete characters in the active editor window. Continue adding characters to narrow the list, or delete existing characters to broaden the list. Press the Backspace key to delete characters.

Selecting an Item in the Code Completion Window

Select an item in the Code Completion window to have the IDE enter that item in the active editor window at the insertion point.

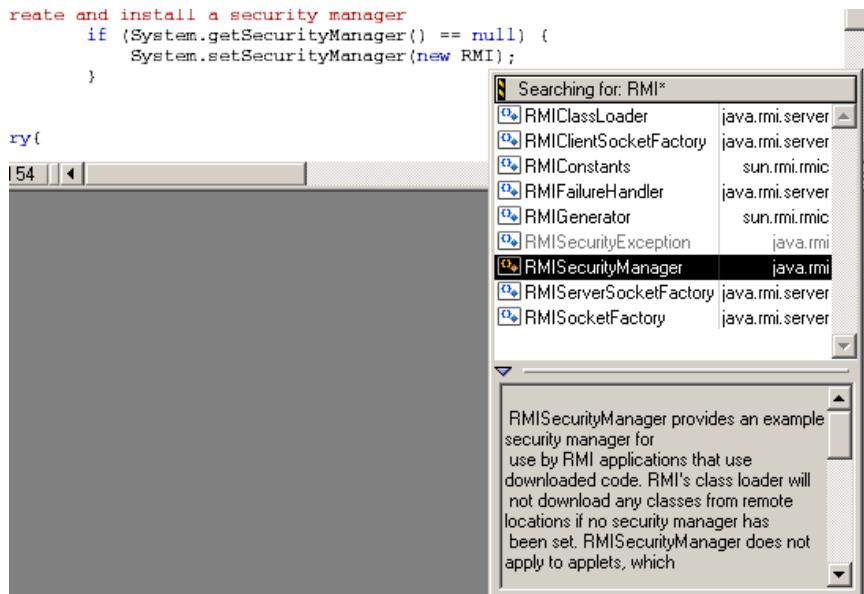
1. Bring forward an editor window.
2. Place the insertion point at the end of the source code to complete.
3. Choose **Edit > Complete Code**.
4. Select an item in the Code Completion list.
5. Enter the item into the active editor window.

Press the **Return** or **Enter** keys on the keyboard or double-click the item to have the IDE insert that item into the editor window.

Completing Code for Data Members and Data Types

Complete code for data members for programming languages that support it. For a list of data members type the period (.) character and activate the code completion window. [Figure 10.3](#) shows an example of helping you select the correct data type depending on what code has been typed in the source file.

Figure 10.3 Code Completion List of Data Types



Completing Code for Parameter Lists

Complete code for parameter lists for programming languages that support it. For example, you can complete code for parameter lists by typing the open parenthesis (character.

1. Bring forward an editor window.
2. Place the insertion point at the end of the function or method to complete.
3. Type an open parenthesis to trigger a parameter-list.

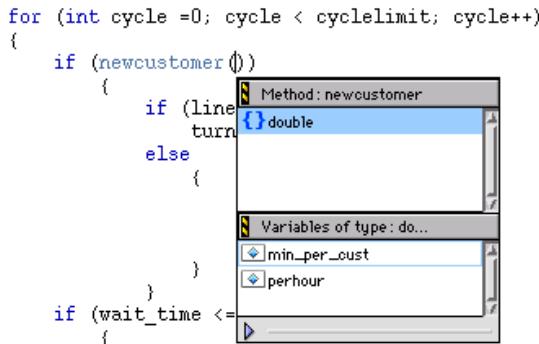
Editing Source Code

Code Completion

4. The Code Completion window appears.

The upper portion of this window lists different (overloaded) versions of the function or method. The lower portion shows possible parameter lists for the selected function or method in the top portion. Use this window to complete the parameter list for the function or method.

Figure 10.4 Code Completion for Parameter Lists



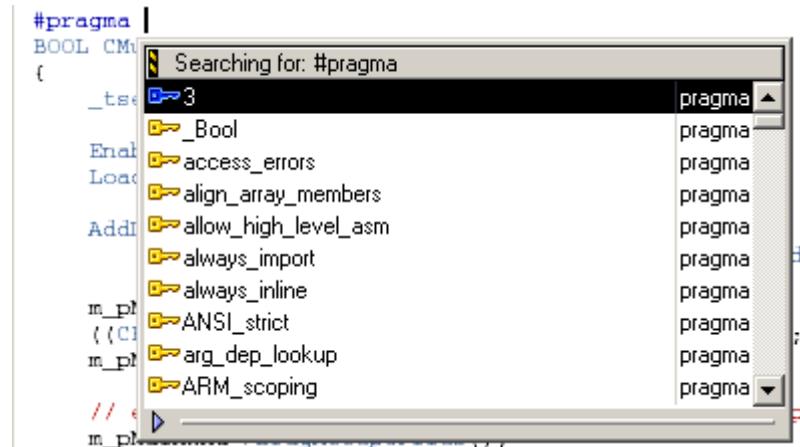
Completing Code for Pragmas

In the Mac or Windows hosted IDE you can display a list of pragmas in the code completion window.

1. Bring forward an editor window.
2. In your source file, type `#pragma` followed by a space.
3. Activate the code completion window (cntrl . or Alt .).

The code completion window will display list of pragmas.

Figure 10.5 Code Completion for Pragmas



Editing Source Code

Code Completion

Navigating Source Code

This chapter explains how to navigate source code in the CodeWarrior™ IDE. Navigate source code to accomplish these tasks:

- Find specific items—the editor finds interface files, functions, and lines of source code.
- Go to a specific line—the editor can scroll to a specific line of source code.
- Use markers—the editor allows labelling of specific items of text. These labels, or markers, provide intuitive navigation of text.

Read this chapter to learn more about typical tasks for navigating source code.

This chapter contains these sections:

- [“Finding Interface Files, Functions, and Lines” on page 113](#)
- [“Going Back and Forward” on page 116](#)
- [“Using Markers” on page 116](#)
- [“Symbol Definitions” on page 119](#)
- [“Reference Templates \(Macintosh\)” on page 120](#)

Finding Interface Files, Functions, and Lines

Find interface files, functions, and lines of source code to expedite programming. You can find these types of items:

- interface files
- functions
- lines of source code

Navigating Source Code

Finding Interface Files, Functions, and Lines

Finding Interface Files

Find interface (header) files referenced by the current source code. Some programming languages, such as C++, use interface files in conjunction with source code. Interface files typically define functions or objects used in the source code. Interface files also separate function or object declarations from implementations. This section explains how to find interface files.

Using the Interface Menu



Use the Interface menu in editor windows to open interface or header files referenced by the current file. The project file must be open for the Interface menu to operate.

1. Click the Interface menu.
2. Select the filename of the interface file that you want to open.

If found, the file is opened in an editor window. If not found, an alert sounds.

NOTE Only source code interface files can be opened. Libraries and pre-compiled header files can not be opened.

Locating Functions

Find functions to expedite source-code editing. Most source files contain several functions that divide a complicated task into a series of simpler tasks. The editor allows scrolling to individual functions within the current source file. This section explains how to find functions.

Using the Functions Menu



Use the Functions menu in editor windows to quickly navigate to specific functions or routines in the current source file.

1. Click the Functions menu.
2. Select the function name to view.

The editor scrolls to display the selected function.

Alphabetizing Functions Menu with the Mouse and Keyboard

The default behavior of the Functions menu is to list functions in order of appearance in the source file. You can use the mouse and keyboard to list functions in alphabetical order.

[Table 11.1 Alphabetizing the Functions list](#) explains how to use the mouse and keyboard to alphabetize functions in the Functions menu.

Table 11.1 Alphabetizing the Functions list

On this host...	Do this...
Windows	Ctrl-click the Functions menu.
Macintosh	Option-click the Functions menu.
Solaris	Alt-click the Functions menu.
Linux	Alt-click the Functions menu.

Alphabetizing Functions Menu Order

The default behavior of the Functions menu is to list functions in order of appearance in the source file. You can select the [Sort function popup](#) option in the **Editor Settings** preference panel to list functions in alphabetical order.

1. Open the **IDE Preferences** window.
Choose **Edit > Preferences**.
2. Select the **Editor Settings** preference panel.
3. Select the [Sort function popup](#) option.
4. Save your modifications to the **Editor Settings** panel.

Going Back and Forward

Go back and forward in source files to edit existing code. Most source files contain more than one screen of code. The editor always counts the number of lines in the source files. Go to a particular line to scroll a particular item into view.

Going to a Line

Use the **Goto Line** command to navigate to a specific source line in an editor window if you know its number. Lines are numbered consecutively, with the first line designated as line 1. The **Line Number** control at the bottom of the editor window shows the line number where the text insertion point is positioned.

1. Open the Line Number window.
 - Click the **Line and Column Indicator** control in bottom left corner of editor window, or
 - Choose **Search > Go To Line**
2. Type a line number in the **Line Number** text box.
3. Click **OK**.

NOTE If a line number does not exist, the insertion point jumps to the last line of the source file.

Using Markers

Markers behave like labels in the editor, identifying specific parts of source code. Use these tasks to work with markers:

- Add markers to a source file
- Navigate to a marker
- Remove some or all markers from a source file

Remove Markers Window

Use the **Remove Markers** window to manage the use of destination markers in source files. [Figure 11.1](#) shows the Remove Markers window. [Table 11.2](#) explains the items in the window.

Figure 11.1 Remove Marker window

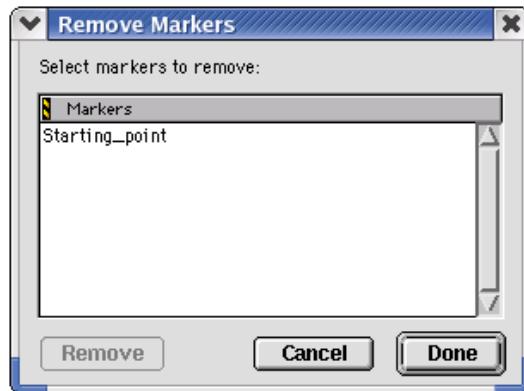


Table 11.2 Remove Markers window—items

Item	Explanation
Markers list	Displays a list of all markers in the current source file.
Remove button	Click to remove all selected markers.
Cancel button	Click to close the Remove Markers window without applying changes.
Done button	Click to close the Remove Markers window and apply changes.

Adding Markers to a Source File

Use the **Add Marker** command to add a marker to a file to identify specific line locations by name.

1. Position the cursor on a line.
2. Click on Marker icon and select Add Marker.

Navigating Source Code

Using Markers

3. Type a name for the new marker.
4. Click **Add**.

The IDE adds the marker to the file.

Navigating to a Marker

Once you add a marker, you can use the Marker menu to return to it later.

1. Select the marker name from the Marker menu.
 2. The editor window scrolls to display the selected marker.
-

Removing a Marker from a Source File

Use the **Remove Marker** command to remove one or more markers from a source file.

1. Click Marker icon and select Remove Markers
2. Select the marker name to remove from the list.
3. Click **Remove**.

The IDE removes the selected marker.

Removing All Markers from a Source File

Use the **Remove Marker** command to remove one or more markers from a source file.

1. Click Marker icon and select Remove Markers
2. Select all markers in the **Markers** list, as explained in [Table 11.3](#).

Table 11.3 Selecting all markers in the Markers list

On this host...	Do this...
Windows	Shift-click each marker name in the list.
Macintosh	Select Edit > Select All .

Table 11.3 Selecting all markers in the Markers list (continued)

On this host...	Do this...
Solaris	Select Edit > Select All.
Linux	Select Edit > Select All.

3. Click **Remove**.

The IDE removes all markers.

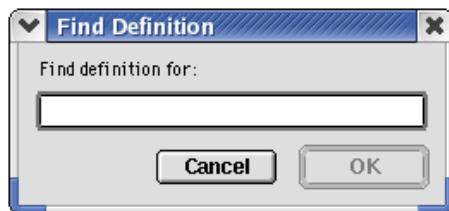
Symbol Definitions

You can find a symbol definition in your project's source code. For the Mac OS, you can also look up a symbol definition using the online documentation viewer in the **IDE Extras** selection in the **IDE Preferences** panel.

Supported online reference viewers include HTMLHelp (Windows) and QuickHelp (Mac OS), as well as older online help systems such as QuickView (Mac OS) and THINK Reference (Mac OS).

TIP You can also use the browser to look up symbol definitions.

Figure 11.2 Find Definition



Looking Up Symbol Definitions

To look up the definition of a selected symbol, follow these steps:

1. Choose **Search > Find Definition**
2. Enter the symbol definition.

Navigating Source Code

Reference Templates (Macintosh)

3. Click **OK**.

CodeWarrior searches all 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.

TIP To return to your original location after viewing a symbol definition, press Shift-Ctrl B (Windows) or Shift-Command B (Mac OS) or Meta-Shift B (Linux/Solaris). This key binding is equivalent to the **Go Back** menu command.

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 command, CodeWarrior searches the online documentation for the symbol definition. After you select a symbol and choose the Find Definition & Reference command, the IDE searches both the project files and the online documentation for the symbol definition. If CodeWarrior does not find a definition or reference, it notifies you with a beep.

Reference Templates (Macintosh)

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 editor window at the text-insertion point. If you know the name of the call that you want to add, but are not familiar with the call parameters, this technique is useful.

[Listing 11.1](#) shows a sample routine template.

Listing 11.1 Sample routine template

```
SetRect (r, left, top, right, bottom);
```

Inserting a Reference Template

To insert a reference template into your code, follow these steps:

1. From the online viewer window, type the routine name that you want to insert.

2. Select the name you just typed.
3. Choose **Insert Reference Template** from the **Edit** menu.

The 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 the IDE finds the routine, the IDE copies the template to the active editor window and replaces the text you selected with the template.

Finding and Replacing Text

This chapter explains how to work with the find-and-replace features in the CodeWarrior™ IDE.

This chapter contains these sections:

- [“Single-File Find” on page 123](#)
- [“Single-File Find and Replace” on page 126](#)
- [“Multiple-File Find and Replace” on page 129](#)
- [“Search Results Window” on page 141](#)
- [“Text-Selection Find” on page 143](#)
- [“Regular-Expression Find” on page 145](#)
- [“Comparing Files and Folders” on page 148](#)

Single-File Find

Use the **Find** window to search for text within a single file:

- The **Find** operation returns a single instance of matching text.
- The **Find All** operation returns all instances of matching text.

[Figure 12.1](#) shows the Find window. [Table 12.1 on page 124](#) explains the items in the Find window.

Finding and Replacing Text

Single-File Find

Figure 12.1 Find window

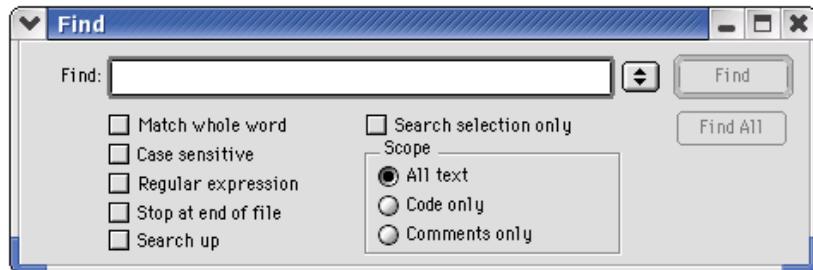


Table 12.1 Find window—items

Item	Explanation
Find text/list box	Enter a search string. Click the arrow symbol to select a search string that you entered previously.
Find button	Click to start a search operation using the string in the Find text/list box.
Find All button	Click to search for all matches in the active editor window.
Cancel button	Click to close the Find window without performing a search.
Match whole word checkbox	<p>Check to search for whole-word matches only, ignoring matches within words.</p> <p>Clear to search for all matches of the search string, including matches within words.</p>
Case sensitive checkbox	<p>Check to consider text case during the search. The search operation distinguishes between a capital letter and the same letter in lower case.</p> <p>Clear to disregard text case during the search. The search operation does not distinguish between a capital letter and the same letter in lower case.</p>
Regular expression checkbox	<p>Check to treat the search string as a regular expression.</p> <p>Clear to treat the search string as plain text.</p>

Table 12.1 Find window—items (*continued*)

Item	Explanation
Stop at end of file checkbox	Check to stop a search at the end of a file and not wrap around to the beginning of the file. Clear to wrap around to the beginning of the file and continue a search. The search stops at the first match or at the current cursor position.
Search up checkbox	Check to perform a search operation back from the current selection. Clear to perform a search operation forward of the current selection
Search selection only checkbox	Check to search only the currently selected text and not the entire file. Clear to search the entire file.
All text option button	Select to search all text in the file.
Code only option button	Select to search only source code in the file.
Comments only option button	Select to search only comments in the file.

Searching Text in a Single File

Use the **Find** command to search for text in the active editor window.

1. Click **Search > Find**.

The Find window appears.

NOTE (Mac OS, Solaris, and Linux) Use the **Customize IDE Commands** window to activate the **Find** menu command.

2. Enter search text into **Find** text/list box.
3. Set search options.

Finding and Replacing Text

Single-File Find and Replace

4. Click the **Find** or **Find All** button to start the search.

The IDE searches the current file until it finds a match or reaches the end of the search. A single match appears highlighted in the editor window, or multiple matches appear in a Search Results window. The IDE beeps if it does not find any matching text.

TIP If you clicked the Find button to start the search, click **Search > Find Next** to find the next match in the file.

Single-File Find and Replace

Use the **Find and Replace** window to perform these tasks:

- Search a single file.
- Replace found text in a single file.

[Figure 12.2](#) shows the Find and Replace window. [Table 12.2](#) explains the items in the Find and Replace window.

Figure 12.2 Find and Replace window

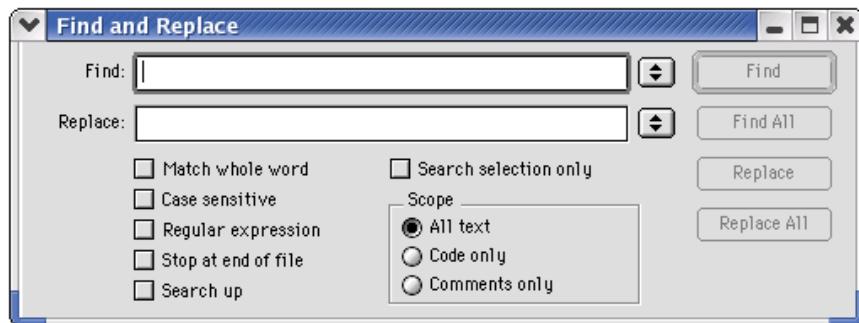


Table 12.2 Find and Replace window—items

Item	Explanation
Find text/list box	Enter a search string. Click the arrow symbol to select a search string that you entered previously.
Replace with text/list box	Enter the replacement string. Click the arrow symbol to select a replacement string that you entered previously.
Find button	Click to start a search operation using the string in the Find text/list box.
Replace button	Click to replace the current match with the replacement string.
Replace All button	Click to replace all matches with the replacement string.
Cancel button	Click to close the Find and Replace window without performing a search.
Match whole word checkbox	<p>Check to search for whole-word matches only, ignoring matches within words.</p> <p>Clear to search for all matches of the search string, including matches within words.</p>
Case sensitive checkbox	<p>Check to consider text case during the search. The search operation distinguishes between a capital letter and the same letter in lower case.</p> <p>Clear to disregard text case during the search. The search operation does not distinguish between a capital letter and the same letter in lower case.</p>
Regular expression checkbox	<p>Check to treat the search string as a regular expression.</p> <p>Clear to treat the search string as plain text.</p>
Stop at end of file checkbox	<p>Check to stop a search at the end of a file and not wrap around to the beginning of the file.</p> <p>Clear to wrap around to the beginning of the file and continue a search. The search stops at the first match or at the current cursor position.</p>

Finding and Replacing Text

Single-File Find and Replace

Table 12.2 Find and Replace window—items (*continued*)

Item	Explanation
Search up checkbox	Check to perform a search operation back from the current selection. Clear to perform a search operation forward of the current selection
Search selection only checkbox	Check to search only the currently selected text and not the entire file. Clear to search the entire file.
All text option button	Select to search all text in the file.
Code only option button	Select to search only source code in the file.
Comments only option button	Select to search only comments in the file.

Replacing Text in a Single File

Use the **Replace** command to replace matching text.

1. Click **Search > Replace** or **Search > Find and Replace**.

The Find window appears.

2. Enter search text into the **Find** text/list box.
3. Enter replacement text into the **Replace with** text/list box.
4. Set search options.
5. Find and replace text:

- a. Click the **Find** button to search for matching text.

The IDE searches the current file until it finds a match or reaches the end of the search. A single match appears highlighted in the editor window.

The IDE beeps if it does not find any matching text.

- b. Click the **Replace** or **Replace All** button to replace the matching text.

Click the Replace button to replace the current match. Click the Replace button repeatedly to replace subsequent matches. Click the Replace All button to replace all matching text in the file.

To replace consecutive matches, click the Find button to find the first match, then repeatedly click the Replace button. To replace one match at a time, or to replace non-consecutive matches, click the Find button to find a match, then click the Replace button as needed.

TIP If you clicked the Find button to start the search, click **Search > Find Next** to find the next match in the file.

Multiple-File Find and Replace

Use the **Find in Files** window to perform these tasks:

- Search several files.
- Replace found text in multiple files, folders, symbolics files, or projects.
- Replace found text in files within a specific build target.

[Figure 12.3](#) shows the Find in Files window. [Table 12.3](#) explains the items in the window.

Finding and Replacing Text

Multiple-File Find and Replace

Figure 12.3 Find in Files window

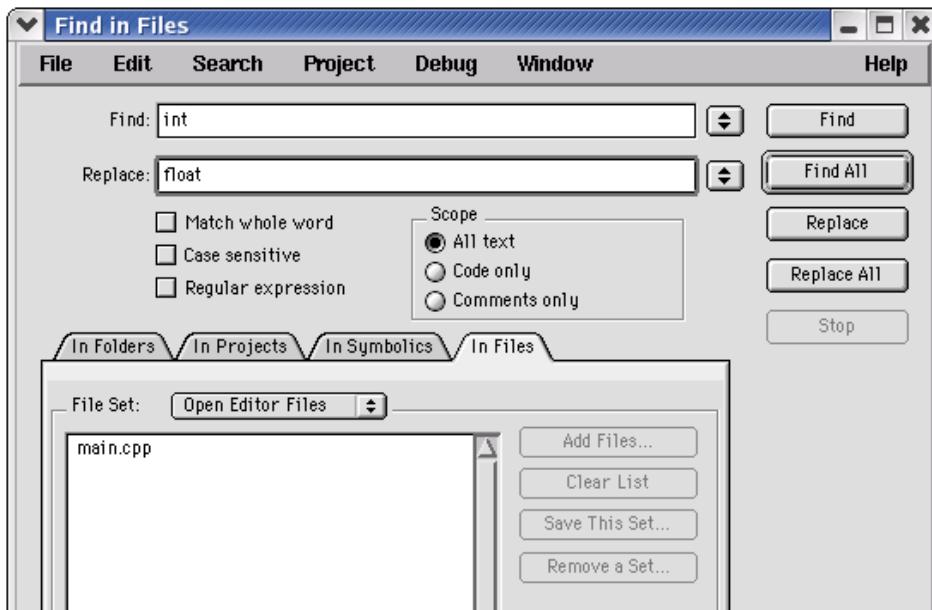


Table 12.3 Find in Files window—items

Item	Explanation
Find text/list box	Enter a search string. Click the arrow symbol to select a search string that you entered previously.
Replace with text/list box	Enter the replacement string. Click the arrow symbol to select a replacement string that you entered previously.
Find button	Click to start a search operation using the string in the Find text/list box.
Find All button	Click to search for all matches in the selected items.
Replace button	Click to replace the current match with the replacement string.
Replace All button	Click to replace all matches with the replacement string.
Stop button	Click to stop the current operation.

Table 12.3 Find in Files window—items (*continued*)

Item	Explanation
Match whole word checkbox	Check to search for whole-word matches only, ignoring matches within words. Clear to search for all matches of the search string, including matches within words.
Case sensitive checkbox	Check to consider text case during the search. The search operation distinguishes between a capital letter and the same letter in lower case. Clear to disregard text case during the search. The search operation does not distinguish between a capital letter and the same letter in lower case.
Regular expression checkbox	Check to treat the search string as a regular expression. Clear to treat the search string as plain text.
All text option button	Select to search all text in the selected items.
Code only option button	Select to search only source code in selected items.
Comments only option button	Select to search only comments in selected items.
In Folders tab	Click to bring forward the In Folders page. Use this page to search specific folders in the host file system.
In Projects tab	Click to bring forward the In Projects page. Use this page to search active projects and build targets.
In Symbolics tab	Click to bring forward the In Symbolics page. Use this page to search files containing symbolics (debugging and browsing) information generated by the IDE.
In Files tab	Click to bring forward the In Files page. Use this page to search files contained in custom file sets.

In Folders

Use the **In Folders** page to search folder contents for matching text. [Figure 12.4](#) shows the In Folders page. [Table 12.4](#) explains the items in the page.

Finding and Replacing Text

Multiple-File Find and Replace

Figure 12.4 Find in Files window—In Folders page

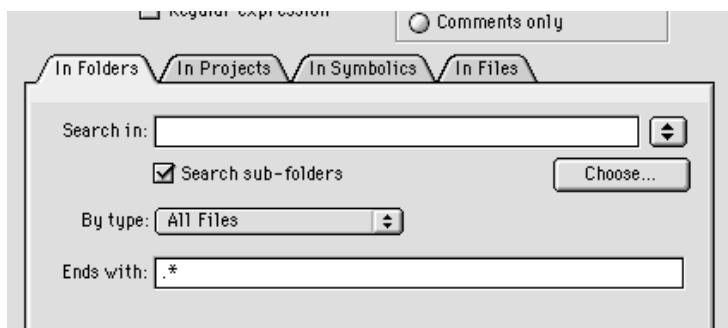


Table 12.4 Find in Files window—In Folders items

Item	Explanation
Search in text/list box	Enter the path to the folder that you want to search. Click the arrow symbol to select a path that you entered previously.
Browse button	Click to open a dialog box that lets you pick the folder that you want to search.
Search sub-folders checkbox	Check to search sub-folders of the selected folder. Clear to search the selected folder only, ignoring any sub-folders it may contain.
By type text/list box	Enter the filename extensions of the files that you want to search. Click the arrow symbol to select a set of filename extensions. The search ignores files whose filename extensions do not appear in this text/list box.

Searching for Text Across Multiple Folders

Use the **In Folders** page to search for text in folder contents.

1. Click **Search > Find in Files**.

The Find in Files window appears.

2. Enter search text into the **Find** text/list box.
3. Enter replacement text into the **Replace with** text/list box.
4. Set general search options.

5. Set the **In Folders** page search options:

- a. Enter a folder path into the **Search in** text/list box, or click the **Browse** button to select a folder.
- b. Check or clear the **Search sub-folders** checkbox.
- c. Enter filename extensions into the **By type** text/list box.

6. Find and replace text:

- a. Click the **Find** or **Find All** button to search for matching text.

The IDE searches the specified folder contents until it finds a match or reaches the end of the search. A single match appears highlighted in an editor window, or multiple matches appear in a Search Results window. The IDE beeps if it does not find any matching text.

- b. Click the **Replace** or **Replace All** button to replace the matching text.

Click the Replace button to replace the current match. Click the Replace button repeatedly to replace subsequent matches. Click the Replace All button to replace all matching text.

To replace consecutive matches, click the Find button to find the first match, then repeatedly click the Replace button. To replace one match at a time, or to replace non-consecutive matches, click the Find button to find a match, then click the Replace button as needed.

TIP If you clicked the Find button to start the search, click **Search > Find Next** to find the next match.

Finding and Replacing Text

Multiple-File Find and Replace

In Projects

Use the In Projects page to search active projects and build targets for matching text. [Figure 12.5](#) shows the In Projects page. [Table 12.5](#) explains the items in the page.

Figure 12.5 Find in Files window—In Projects page

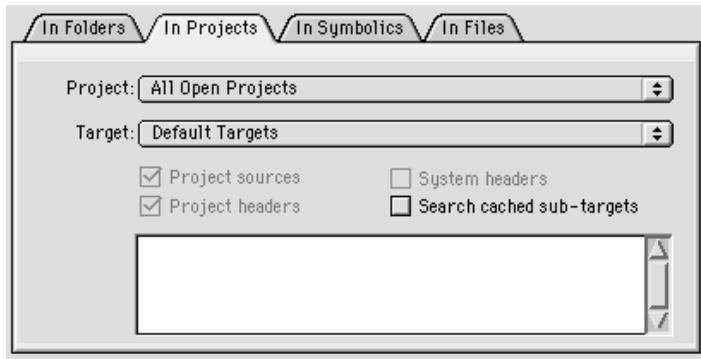


Table 12.5 Find in Files window—In Projects items

Item	Explanation
Project list box	Specify the projects that you want to search.
Target list box	Specify the build targets that you want to search.
Project sources checkbox	Check to search the source-code files of selected projects. Clear to ignore source-code files of selected projects.
Project headers checkbox	Check to search the header files of selected projects. Clear to ignore header files of selected projects.
System headers checkbox	Check to search system header files. Clear to ignore system header files.
Search cached sub-targets checkbox	Check to search sub-targets that the IDE cached for the selected build targets. Clear to ignore the sub-targets that the IDE cached for the selected build targets.
File list	This list shows files that the IDE will search. To remove a file from this list, select it and press Backspace or Delete. To open a file in this list, double-click its name.

Searching for Text Across Multiple Projects

Use the **In Projects** page to search for text in active projects and build targets.

1. Click **Project > Make**.

The IDE updates the project data to correctly list source-code files, header files, and build targets in the **In Projects** page of the **Find in Files** window.

2. Click **Search > Find in Files**.

The Find in Files window appears.

3. Enter search text into the **Find** text/list box.

4. Enter replacement text into the **Replace with** text/list box.

5. Set general search options.

6. Set the **In Projects** page search options:

- a. Use the **Project** list box to specify the projects that you want to search.

- b. Use the **Target** list box to specify the build targets that you want to search.

- c. Check or clear the checkboxes to refine your search criteria.

- d. Remove files from the File list as needed.

7. Find and replace text:

- a. Click the **Find** or **Find All** button to search for matching text.

The IDE searches the specified projects and build targets until it finds a match or reaches the end of the search. A single match appears highlighted in an editor window, or multiple matches appear in a Search Results window. The IDE beeps if it does not find any matching text.

- b. Click the **Replace** or **Replace All** button to replace the matching text.

Click the Replace button to replace the current match. Click the Replace button repeatedly to replace subsequent matches. Click the Replace All button to replace all matching text.

To replace consecutive matches, click the Find button to find the first match, then repeatedly click the Replace button. To replace one match at a time, or to replace non-consecutive matches, click the Find button to find a match, then click the Replace button as needed.

Finding and Replacing Text

Multiple-File Find and Replace

TIP If you clicked the Find button to start the search, click **Search > Find Next** to find the next match.

In Symbolics

Use the **In Symbolics** page to search files containing symbolics information for matching text. [Figure 12.6](#) shows the In Symbolics page. [Table 12.6](#) explains the items in the page.

Figure 12.6 Find in Files window—In Symbolics page

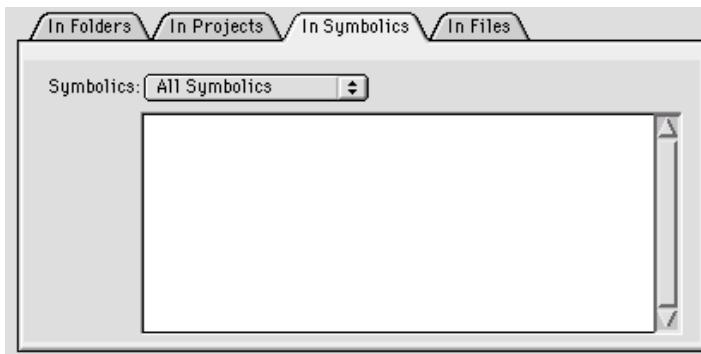


Table 12.6 Find in Files window—In Symbolics items

Item	Explanation
Symbolics list box	Specify the symbolics files that you want to search.
Symbolics list	This list shows the symbolics files that the IDE will search. To remove a file from this list, select it and press Backspace or Delete. To open a file in this list, double-click its name.

Searching for Text Across Multiple Symbolics Files

Use the **In Symbolics** page to search for text in symbolics files. You must generate browser data in order to search symbolics files.

1. Enable browser data for the build targets that you want to search.

Use the **Build Extras** target settings panel to **Generate Browser Data From** a compiler or language parser, then **Apply** or **Save** your changes. Configuring this option enables browser data.

2. Click **Project > Debug**.

Starting a debugging session causes the IDE to generate browser data for the project.

NOTE The IDE does not generate browser data for some files, such as libraries.

3. Click **Debug > Kill**.

The debugging session ends.

4. Click **Search > Find in Files**.

The Find in Files window appears.

5. Enter search text into the **Find** text/list box.

6. Enter replacement text into the **Replace with** text/list box.

7. Set general search options.

8. Set the **In Symbolics** page search options:

- a. Use the **Symbolics** list box to specify the symbolics files that you want to search.

- b. Remove symbolics files from the Symbolics list as needed.

9. Find and replace text:

- a. Click the **Find** or **Find All** button to search for matching text.

The IDE searches the specified symbolics files until it finds a match or reaches the end of the search. A single match appears highlighted in an editor window, or multiple matches appear in a Search Results window. The IDE beeps if it does not find any matching text.

Finding and Replacing Text

Multiple-File Find and Replace

- b. Click the **Replace** or **Replace All** button to replace the matching text.

Click the Replace button to replace the current match. Click the Replace button repeatedly to replace subsequent matches. Click the Replace All button to replace all matching text.

To replace consecutive matches, click the Find button to find the first match, then repeatedly click the Replace button. To replace one match at a time, or to replace non-consecutive matches, click the Find button to find a match, then click the Replace button as needed.

TIP If you clicked the Find button to start the search, click **Search > Find Next** to find the next match.

In Files

Use the In Files page to search file sets for matching text. [Figure 12.7](#) shows the In Files page. [Table 12.7 on page 139](#) explains the items in the page.

Figure 12.7 Find in Files window—In Files page

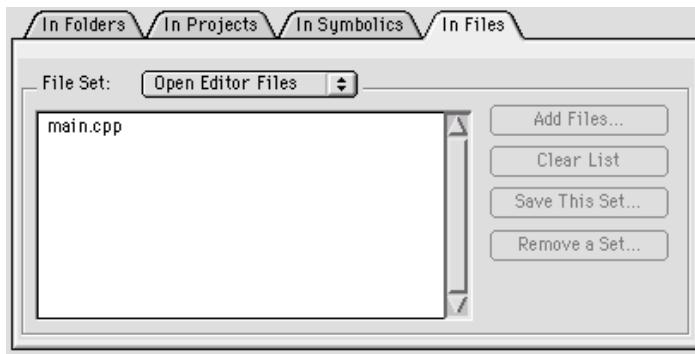


Table 12.7 Find in Files window—In Files items

Item	Explanation
File Set list box	Specify the file set that you want to search. Select New File Set to create a new set.
File Set list	This list shows the files that the IDE will search. To remove a file from this list, select it and press Backspace or Delete. To add files to this list, click the Add Files button, or drag and drop files and folders into the list. To open a file in this list, double-click its name.
Add Files button	Click to open a dialog box that lets you add files to the current file set. To enable this button, select from the File Set list box an existing file set or the New File Set option.
Clear List button	Click to clear the current File Set list. To enable this button, select from the File Set list box a file set that has at least one file.
Save This Set button	Click to save the current file set under a specific name. The file set must have at least one file. The name appears in the File Set list box. To enable this button, modify the current file set or select an existing file set from the File Set list box.
Remove a Set button	Click to open a dialog box that lets you remove file sets that you created previously. The removed file sets no longer appear in the File Set list box. To enable this button, select from the File Set list box an existing file set or the New File Set option.

Searching for Text Across Multiple Files

Use the **In Files** page to search for text in file sets.

1. Click **Search > Find in Files**.

The Find in Files window appears.

2. Enter search text into the **Find** text/list box.

3. Enter replacement text into the **Replace with** text/list box.

4. Set general search options.

5. Set the **In Files** page search options:

a. Use the **File Set** list box to specify the file set that you want to search.

b. Use the buttons to manage the File Set list as needed.

Finding and Replacing Text

Multiple-File Find and Replace

- c. Remove files from the File Set list as needed.
6. Find and replace text:
 - a. Click the **Find** or **Find All** button to search for matching text.

The IDE searches the specified files until it finds a match or reaches the end of the search. A single match appears highlighted in an editor window, or multiple matches appear in a Search Results window. The IDE beeps if it does not find any matching text.
 - b. Click the **Replace** or **Replace All** button to replace the matching text.

Click the Replace button to replace the current match. Click the Replace button repeatedly to replace subsequent matches. Click the Replace All button to replace all matching text.

To replace consecutive matches, click the Find button to find the first match, then repeatedly click the Replace button. To replace one match at a time, or to replace non-consecutive matches, click the Find button to find a match, then click the Replace button as needed.

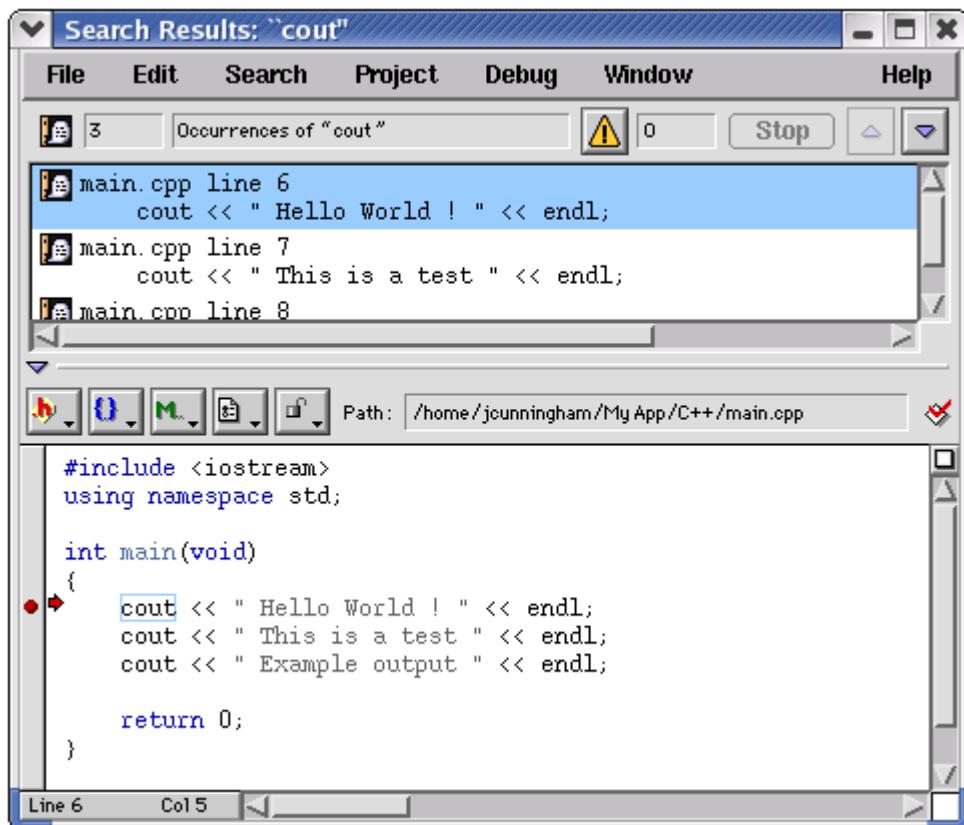
TIP If you clicked the Find button to start the search, click **Search > Find Next** to find the next match in the file.

Search Results Window

Use the **Search Results** window to explore multiple matches that the IDE finds. The IDE opens this window automatically after it finds multiple matches. Also use this window to stop searches in progress.

[Figure 12.8](#) shows the Search Results window. [Table 12.8](#) explains the items in the window.

Figure 12.8 Search Results window



Finding and Replacing Text

Search Results Window

Table 12.8 Search Results window—items

Item	Icon	Explanation
Result Count text box		Shows the total number of search results.
Search Criteria text box		Shows the search criteria.
Warnings button		Click to display compiler and linker warnings in the Results pane. The text box to the right of this button shows the total number of warnings.
Stop button		Click to stop the search in progress.
Previous Result button		Click to select the previous search result.
Next Result button		Click to select the next search result.
Results pane		Lists individual search results.
Source Code pane disclosure triangle		Click to show or hide the Source Code pane.
Pane resize bar		Drag to resize the Results and Source Code panes.
Source Code pane		Shows the source code corresponding to the selected item in the Results pane. This pane operates the same as an editor window without pane-splitter controls.

Text-Selection Find

After you use the **Find**, **Find and Replace**, or **Find in Files** windows to perform a successful search, you can use menu commands to apply the same search criteria to additional searches. This way, you do not have to open the windows again to use the same search criteria. You select text in the active editor window to define the search string.

Using the Find Next Command

When searching for text, you can use the **Find Next** command to have the IDE find the next match:

1. Start a search with the **Find**, **Find and Replace**, or **Find in Files** windows.
2. After the IDE finds a match, click **Search > Find Next** to find a subsequent match.

NOTE Find Next always searches forward and ignores the Search up checkbox.

Using the Find Previous Command

When searching for text, you can use the **Find Previous** command to have the IDE find the previous match. You must enable the Find Previous command in the **Customize IDE Commands** window.

1. Click **Edit > Commands & Key Bindings**.

The Customize IDE Commands window opens.

2. Click the **Commands** tab in the Customize IDE Commands window.
3. Expand the **Search** item in the **Commands** pane tree structure.
4. Select the **Find Previous** item in the expanded list.

Scroll as needed in order to see the Find Previous item. After you select the Find Previous item, its settings appear in **Details** pane.

Finding and Replacing Text

Text-Selection Find

5. Check the **Appears in Menus** checkbox.

The Find Previous command will appear in the **Search** menu in the main IDE menu bar.

6. Click **Save** to confirm your changes.

7. Close the **Customize IDE Commands** window.

You can now select the Find Previous command in the Search menu. You can also use the key binding associated with the command.

NOTE (Macintosh) Hold down the Shift key in order to click **Search > Find Previous**.

Changing the Find String

Use the **Enter Find String** command to change the current find string.

1. Select the text that you want to use as the new find string.
2. Click **Search > Enter Find String**.

The selected text replaces the find string that you specified in the **Find**, **Find and Replace**, or **Find in Files** windows.

You can now use the new find string to perform find and replace operations.

Searching with a Text Selection

Use the **Find Selection** command to search the active editor window for selected text.

1. Select the text that you want to use as the search string.
2. Click **Search > Find Selection**.

The IDE searches the active editor window until it finds a match or reaches the end of the search. A single match appears highlighted in the editor window. The IDE beeps if it does not find any matching text.

You can also use the **Find Next** and **Find Previous** commands to search for additional matching text.

Regular-Expression Find

Use regular expressions to search text according to sophisticated text-matching rules. A *regular expression* is a text string used as a mask for matching text in a file. To use regular expressions, select **Regular expression** in the **Find**, **Find and Replace**, or **Find in Files** windows. Certain characters are operators with special meanings in a regular expression.

TIP For an in-depth description of regular expressions, refer to *Mastering Regular Expressions* by Jeffrey E.F. Friedl, published by O'Reilly & Associates, Inc. On a UNIX system, also refer to the man pages for `regexp`.

[Table 12.9](#) explains the regular-expression operators that the IDE recognizes.

Table 12.9 Regular-expression operators recognized by the IDE

Operator	Name	Explanation
.	match any	Matches any single printing or non-printing character except newline and null.
*	match zero or more	Replaces the smallest/preceding regular expression with a sub-expression.
+	match one or more	Repeats the preceding regular expression at least once and then as many times as necessary to match the pattern.
?	match zero or one	Repeats the preceding regular expression once or not at all.
\n	back reference	Refers to a specified group (a unit expression enclosed in parentheses) in the find string. The digit <i>n</i> identifies the <i>n</i> th group, from left to right, with a number from 1 to 9.
	alternation	Matches one of a choice of regular expressions. If this operator appears between two regular expressions, the IDE matches the largest union of strings.

Finding and Replacing Text

Regular-Expression Find

Table 12.9 Regular-expression operators recognized by the IDE (*continued*)

Operator	Name	Explanation
^	match beginning of line	Matches items from the beginning of a string or following a <code>newline</code> character. This operator also represents a NOT operator when enclosed within brackets.
\$	match end of line	Matches items from the end of a string or preceding a <code>newline</code> character.
[. . .]	list	Defines a set of items to use as a match. The IDE does not allow empty lists.
(. . .)	group	Defines an expression to be treated as a single unit elsewhere in the regular expression.
-	range	Specifies a range. The range starts with the character preceding the operator and ends with the character following the operator.

[Table 12.10](#) shows various examples of using regular expressions to match particular text in a text sample.

Table 12.10 Examples of using regular expressions

Example Type	This regular expression...	...matches this text...	...in this text sample:
Matching simple expressions	ex	ex	sample text
	[() [.] stack [()]]	(. stack)	ADDR(. stack)
Matching any character	var.	var1 var2	cout << var1; cout << var2;
	c.t	cut cot	cin >> cutF; cin >> cotG;
Repeating expressions	s*ion	ion ssion	information the session
	s+ion	sion ssion	confusion the session

Table 12.10 Examples of using regular expressions (*continued*)

Example Type	This regular expression...	...matches this text...	...in this text sample:
Grouping expressions	ris	ris	surprise
	r(i)s	r is	theVar is
Choosing one character from many	[bls]ag	sag bag lag	sagging bag lagged
	[[aeiou][0-9]]	[2 u9	cout << a[2] << u9;
	[^bls]ag	rag	sagging rag lagged
	[-ab]V	aV -v	aVal-Val;
Matching line beginnings and endings	^([\t]* cout)	cout cout	cout << "no tab"; cout << "tab";
	(l*;) \$	l; ;	a-ct; a = battLvl; b-ct;

Using the Find String in the Replace String

Use the & operator to incorporate matching text into a replacement string. The IDE substitutes the matching text for the & operator. Use \& to indicate a literal ampersand in the replacement string.

[Table 12.11](#) shows examples of using the find string in the replace string of regular expressions.

Table 12.11 Examples of using the find string in the replace string

Find string	Replace string	Matching text	After replacement
var[0-9]	my_&	var1	my_var1
tgt	\&target	tgt	&target

Remembering Sub-expressions

Use the `\n` construct to recall sub-expressions from the find string in the replacement string. The digit n ranges from 1 to 9 and represents the nth sub-expression in the find string, counting from left to right. Enclose each sub-expression in parentheses.

Consider these sample definitions:

- Find string: `\#define [\t]+(.+) [\t]+([0-9]+);`
- Replace string: `const int \1 = \2;`
- Sub-expression `\1`: `(.+)`
- Sub-expression `\2`: `([0-9]+)`

These definitions show a replacement operation that recalls two sub-expressions. [Table 12.12](#) shows the result of applying these sample definitions to some text.

Table 12.12 Remembering sub-expressions

Before replacement	\1 matches this text	\2 matches this text	After replacement
<code>#define var1 10;</code>	<code>var1</code>	<code>10</code>	<code>const int var1 = 10;</code>
<code>#define a 100;</code>	<code>a</code>	<code>100</code>	<code>const int a = 100;</code>

Comparing Files and Folders

The IDE can compare files or folder contents and graphically show you the differences between them. You can perform these tasks:

- Compare two files.
- Compare the contents of two folders.

You perform the comparison by specifying a *source* item and a *destination* item. You can apply or unapply the changes in the source item to the destination item.

Comparison Setup

You use the **Compare Files Setup** window to enter information about the files or folders that you want to compare. [Figure 12.9 on page 149](#) shows the Compare Files Setup window. [Table 12.13 on page 149](#) explains items in the window.

Figure 12.9 Compare Files Setup window

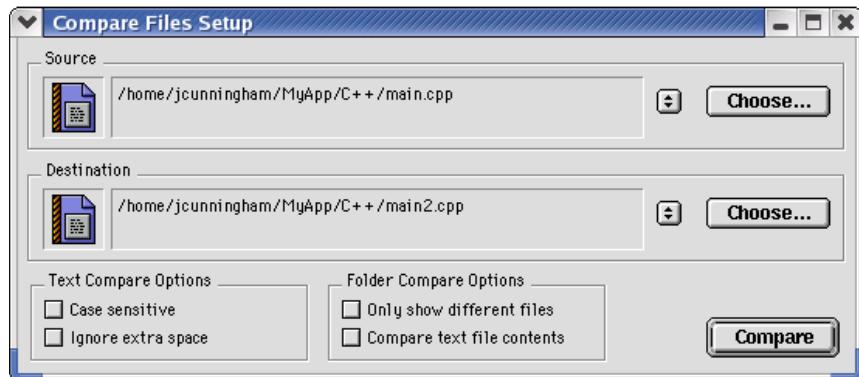


Table 12.13 Compare Files Setup window—items

Item	Explanation
Source box	Click the Choose button to specify the source file or folder for the comparison, or drag and drop a file or folder into the box. Click the selector to the left of the Choose button to specify a file in an open editor window.
Destination box	Click the Choose button to specify the destination file or folder for the comparison, or drag and drop a file or folder into the box. Click the selector to the left of the Choose button to specify a file in an open editor window.
Case sensitive checkbox	<p>Check to consider text case during the compare operation. The comparison distinguishes between a capital letter and the same letter in lower case.</p> <p>Clear to disregard text case during the compare operation. The comparison does not distinguish between a capital letter and the same letter in lower case.</p>

Finding and Replacing Text

Comparing Files and Folders

Table 12.13 Compare Files Setup window—items (continued)

Item	Explanation
Ignore extra space checkbox	<p>Check to consider extra spaces and tabs during the compare operation. The comparison distinguishes differences in the number of spaces and tabs in the compared files.</p> <p>Clear to disregard extra spaces and tabs during the compare operation. The comparison does not distinguish differences in the number of spaces and tabs in the compared files.</p>
Only show different files checkbox	<p>Check to have the Folder Compare Results window show only the differences between the compared folders. The Files in Both Folders pane stays blank.</p> <p>Clear to have the Folder Compare Results window show all files from the compared folders as well as the differences between those folders. The Files in Both Folders pane shows the common files between the compared folders.</p>
Compare text file contents checkbox	<p>Check to identify differences in terms of a byte-by-byte comparison of the files.</p> <p>Clear to identify differences in terms of only the sizes and modification dates of the files.</p>
Compare button	Click to compare the specified files or folders.

Choosing Files to Compare

Use the **Compare Files** command to specify two files that you want to compare.

1. Click **Search > Compare Files**.

The **Compare Files Setup** window appears.

2. Specify a source file for the comparison.

Click the **Choose** button in the **Source** box or drag and drop the file into the **Source** box. To specify a file in an open editor window, click the selector in the **Source** box.

3. Specify a destination file for the comparison.

Click the **Choose** button in the **Destination** box or drag and drop the file into the Destination box. To specify a file in an open editor window, click the selector in the Destination box.

4. Configure the checkboxes in the **Text Compare Options** group.

Check the **Case sensitive** checkbox to distinguish between a capital letter and the same letter in lower case. Check the **Ignore extra space** checkbox to disregard extra spaces or tabs in the files.

5. Click the **Compare** button.

The IDE performs the file comparison. The **File Compare Results** window appears.

Choosing Folders to Compare

Follow these steps to specify two folders that you want to compare:

1. Click **Search > Compare Files**.

The **Compare Files Setup** window appears.

2. Specify a source folder for the comparison.

Click the **Choose** button in the **Source** box or drag and drop the folder into the Source box.

3. Specify a destination folder for the comparison.

Click the **Choose** button in the **Destination** box or drag and drop the folder into the Destination box.

4. Configure the checkboxes in the **Text Compare Options** group.

These options apply to the files inside the compared folders. Check the **Case sensitive** checkbox to distinguish between a capital letter and the same letter in lower case. Check the **Ignore extra space** checkbox to disregard extra spaces or tabs in the files.

5. Configure the checkboxes in the **Folder Compare Options** group.

These options apply to the contents of the compared folders. Check the **Only show different files** checkbox to have the **Folder Compare Results** window show only the files that differ between the source folder and

Finding and Replacing Text

Comparing Files and Folders

destination folder. Check this option to have the **Files in Both Folders** pane of the Folder Compare Results window stay blank.

Check the **Compare text file contents** checkbox to have the IDE perform a content-based comparison of the text files in the compared folders. Check this option to have the Folder Compare Results window show differences in terms of file content instead of file sizes and modification dates.

6. Click the **Compare** button.

The IDE performs the folder comparison. The **Folder Compare Results** window appears.

CAUTION The compare operation ignores folders matching the criteria that you specify in the **Shielded Folders** preference panel.

File Comparison

The IDE file-comparison feature identifies additions, changes, and deletions between two text files. In addition, this feature allows you to apply the differences in the source file to the destination file.

You can also use this feature to merge changes between two versions of the same text file. Specify one version of the text file as the source file and the other version of the text file as the destination file. Then you can apply changes from the source file to the destination file. The destination file becomes the merged file.

After you use the **Compare Files Setup** window to specify two files for comparison, click the **Compare** button. The **File Compare Results** window appears. This window shows the differences between the source file and destination file. You can apply or unapply those differences to the destination file.

The File Compare Results window shows file differences in the form of highlighted portions of text. The highlighting tracks with the text as you scroll through the compared files.

[Figure 12.10 on page 153](#) shows the File Compare Results window. [Table 12.14 on page 153](#) explains the items in the window.

Figure 12.10 File Compare Results window

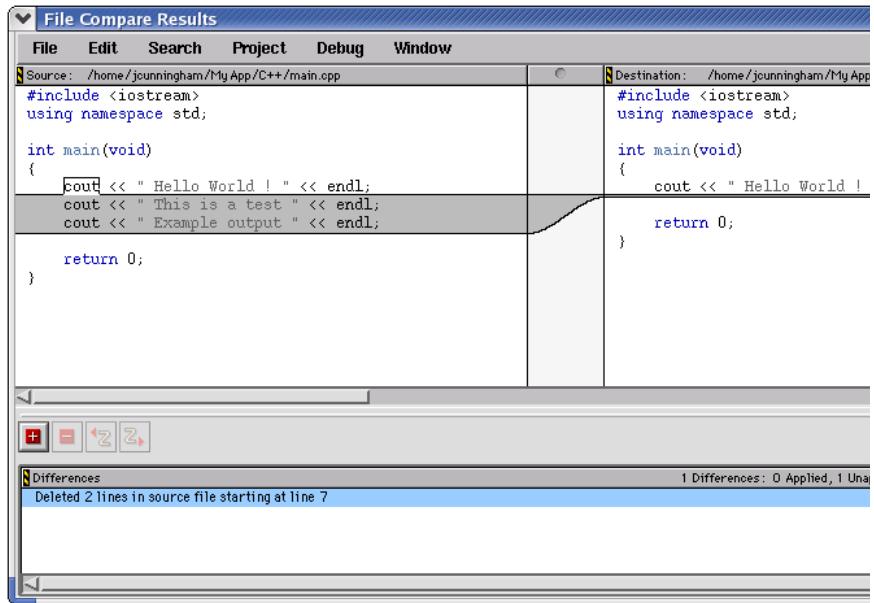


Table 12.14 File Compare Results window—items

Item	Icon	Explanation
Source pane	N/A	Shows the contents of the source file. You cannot edit the contents of this pane.
Destination pane	N/A	Shows the contents of the destination file. You can edit the contents of this pane.
Pane resize bar		Drag to resize the Source and Destination panes.
Apply button		Click to apply the selected Differences pane items to the destination file.
Unapply button		Click to unapply the selected Differences pane items from the destination file.
Undo button		Click to undo your last text edit in the Destination pane.

Finding and Replacing Text

Comparing Files and Folders

Table 12.14 File Compare Results window—items (continued)

Item	Icon	Explanation
Redo button		Click to redo your last text edit in the Destination pane.
Differences pane	N/A	Shows the differences between the Source pane and the Destination pane. Select an item to highlight it in the Source and Destination panes. Applied items appear in an italicized font

Applying File Differences

Use the **Apply Difference** command to apply the selected items in the **Differences** pane to the destination file.

NOTE You cannot alter the source file. You can change the destination file by applying differences from the source file or by editing the contents of the **Destination** pane.

1. Select the items in the Differences pane that you want to apply to the destination file.
2. Click **Search > Apply Difference** or click the **Apply** button in the **File Compare Results** window.

The **Destination** pane updates to reflect the differences that you applied to the destination file. The applied items in the Differences pane change to an italicized font.

TIP Use the **Customize IDE Commands** window to assign a key binding to the **Apply Difference** command. This way, you can use the keyboard to apply differences.

Unapplying File Differences

Use the **Unapply Difference** command to unapply the selected items in the **Differences** pane from the destination file.

NOTE You cannot alter the source file. You can change the destination file by unapplying differences from the source file or by editing the contents of the **Destination** pane.

1. Select the items in the Differences pane that you want to unapply from the destination file.
Items that you can unapply appear in an italicized font.
2. Click **Search > Unapply Difference** or click the Unapply button in the **File Compare Results** window.

The **Destination** pane updates to reflect the differences that you unapplied from the destination file. The unapplied items in the Differences pane no longer appear in an italicized font.

TIP Use the **Customize IDE Commands** window to assign a key binding to the **Unapply Difference** command. This way, you can use the keyboard to unapply differences.

Folder Comparison

The IDE folder-comparison feature identifies the differences between the contents of two folders. It reports the files in both folders, the files only in the source folder, and the files only in the destination folder.

You can also use this feature to analyze the differences between two different releases of a folder of software. Specify one release of the software folder as the source folder and the other release of the software folder as the destination folder. Then you can analyze the differences between the source and destination folders.

After you use the **Compare Files Setup** window to specify two folders for comparison, click the **Compare** button. The **Folder Compare Results** window appears and shows the differences between the source folder and destination folder.

Finding and Replacing Text

Comparing Files and Folders

The Folder Compare Results window shows folder differences in the form of three panes. Italicized items in these panes indicate non-text files.

[Figure 12.11](#) shows the Folder Compare Results window. [Table 12.15](#) explains the items in the window.

Figure 12.11 Folder Compare Results window

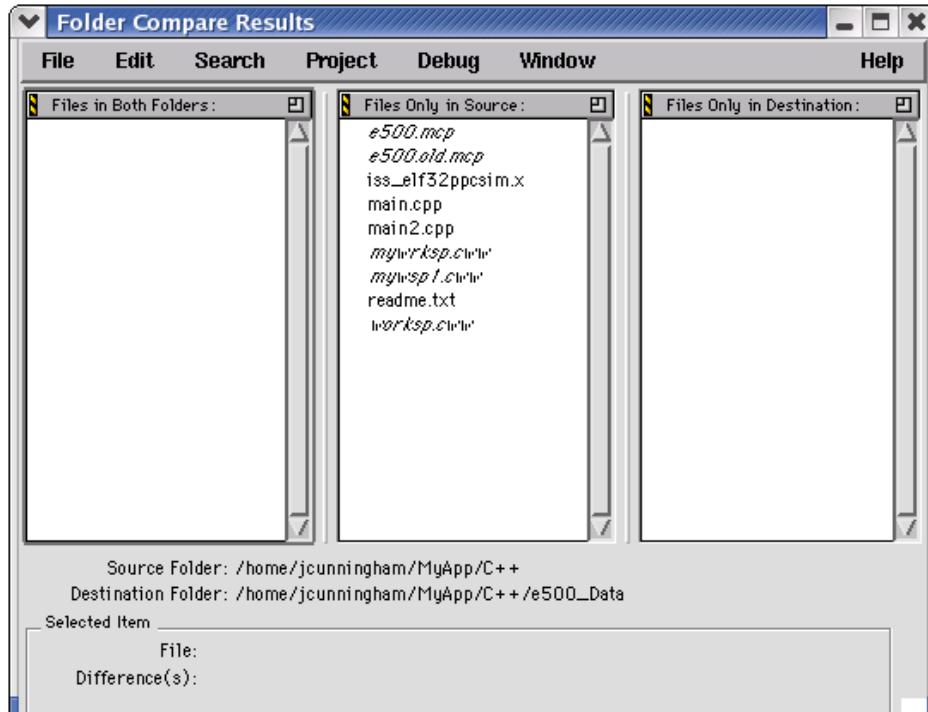


Table 12.15 Folder Compare Results window—items

Item	Icon	Explanation
Pane Expand box		Click to enlarge the pane to fill the window.
Pane Collapse box		Click to reduce an expanded pane to its original size.

Table 12.15 Folder Compare Results window—items (*continued*)

Item	Icon	Explanation
Pane resize bar		Drag to resize the panes on either side of the bar.
Files in Both Folders pane	N/A	Shows the items that are in both the source folder and the destination folder. A bullet next to an item indicates that the item content differs between the two folders.
Files Only in Source pane	N/A	Shows the items that are in the source folder only.
Files Only in Destination pane	N/A	Shows the items that are in the destination folder only.
Selected item group	N/A	Shows file and difference information for the selected item in the window panes.

Examining Items in the Folder Compare Results Window

You can use the **Folder Compare Results** window to open text files and compare file differences.

Double-click a text file to view and change its contents in an editor window.

A file whose contents differ between the source and destination folders has a bullet next to its name. Double click the file to open a **File Comparison Results** window. Use this window to examine the differences between the file contents.

Finding and Replacing Text

Comparing Files and Folders

Browser

This section contains these chapters:

- [Using the Browser](#)
- [Using Class Browser Windows](#)
- [Using Other Browser Windows](#)
- [Using Browser Wizards](#)

Using the Browser

This chapter explains how to work with the browser in the CodeWarrior™ IDE. Use the browser to perform these tasks:

- Generate a browser database—the browser stores collected symbol information in a browser database for the project. You can generate browser data from the compiler or the language parser.
- Collect symbol information—symbols include functions, variables, and objects. Enable the browser to collect information about the symbols in a project.

Read this chapter to learn more about typical tasks for working with the browser.

This chapter contains these sections:

- [“Browser Database” on page 161](#)
- [“Browser Symbols” on page 164](#)

Browser Database

The browser database contains information about symbols in a program, which include (depending on program language) global variables, functions, classes, and type declarations, among others.

Some IDE windows require that the project contain a browser database. For example, the **Class Hierarchy** window only displays information for a project that contains a browser database. This section explains how to configure a project to generate its browser database.

NOTE Generating a browser database increases the project’s size. To minimize the project’s size, generate the browser database only for targets you frequently use.

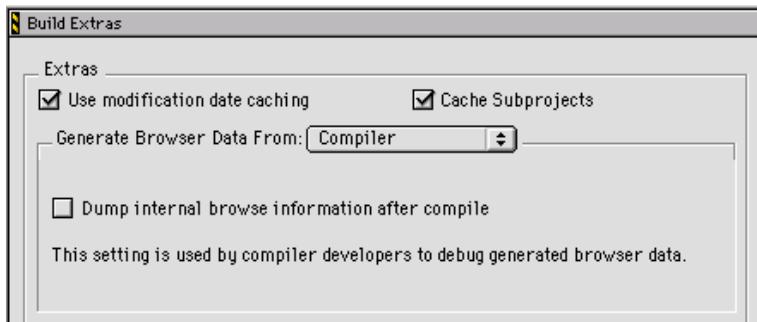
Browser Data

Browser data contains symbolic and relationship information about the project code. The browser uses this data to access the code information.

Use the **Generate Browser Data From** menu ([Figure 13.1](#)) in the **Build Extras** target settings panel to enable and disable browser data generation. This drop-down menu provides these options, which determine how the IDE generates browser data:

- **None**—The IDE does not generate browser data. Use **None** to *disable* browser data. Select None to generate faster compiles (with no browser features).
- **Compiler**—The Compiler generates the browser data. While it compiles more slowly, the compiler generates the most accurate browser data.
- **Language Parser**—The Code Completion plug-in associated with the project's programming language generates the browser data.

Figure 13.1 Generate Browser Data From menu



Generating Browser Data

You can select an option in the **Generate Browser Data From** drop-down menu to establish what the IDE uses to generate browser data for a project file.

To generate browser data, follow these steps:

1. Choose **Edit > Target Settings**.
2. From the **Target Settings Panels** list, select **Build Extras**.

3. Choose **Compiler** or **Language Parser** from the **Generate Browser Data From** menu.

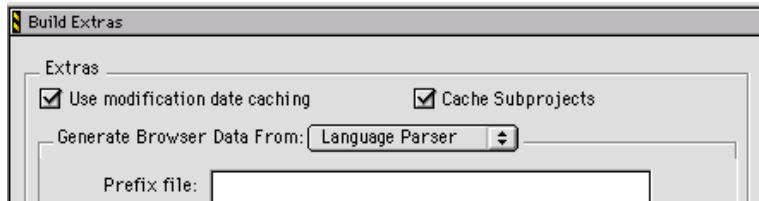
NOTE Some compilers do not generate browser data.

- a. **Compiler**—The compiler generates browser data and the following associated item appears.
If you enable **Dump internal browse information after compile**, the generated browser data appears in a log window after you compile a file.
- b. **Language Parser**—The **Code Completion** plug-in associated with the project's programming language generates the browser data. Browser data and the #include pop-up window update as you edit.

NOTE Choose Language Parser for C/C++ code completion.

The **Prefix** and **Macro** files ([Figure 13.2](#)) are applicable to C/C++ Code Completion.

Figure 13.2 Generate browser data from language parser



- **Prefix file**—Similar to that used in the **C/C++ Language Settings** panel, the Prefix file contains header files that help the C/C++ Code Completion plug-in parse code. The Prefix file should only include text files (not pre-compiled header files).
- **Macro file**—Contains C/C++ macro files that help the Code Completion plug-in resolve any #ifdefs found in the source code or in the header files.

4. If you selected **Compiler**, choose **Project > Bring Up To Date** or **Make**.

The IDE generates browser data for the project.

If you selected **Language Parser**, the IDE generates browser data in the background.

Using the Browser

Browser Symbols

Disabling Browser Data

Select **None** to disable browser data and stop the IDE from generating browser information for the project.

1. Choose **Edit > Target Settings**.
2. Select **Build Extras** from the **Target Settings Panels** list.
3. In the **Generate Browser Data From** drop-down menu, select **None**.
4. Click **Save**.
5. Choose **Project > Make**.

The IDE stops generating browser information.

Browser Symbols

Navigate browser symbols to open browser views, find symbol definitions, and examine inheritance.

You can navigate browser symbols in these ways:

- Use the Browser contextual menu to open various browser windows for a selected symbol.
- Double-click a symbol name in the Class Browser window to open the file that contains the declaration of that symbol.
- Use the class hierarchy windows to determine the ancestors or descendants of a selected symbol.

Browser Contextual Menu

Use the IDE's browser contextual menu to enhance source-code editing in the IDE. Use this menu to streamline text entry in editor windows. You can enter the first few letters of a function name, then use the browser contextual menu to complete the entry.

Using the Browser Contextual Menu

1. Open the browser contextual menu, as explained in [Table 13.1](#).

Table 13.1 Opening a browser contextual menu

On this host...	Do this...
Windows	Right-click a symbol name.
Macintosh	Click and hold on a symbol name.
Solaris	Click and hold on a symbol name.
Linux	Click and hold on a symbol name.

2. Select a command from the contextual menu.

Identifying Symbols in the Browser Database

As a shortcut, you can use browser coloring to help recognize if a symbol resides in the browser database. When you activate a browser, you can see browser-database symbols because they appear in the editor and browser windows according to the colors you select.

TIP The default color setting is identical for all eight types of browser-database symbols. You can choose a different color for each symbol type.

To change the browser symbol colors the editor uses, follow these steps:

1. Choose **Edit > Preferences**.
2. Select the **Text Colors** panel from the **IDE Preference Panels** list.
3. Select the **Activate Syntax Coloring** option.
4. Select the **Activate Browser Coloring** option.
5. Click the color swatch next to the symbol name to set that symbol's color.
6. Click **Save**.

Using the Browser

Browser Symbols

Using Class Browser Windows

This chapter explains how to work with the Class Browser windows in the CodeWarrior™ IDE. Use the Class Browser to perform these tasks:

- View browser data—the class browser collects information about the elements of a computer program. Such elements include functions, variables, and classes. The class browser displays these elements in organized lists.
- Show data relationships—the class browser shows the relationships between classes, data members, and methods. The class browser also updates the display to reflect changes in class scope.

Read this chapter to learn more about typical tasks for working with Class Browser windows.

This chapter contains these sections:

- [“Class Browser window” on page 167](#)
- [“Classes pane” on page 174](#)
- [“Member Functions pane” on page 175](#)
- [“Data Members pane” on page 176](#)
- [“Source pane” on page 177](#)
- [“Status Area” on page 178](#)

Class Browser window

Use the Class Browser window to view information about the elements of a computer program. This section explains how to use the Class Browser window to view browser data.

Using Class Browser Windows

Class Browser window

[Figure 14.1 on page 168](#) shows the Class Browser window. [Table 14.1 on page 168](#) explains the items in the window. [Table 14.2 on page 170](#) explains the options in the Browser Access Filters list box.

Figure 14.1 Class Browser window

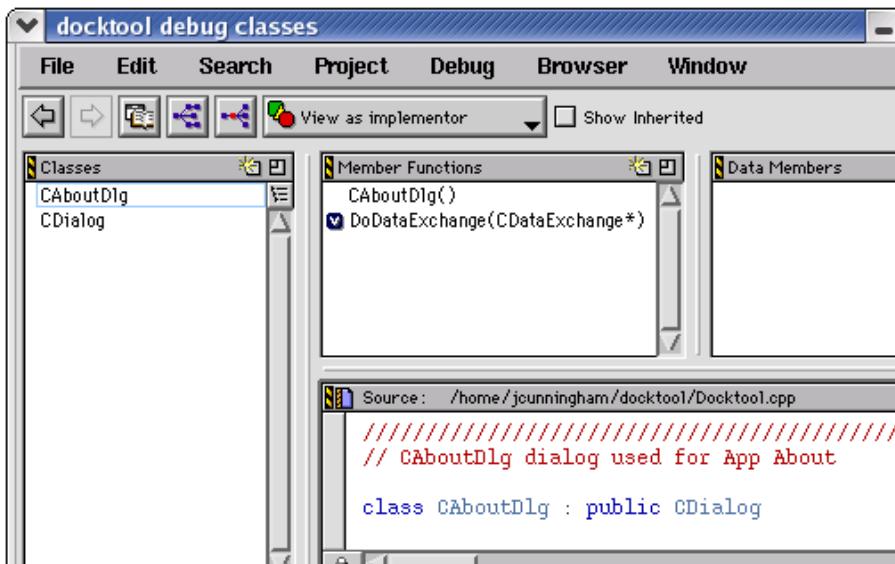


Table 14.1 Class Browser window—items

Item	Icon	Explanation
Go Back button		Click to return to the preceding browser view.
Go Forward button		Click to move to the succeeding browser view.
Browser Contents button		Click to open the Browser Contents window.
Class Hierarchy button		Click to open the Multi-class Hierarchy window.

Using Class Browser Windows

Class Browser window

Table 14.1 Class Browser window—items (*continued*)

Item	Icon	Explanation
Single Class Hierarchy Window button		Click to open the Single-class Hierarchy window for the selected class.
Browser Access Filters list box		Select filters for displaying items in class-browser panes.
Show Inherited	<input checked="" type="checkbox"/> Show Inherited	Select to show inherited items in the Member Functions pane and Data Members pane . Clear to hide inherited items from these panes.
Classes pane		Lists all classes in the project browser database.
Member Functions pane		Lists all member functions defined in the currently selected class.
Data Members pane		Lists all data members defined in the selected class.
Source pane		Displays source code for the currently selected item.
Status Area		Displays various status messages and other information.
Display toggle buttons	Alphabetical Hierarchical	Toggles the Classes display between alphabetical and hierarchical listings.
New Item button		Opens wizards to create new items (e.g., classes, data members, member functions).
Pane Expand box		Expands the pane to the width of the full window.
Pane Collapse Box		Collapses the pane to its original size.
Classes Pane button		Lists all classes in the project browser database.

Using Class Browser Windows

Class Browser window

Table 14.1 Class Browser window—items (*continued*)

Item	Icon	Explanation
Class Declaration button		Opens a window that shows declarations for all classes in the project.
Open File button		Opens the current source file in a new editor window.
VCS list pop-up		With a version control system enabled, choose the version-control command to execute on the displayed source file.

Table 14.2 Browser access filters

Filter	Icon	Show items with this access:		
		Public	Private	Protected
View as implementor		•	•	•
View as subclass		•		•
View as user		•		
Show public		•		
Show protected				•
Show private			•	

Viewing Class Data from Browser Contents Window

To view class data for a project in the **Browser Contents** window, follow these steps:

Using Class Browser Windows

Class Browser window

1. Open the **Browser Contents** window, as explained in [Table 14.3 on page 171](#).

Table 14.3 Opening the Browser Contents window

On this host...	Do this...
Windows	Select View > Browser Contents .
Macintosh	Select Window > Browser Contents .
Solaris	Select Window > Browser Contents .
Linux	Select Window > Browser Contents .

2. Select a class in the Browser Contents window.
3. Open a contextual menu for the selected class, as explained in [Table 14.4](#).

Table 14.4 Opening a contextual menu for the selected class

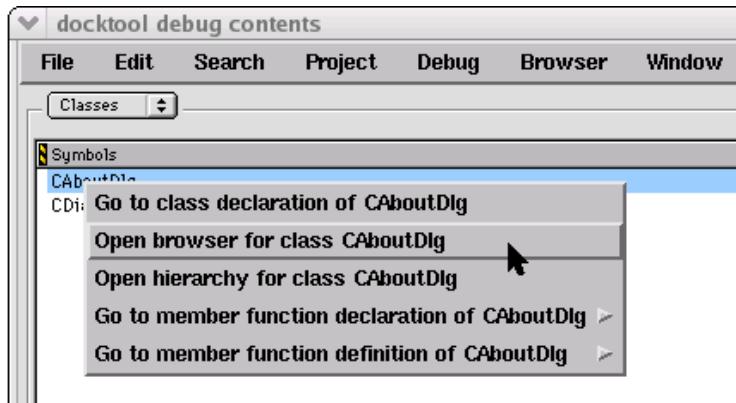
On this host...	Do this...
Windows	Right-click the selected class.
Macintosh	Control-click the selected class.
Solaris	Click and hold on the selected class.
Linux	Click and hold on the selected class.

A contextual menu like the one shown in [Figure 14.2](#) appears.

Using Class Browser Windows

Class Browser window

Figure 14.2 Browser Contents window—contextual menu



4. Select **Open browser for class *classname*** from the contextual menu.

The *classname* is the name of the class that you selected.

A **Class Browser** window appears.

Viewing Class Data from Hierarchy Windows

To view class data from a hierarchy window, follow these steps:

1. Open a **Single-Hierarchy** or **Multi-Class Hierarchy** window:

- a. Click the **Single Class Hierarchy Window** button  in the browser toolbar, or
- b. Click the **Class Hierarchy** button  in the browser toolbar.

2. In the Single- or Multi-Class Hierarchy window, double-click a class name.

A **Class Browser** window appears.

Expanding Browser Panes

Click the **Pane Expand** box (just above the scroll bar in the upper right-hand corner of the pane) to expand the Classes, Function Members, Data Members, or Source panes in a **Browser** window.

1. Click the **Pane Expand** box  to expand a pane.

This pane expands to fill the **Browser** window.

2. Use the enlarged pane to view data.

Alternately, you can use the resize bar between the panes to enlarge each pane.

1. Rest the cursor over the resize bar.

The cursor icon changes to this: 

2. Hold down the mouse button.
3. Drag the resize bar to enlarge or shrink the pane.

Collapsing Browser Panes

Click the **Pane Collapse** box (just above the scroll bar in the upper right-hand corner of the pane) to collapse the Classes, Function Members, Data Members, or Source panes in a **Browser** window.

1. Click the **Pane Collapse** box  to collapse a pane.

The chosen pane collapses to its original size.

2. You can now view other panes in a Browser window.

Alternately, you can use the resize bar between the panes to collapse each pane.

1. Rest the cursor over the resize bar.

The cursor icon changes to this: 

2. Hold down the mouse button.
3. Drag the resize bar to collapse the pane.

Using Class Browser Windows

Classes pane

Classes pane

Use the Classes pane to perform these tasks:

- Create a new class
- Toggle viewing of classes
- Sort classes

[Figure 14.1 on page 168](#) shows the Classes pane. [Table 14.5](#) explains the items in the pane.

Table 14.5 Classes pane—items

Item	Icon	Explanation
New Item		Click to create a new class using the New Class Wizard.
Sort Alphabetical		Click to sort the Classes list in alphabetical order.
Sort Hierarchical		Click to sort the Classes list in hierarchical order.

Creating a New Class

Use the **New Class** wizard to specify the name, declaration, and location for a new class. Click **Finish** in any screen to apply default values to any remaining parameters and complete the process. The New Class wizard creates the files that define the class.

1. From the Classes pane, click the **New Item** button .
2. Enter the **Name** and **Location** in the New Class window.
3. To create a more complex class, click **Next** (optional).
Follow the on-screen directions to further define the class.
4. Click **Finish** to complete the New Class process.

Showing the Classes Pane

Use the **Show Classes** button to expand the Classes pane.

1. Click the **Show Classes** button: 
2. The Classes pane appears in the **Class Browser** window.

Hiding the Classes Pane

Use the **Hide Classes** button to collapse the Classes pane.

1. Click the **Hide Classes** button: 
2. The Classes pane disappears from the **Class Browser** window.

Sorting the Classes List

Use the **Sort Alphabetical** and **Sort Hierarchical** commands to specify the sort order of classes in the Classes pane. The displayed icon always represents the alternate sort order. For example, when the Classes list appears in alphabetical order, the Sort Hierarchical icon is visible.

- Click the **Sort Alphabetical** icon  .
The IDE sorts the Classes list in alphabetical order.
- Click the **Sort Hierarchical** icon  .
The IDE sorts the Classes list in hierarchical order.

Member Functions pane

Use the **Member Functions** pane to perform these tasks:

- Create a new member function
- Determine the inheritance type of a member function

Using Class Browser Windows

Data Members pane

Table 14.6 Member Function and Data Member identifier icons

Meaning	Icon	The member is...
static		a static member
virtual		a virtual function that can be overridden, or an override of an inherited function
pure virtual or abstract		a member function that must be overridden in a subclass to create instances of that subclass

Creating a New Member Function

Use the **New Member Function** wizard to specify the name, return type, and parameters for a new member function. Click **Finish** in any screen to apply default values to any remaining parameters and complete the process.

1. Click the **New Item** button
2. Enter the **Member Function Declarations** in the **New Member Function** window.
3. Click **Next**.
4. Enter **Member function file locations** and **Include Files** information.
5. Click **Finish**.
6. Review the settings summary, then click **Generate**.

The IDE adds the new member function to the class declaration.

Data Members pane

Use the **Data Members** pane to create a new data member. This section explains how to create the data member.

Click the New Item button in the Data Members pane to open the New Data Member wizard. See [Table 14.6](#) for a complete list of identifier icons that appear in the Data Members pane.

Creating a New Data Member

Use the **New Data Member** wizard to specify the name, type, and initializer for the new data member. Specify other options to further refine the data member. Click **Finish** in any screen to apply default values to any remaining parameters and complete the process.

1. From the **Data Members** pane, click the **New Item** button: 
2. Enter the **Data Member Declarations** in the **New Data Member** window.
3. Click **Next**.
4. Enter Data Member file locations and `#include files` information.
5. Click **Finish**.
6. Review the settings summary, then click **Generate**.

The IDE adds the new data member to the class declaration.

Source pane

Use the **Source** pane to view the source code that corresponds to the selected class, member function, or data member. This section explains the items in the **Source** pane.

[Figure 14.1 on page 168](#) shows the Source pane. [Table 14.7](#) explains the items in the pane.

For information on editing source code, see [“Editing Source Code” on page 97](#).

Table 14.7 Source pane—items

Item	Icon	Explanation
Open File		Click to open the current source file in a new editor window.
VCS menu		Enable a version-control system in order to activate this menu. Use this menu to select and execute a version-control command on the source file.

Using Class Browser Windows

Status Area

Status Area

Use the status area to perform these tasks:

- Toggle viewing of the **Classes** pane
- View class declarations
- View classes according to public, private, or protected access

[Figure 14.1 on page 168](#) shows the status area. [Table 14.8 on page 178](#) explains items in the status area.

Table 14.8 Status area—items

Item	Icon	Explanation
Show Classes Pane		Click to display the Classes pane in the Class Browser window.
Hide Classes Pane		Click to hide the Classes pane in the Class Browser window.
Class Declaration		Click to show the declaration of the current class.
Access Filter Display		Displays the access state of the current class.

Using Other Browser Windows

This chapter explains how to work with the Class Hierarchy windows in the CodeWarrior™ IDE. Use Class Hierarchy windows to perform these tasks:

- View hierarchical browser data—the class hierarchy window shows a graphical representation of hierarchical structure. Object-oriented languages, such as C++ and Java, allow hierarchical relationships between classes.
- Analyze inheritance structure—the class hierarchy window shows the inheritance structure of classes. This structure reveals the data-handling capabilities of a particular class.

Read this chapter to learn more about typical tasks for working with Class Hierarchy windows.

This chapter contains these sections:

- [“Multiple-Class Hierarchy Window” on page 179](#)
- [“Single-Class Hierarchy Window” on page 182](#)
- [“Browser Contents window” on page 183](#)
- [“Symbols window” on page 185](#)

Multiple-Class Hierarchy Window

Use the Multi-Class Hierarchy window to visually examine the structure of every class in the browser database. Each class name appears in a box, and lines connect boxes to indicate related classes. The left-most box is the base class, and subclasses appear to the right.

[Figure 15.1 on page 180](#) shows the Multi-Class Hierarchy window. [Table 15.1 on page 180](#) explains the items in the window.

Using Other Browser Windows

Multiple-Class Hierarchy Window

Figure 15.1 Multi-Class Hierarchy window

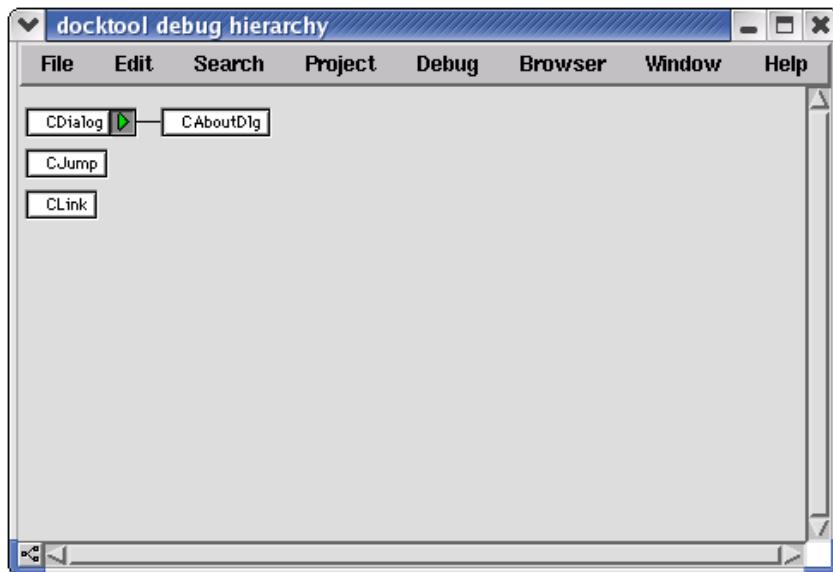


Table 15.1 Multi-class hierarchy window—items

Item	Icon	Explanation
Hierarchy Control		Click to expand or collapse the subclasses displayed for a specific class.
Ancestor menu		Click and hold on class or subclass box to display a menu. Select a class from menu to display that class.
Line button		Click to toggle the lines that connect classes between diagonal and straight lines.

Viewing Browser Data by Inheritance

Use a **Hierarchy** window to view data in graphical form and better understand class relationships. Use the expand and collapse arrows to enlarge or shrink the class views.

1. Activate the browser.
2. Update the browser database by using the **Bring Up To Date**, **Make**, **Run**, or **Debug** command.
3. Open a graphical **Hierarchy** window, as explained in [Table 15.2](#).

Table 15.2 Opening the Hierarchy window

On this host...	Do this...
Windows	Select View > Class Hierarchy
Macintosh	Select Window > Class Hierarchy
Solaris	Select Window > Class Hierarchy
Linux	Select Window > Class Hierarchy Window

Printing Class Hierarchies

To print the contents of a **Class Hierarchy** window, save an image of the window contents, then print the image file from a graphics-processing application.

The IDE saves the image in a graphics-file format based on the host platform, as shown in [Table 15.3](#).

Table 15.3 Graphics-file format for host platforms

Host	Graphics-file Format
Windows	EMF (Enhanced Metafile)
Macintosh	PICT (Picture)
Solaris	PICT (Picture)
Linux	PICT (Picture)

1. Open the **Class Hierarchy** window.
2. Choose **File > Save a Copy As**.

Using Other Browser Windows

Single-Class Hierarchy Window

3. Save the image to a file.
4. Open the image file in an graphics-processing application.
5. Print the image file.

The graphics-processing application prints the image of the class hierarchy.

Changing Line Views in a Hierarchical Window

Use the **Diagonal Line** and **Straight Line** commands to change the appearance of the connecting lines between classes and subclasses in a hierarchical window display.

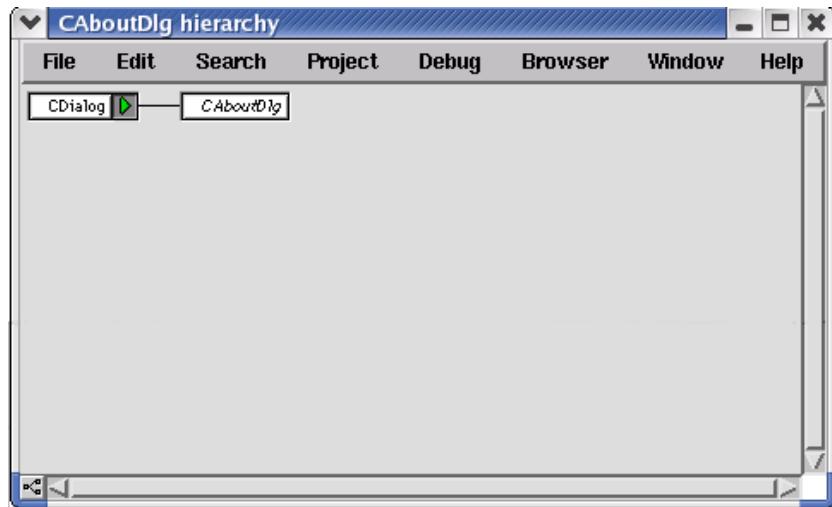
- Click the **Diagonal Line** icon  .
The Hierarchical window display updates to use diagonal lines.
- Click the **Straight Line** icon  .
The Hierarchical window display updates to use straight lines.

Single-Class Hierarchy Window

Use the Single-Class Hierarchy window to examine the structure of a single class in the browser database. The Single-Class Hierarchy window operates identically to the Multi-Class Hierarchy window, but restricts the display to a single class.

The Single-Class Hierarchy window contains the same components as the Multi-Class Hierarchy window.

Figure 15.2 Single-Class Hierarchy window



Opening a Single-Class Hierarchical window

Use one of these methods to open a Single-Class Hierarchical window:



- Click the **Show Single-Class Hierarchy** icon in a Browser toolbar.
 - Use the Browser Contextual menu in one of these windows:
 - New Class Browser window
 - Browser Contents window
 - Multi-Class Hierarchical window
- A Single-Class Hierarchical window appears

Browser Contents window

Use the Browser Contents window to view browser data sorted by category into an alphabetical list. This section explains how to use the Browser Contents window to view browser data.

Using Other Browser Windows

Browser Contents window

[Figure 15.3](#) shows the Browser Contents window. [Table 15.4](#) explains the items in the window.

Figure 15.3 Browser Contents window

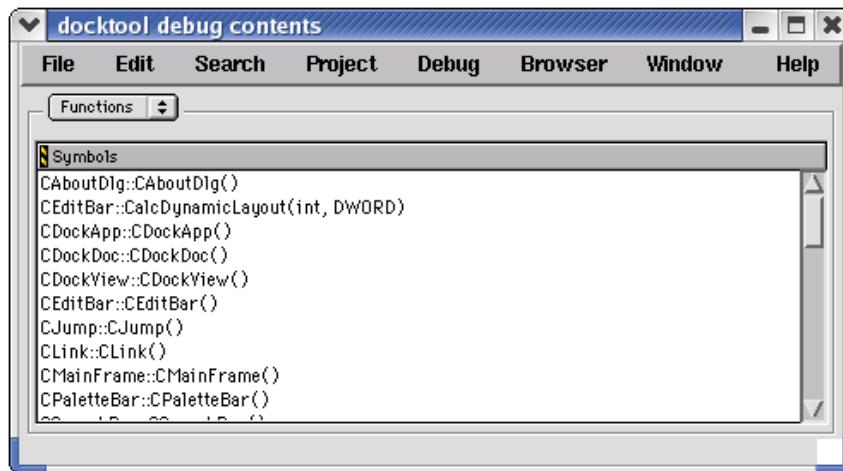


Table 15.4 Browser Contents window—items

Item	Icon	Explanation
Symbols list box	Functions	Select the type of symbol to display in the Symbols list.
Symbols list		Double-click a symbol name to display the source file in a new editor window that defines the symbol.

Viewing Browser Data by Contents

Use the **Browser Contents** window to display symbol information stored in the browser database, listed in alphabetical order. You can choose from these categories:

- classes
- constants

- enumerations
- functions
- global variables
- macros
- function templates
- type definitions

1. Activate the browser.
2. Use the **Bring Up To Date**, **Make**, **Run**, or **Debug** command to update the browser database.
3. Open the Browser Contents window, as explained in [Table 15.5](#).

Table 15.5 Opening the Browser Contents window

On this host...	Do this...
Windows	Select View > Browser Contents
Macintosh	Select Window > Browser Contents
Solaris	Select Window > Browser Contents
Linux	Select Window > Browser Contents

4. Select a category from the **Category** list pop-up.

The symbol information for the selected category appears in alphabetical order in the **Symbols** list.

Symbols window

The Symbols window displays information from project browser databases. With the browser enabled, the IDE generates a browser database for a project during the build process.

The Symbols window displays symbols that have multiple definitions in the browser database. For example, the window displays information about multiple versions of overridden functions in object-oriented code.

Using Other Browser Windows

Symbols window

[Figure 15.4 on page 186](#) shows the Symbols window. [Table 15.5 on page 185](#) explains the items in the window.

Figure 15.4 Symbols window

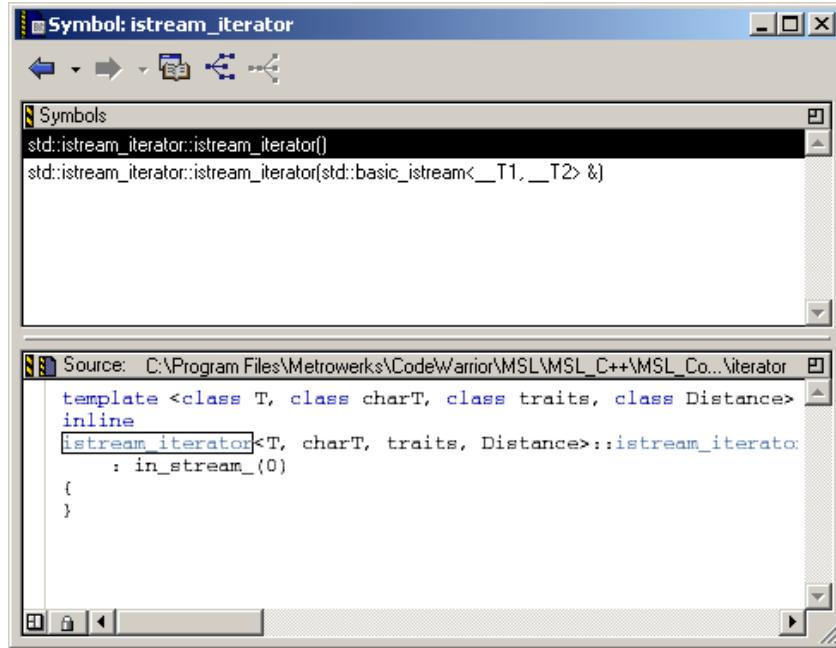


Table 15.6 Symbols window—items

Item	Explanation
Symbols toolbar	Provides one-click access to common browser commands and class-filtering commands.
Symbols pane	Displays a list of all symbols with multiple declarations.
Source pane	Displays the source code for the currently selected item.

Opening the Symbols Window

Use the **Symbols** window to list all implementations, whether overridden or not, of any symbol that has multiple definitions. You can access the Symbols window by using a contextual menu.

1. Open a contextual menu, as explained in [Table 15.7](#).

Table 15.7 Opening the Symbols window

On this host...	Do this...
Windows	Right-click the symbol name.
Macintosh	Control-click the symbol name.
Solaris	Click and hold on the symbol name.
Linux	Click and hold on the symbol name.

2. Select **Find all implementations** from the contextual menu that appears.
3. The Symbols window opens.

Symbols toolbar

Most of the Symbol toolbar items are identical to those in the [Class Browser window](#).

Symbols pane

The **Symbols** pane lists symbols with multiple definitions in the browser database. Select a symbol from the list to view its definition in the **Source** pane.

Source pane

The **Source** pane used in the Symbols window is identical to the one used by the [Class Browser window](#). See “[Source pane](#)” on page [177](#) for more details.

Using Other Browser Windows

Symbols window

Using Browser Wizards

When you create a new class, member function, or data member in the IDE, you use browser wizards. These wizards provide the steps to help you complete the process.

This chapter provides information on these wizards:

- [“The New Class Wizard” on page 189](#)
- [“The New Member Function Wizard” on page 194](#)
- [“The New Data Member Wizard” on page 197](#)

NOTE Most wizard pages contain default settings. To accept all current settings in the wizard, click **Finish** in any screen. The wizard displays a summary of all current settings for the new project. Click **Generate** to accept the current settings and create the new item, or click **Cancel** to return to the wizard to modify settings.

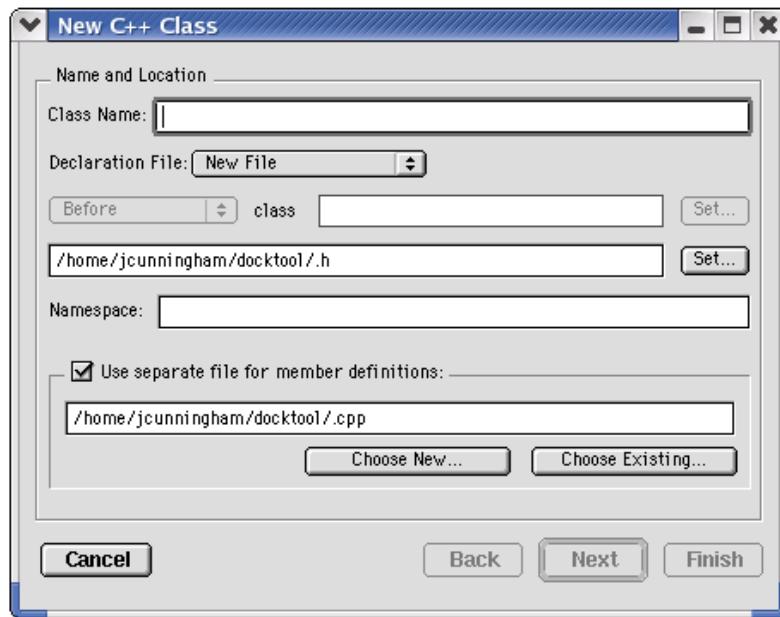
The New Class Wizard

Use the **New Class** wizard to specify the name, declaration, and location for a new class. Click **Finish** in any screen to apply default values to remaining parameters to complete the process. The **New Class** wizard creates the files that define the class.

Using Browser Wizards

The New Class Wizard

Figure 16.1 New Class wizard—Name and Location



Using the New Class Wizard

To use the New Class Wizard, follow these steps:

1. Open the **Class Browser** window, as explained in [Table 16.1](#).

Table 16.1 Opening the Class Browser window

On this host...	Do this...
Windows	Select View > Class Browser
Macintosh	Select Window > Class Browser
Solaris	Select Window > New Class Browser
Linux	Select Window > New Class Browser

2. Select **Browser > New Class**.

NOTE You can also click the New Item icon  in the Class Browser window to create a new class.

3. In the **New C++ Class** wizard, enter **Name and Location** information:
 - a. **Class Name**—Enter a name for the class in this field.
 - b. **Declaration File**—This menu lets you specify whether the file is a **New File**, which is a new declaration file, or **Relative to class**, which is a declaration that depends on an existing file in the project.

If you choose the **New File** option, type in the path where you want to save the file. Alternatively, click **Set** next to the field to choose the path in which to save the file.

If you choose the **Relative to class** option, select **Before** or **After** to establish the order of the new class in relation to existing classes. In the field next to the Before and After drop-down selection, type the name of the class you want to relate to the new class. Alternatively, click **Set** next to this field, type the name of a class in the window that opens, and then click **Select**.

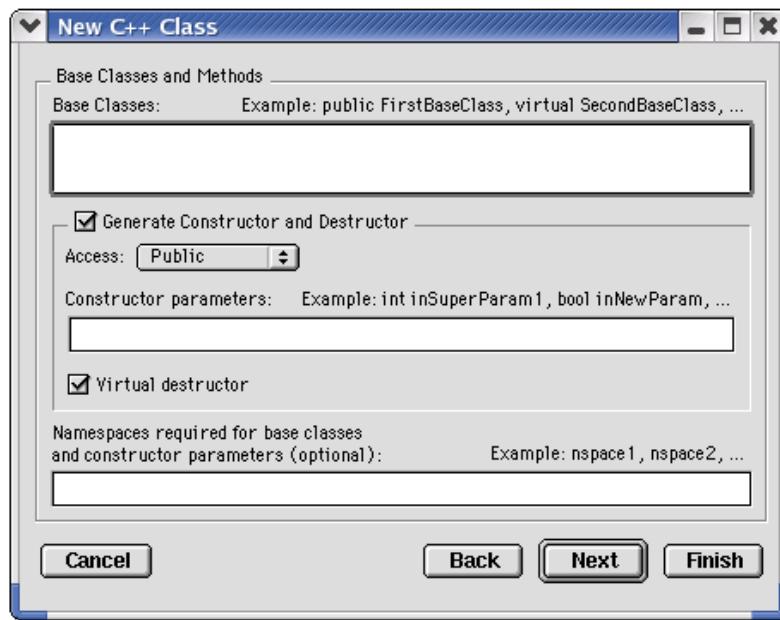
NOTE If you want to use a separate file to define the members of the new class, type the path to the separate file in the field below the **Use separate file for member definitions** checkbox. Alternatively, choose **Existing** to use a standard dialog box to select the file. To create a new, separate file, choose **New** and save the new file to a location on your hard disk.

4. Click **Next**.

Using Browser Wizards

The New Class Wizard

Figure 16.2 New Class wizard—Base Class and Methods



5. Enter **Base Classes and Methods** information.

Enter a list of base classes for the new class:

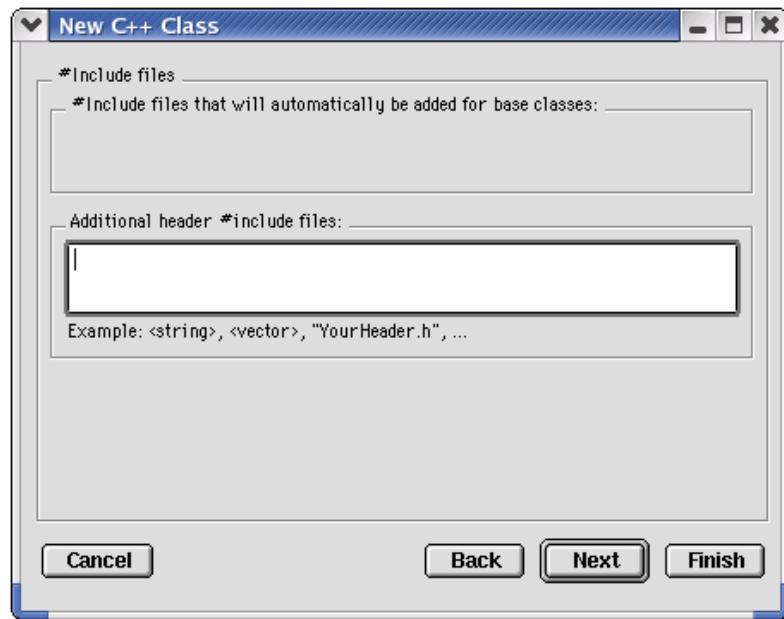
- a. **Access**—From this drop-down menu, choose an access type, **Public**, **Protected**, or **Private**, for the constructor and destructor.
- b. **Constructor parameters**—Enter a list of parameters for the constructor.
- c. **Virtual destructor**—Click this checkbox to create a virtual destructor for the new class.
- d. As an option, you can enter the required namespaces for the base classes and the constructor parameters in the field labeled **Namespaces required for the base classes and constructor parameters**.

Or,

If needed, you can specify the base classes and constructor parameters.

6. Click **Next**.

Figure 16.3 New Class wizard—Include Files



7. Enter **Include Files** information.

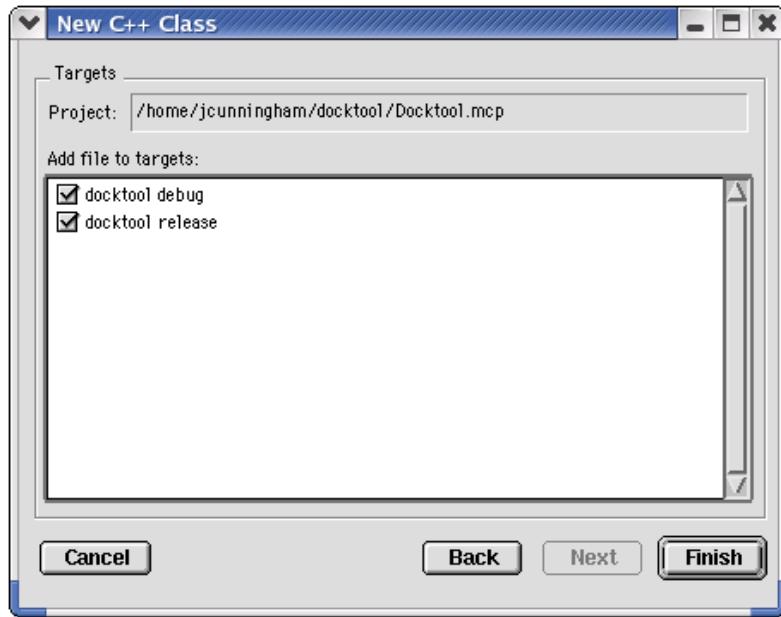
Specify additional header #include files for the new class:

- a. **Include files that will automatically be added for base classes**—This field shows you a list of #include files that the IDE automatically adds to find the base classes.
 - b. **Additional header include files**—Enter a list of other include files for the new class in addition to those in the previous field. Separate each file in the list with a comma.
8. Click **Next**.

Using Browser Wizards

The New Member Function Wizard

Figure 16.4 New Class wizard—Targets



9. Enter **Targets** information:

Select the checkbox next to the build target's name in the list to add the class files to a specific build target.

10. Click **Finish**.

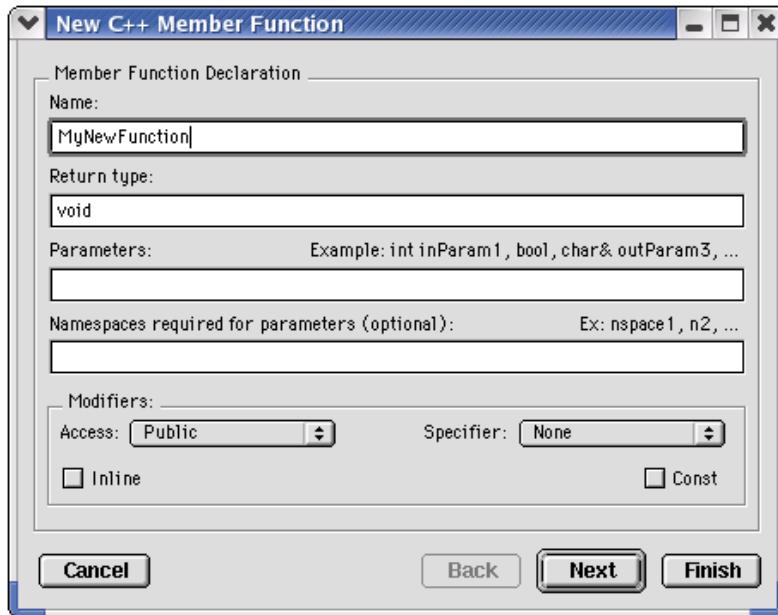
Review the settings summary.

11. Click **Generate**.

The New Member Function Wizard

Use the **New Member Function** wizard to specify the name, return type, and parameters for a new member function. Enter additional information in the wizard fields to refine the function definition.

Figure 16.5 New Member Function wizard



Using the New Member Function Wizard

To use the New Member Function wizard, follow these steps:

1. Open the **Class Browser** window, as explained in [Table 16.2 on page 195](#).

Table 16.2 Opening the Class Browser window

On this host...	Do this...
Windows	Select View > Class Browser
Macintosh	Select Window > Class Browser
Solaris	Select Window > New Class Browser
Linux	Select Window > New Class Browser

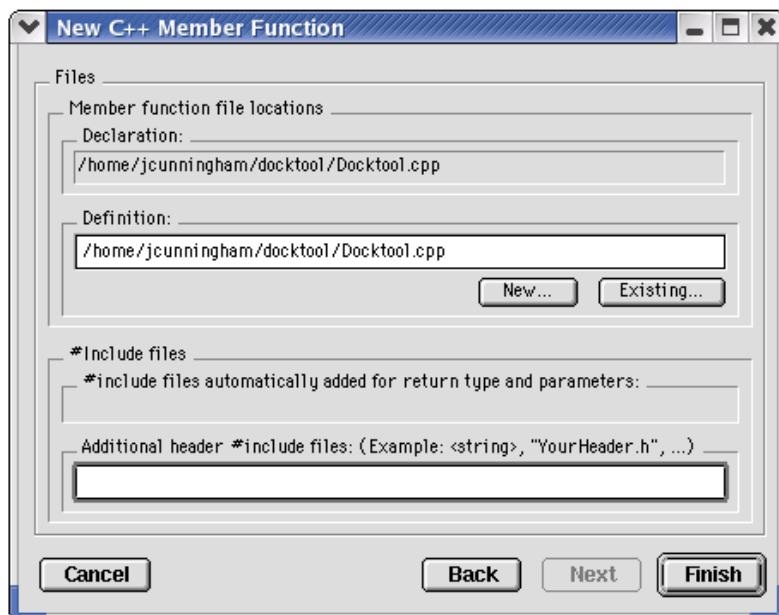
2. Select **Browser > New Member Function**.

Using Browser Wizards

The New Member Function Wizard

3. In the **New C++ Member Function** window, enter the **Member Function Declaration**.
 - a. **Name**—Type a name for the member function.
 - b. **Return Type**—Enter an appropriate function return type.
 - c. **Parameters**—Type a list of function parameters.
 - d. **Namespaces required for parameters (optional)**—Type a list of namespaces required for parameters.
4. Click **Next**.

Figure 16.6 New Member Function wizard—File Locations

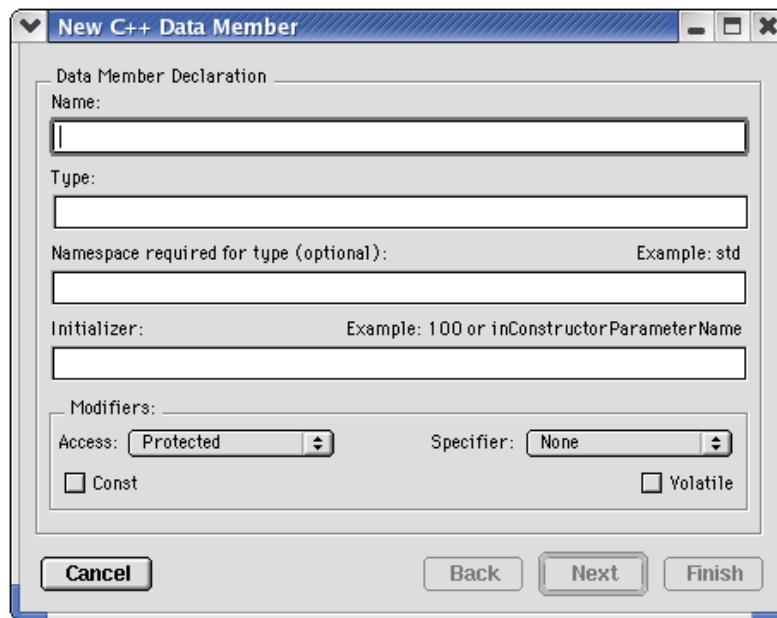


5. Enter **Member Function File Locations** and **Include Files** information.
6. Click **Finish**.
7. Review settings summary, then click **Generate**.

The New Data Member Wizard

Use the **New Data Member** wizard to define the new data-member declaration, and to specify new data member file locations. The wizard offers additional options to further define the function.

Figure 16.7 New Data Member wizard



Using the New Data Member Wizard

To use the New Data Member wizard, follow these steps:

1. Open the **Class Browser** window, as explained in [Table 16.3 on page 198](#).

Using Browser Wizards

The New Data Member Wizard

Table 16.3 Opening the Class Browser window

On this host...	Do this...
Windows	Select View > Class Browser
Macintosh	Select Window > Class Browser
Solaris	Select Window > New Class Browser
Linux	Select Window > New Class Browser

2. Select **Browser > New Data Member**.
3. In the **New C++ Data Member** window, enter the **Name**, **Type**, **Namespaces required for type (optional)**, **Initializer**, and **Modifiers**.
 - a. **Name**—Type a name for the data member in this field.
 - b. **Type**—Enter an appropriate data-member type in this field.
 - c. **Namespaces required for type (optional)**—(Optional) Enter a list of namespaces required for the type in the **Type** field. A sample namespace is `std`.
 - d. **Initializer**—(Optional) Enter an initial value for the data member in this field. Sample initializers are `100` and `inConstructorParameterName`.
 - e. **Modifiers**—Select the access level and type for the new data member.
4. Click **Next**.
5. Specify **Data Member File Locations**.

This section lets you specify file locations associated with the new member functions, including these fields: **Declaration**, **Definition**, **Include file automatically added for member type**, and **Additional header include files**.

- a. **Declaration**—This field shows you the **data member's declaration file location**.
- b. **Definition**—This field is not available in this wizard.
- c. **Include file automatically added for member type**—This field indicates whether an include file will be automatically added for the data-member type.

- d. **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. Example files are <string> and YourHeader.h.
6. Click **Finish**.
7. Review settings summary, then click **Generate**.

Using Browser Wizards

The New Data Member Wizard

Debugger

This section contains these chapters:

- [Working with the Debugger](#)
- [Manipulating Program Execution](#)
- [Working with Variables](#)
- [Working with Memory](#)
- [Working with Debugger Data](#)
- [Working with Hardware Tools](#)

Working with the Debugger

This chapter explains how to work with the debugger in the CodeWarrior™ IDE to control program execution. The main component of the debugger is the Thread window, which shows these items:

- Common debugging controls—step, kill, start, and stop program execution
- Variable information—see the variables in the executing code, their values, and their addresses
- Source code—see the source code under debugger control

This chapter contains these sections:

- [“About the CodeWarrior Debugger” on page 203](#)
- [“About Symbolics Files” on page 204](#)
- [“Thread Window” on page 204](#)
- [“Common Debugging Actions” on page 208](#)
- [“Symbol Hint” on page 211](#)
- [“Contextual Menus” on page 212](#)
- [“Multi-core Debugging” on page 214](#)
- [“External Builds Support”](#)

About the CodeWarrior Debugger

A *debugger* controls program execution and shows the internal operation of a computer program. Use the debugger to find problems while the program executes. Also use the debugger to observe how a program uses memory to complete tasks.

The CodeWarrior debugger provides these levels of control over a computer program:

Working with the Debugger

About Symbolics Files

- Execution of one statement at a time
- Suspension of execution after reaching a specific point in the program
- Suspension of execution after changing a specified memory value

After the debugger suspends program execution, use various windows to perform these tasks:

- View the function-call chain
- Manipulate variable values
- View register values in the computer processor

About Symbolics Files

A *symbolics file* contains debugging information that the IDE generates for a computer program. The debugger uses this information to control program execution. For example, the debugger uses the symbolics file to find the source code that corresponds to the executing object code of the computer program.

Symbolics files contain this information:

- Routine names
- Variables names
- Variable locations in source code
- Variable locations in object code

The IDE supports several types of symbolics files. Some programs generate separate symbolic files, while others do not. For example, when you use CodeView on Windows, the IDE places the symbolics file inside the generated binary file.

Thread Window

The debugger suspends execution of processes in a computer program. The Thread window displays information about a suspended process during a debug session.

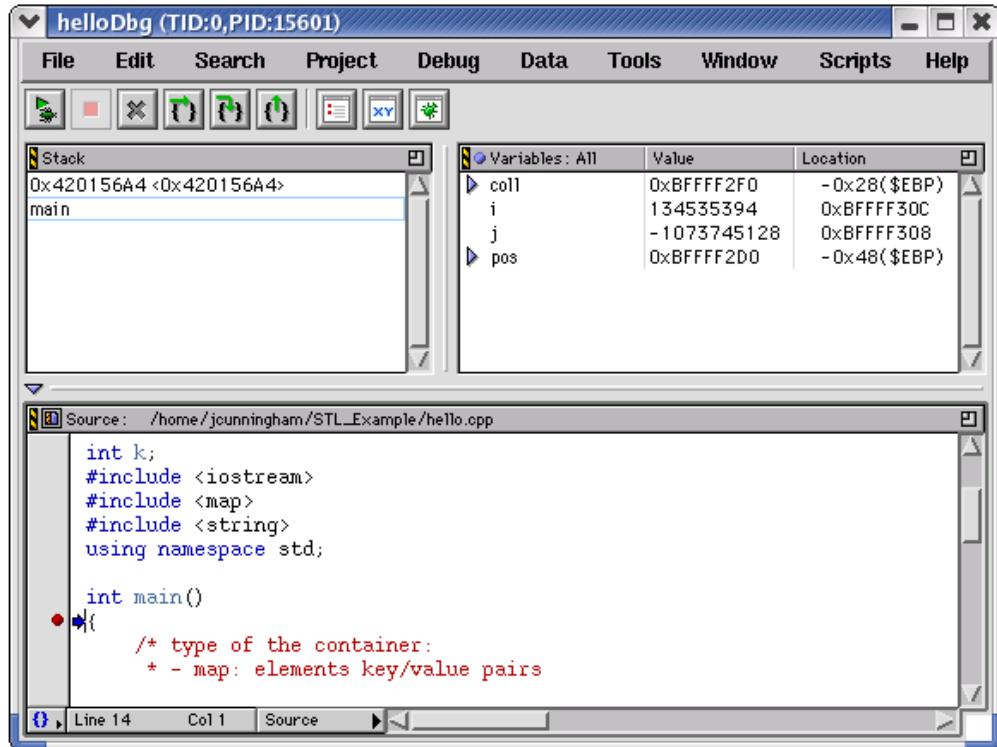
Use the Thread window to perform these tasks:

- View the call chain for a routine
- View routine variables, both local and global

- View a routine in terms of its source code, assembly code, or a mix of both types of code

[Figure 17.1](#) shows the Thread window. [Table 17.1](#) explains the items in the window.

Figure 17.1 Thread window



Working with the Debugger

Thread Window

Table 17.1 Thread window—items

Item	Icon	Explanation
Debug / Run / Resume button		Click to perform these tasks: <ul style="list-style-type: none">Continue execution up to the next breakpoint, watchpoint, or eventpointRun the program until it exitsContinue execution of a currently stopped program
Stop button		Click to stop (pause) program execution.
Kill button		Click to terminate program execution and close the Thread window.
Step Over button		Click to execute the current line, including any routines, and proceed to the next statement.
Step Into button		Click to execute the current line, following execution inside a routine.
Step Out button		Click to continue execution to the end of the current routine, then follow execution to the routine's caller.
Breakpoints button		Click to open the Breakpoints window.
Expressions button		Click to open the Expressions window.
Symbolics button		Click to open the Symbolics window.
Pane Expand box		Click to enlarge the pane to fill the window.
Pane Collapse box		Click to reduce an expanded pane to its original size.

Table 17.1 Thread window—items (*continued*)

Item	Icon	Explanation
Pane resize bar		Drag to resize the panes on either side of the bar.
Stack pane		Shows the current routine calling chain, with the most current routine name at the bottom
Variables pane		Shows local and global variables that the current routine uses.
Variables Pane Listing button		Click this icon to switch among these display states: <ul style="list-style-type: none"> • All—show all local and global variables in the code • Auto—show only the local variables of the routine pointed to by the current-statement arrow • None—show no variables. Use this display state to improve stepping performance for slow remote connections
Source pane disclosure triangle		Click to show or hide the Source pane.
Source pane		Shows the executing source code. This pane operates the same way as an editor window, however, you cannot edit the contents of the pane or use pane-splitter controls.
Source File button		Click to edit the contents of the Source pane in an editor window.
Current-statement arrow		Points to statement that debugger will execute next.
Dash	—	Appears to left of each line at which you can set a breakpoint or eventpoint. Click the dash to set a breakpoint on that line.

Working with the Debugger

Common Debugging Actions

Table 17.1 Thread window—items (*continued*)

Item	Icon	Explanation
Functions list box		Click to show a list of functions declared in the file. Select a function to highlight it in the Source pane.
Line and Column button		Shows the current line and column number of the text-insertion cursor. Click to specify a line to show in the Source pane.
Source list box		Click to specify how to display source code in the Source pane: <ul style="list-style-type: none">• Source—programming-language statements appear exclusively in the pane• Assembler—assembly-language instructions appear exclusively in the pane• Mixed—each programming-language statement shows its corresponding assembly-language instructions

Common Debugging Actions

This section explains how to perform common debugging actions that correct source-code errors, control program execution, and observe memory behavior:

- Start the debugger
- Step into, out of, or over routines
- Stop, resume, or kill program execution
- Run the program
- Restart the debugger

Starting the Debugger

Use the **Debug** command to begin a debugging session. The debugger takes control of program execution, starting at the main entry point of the program.



Select **Project > Debug** or click the Debug button (shown at left) to start the debugger.

After you start the debugging session, the IDE opens a new Thread window.

NOTE Some projects require additional configuration before the debugging session can begin. The IDE might prompt you for permission to perform this configuration automatically.

Stepping Into a Routine

Use the **Step Into** command to execute one source-code statement at a time and follow execution into a routine call.

 Select **Debug > Step Into** or click the Step Into button to step into a routine.

After the debugger executes the source-code statement, the current-statement arrow moves to the next statement determined by these rules:

- If the executed statement did not call a routine, the current-statement arrow moves to the next statement in the source code.
- If the executed statement called a routine, the current-statement arrow moves to the first statement in the called routine.
- If the executed statement is the last statement in a called routine, the current-statement arrow moves to the statement that follows the calling routine.

Stepping Out of a Routine

Use the **Step Out** command to execute the rest of the current routine and stop program execution after the routine returns to its caller. This command causes execution to return up the calling chain.

 Select **Debug > Step Out** or click the Step Out button to step out of a routine.

The current routine executes and returns to its caller, then program execution stops.

Working with the Debugger

Common Debugging Actions

Stepping Over a Routine

Use the **Step Over** command to execute the current statement and advance to the next statement in the source code. If the current statement is a routine call, program execution continues until reaching one of these points:

- the end of the called routine
- a breakpoint
- a watchpoint
- an eventpoint that stops execution

 Select **Debug > Step Over** or click the Step Over button to step over a routine.

The current statement or routine executes, then program execution stops.

Stopping Program Execution

Use the **Break** or **Stop** command to suspend program execution during a debugging session.

 Select **Debug > Break**, **Debug > Stop**, or click the Stop button to stop program execution.

The operating system surrenders control to the debugger, which stops program execution.

Resuming Program Execution

Use the **Resume** command to continue executing a suspended debugging session. If the debugging session is already active, use this command to switch view from the Thread window to the executing program.

 Select **Project > Resume** or click the Debug button to resume program execution.

The suspended session resumes, or the view changes to the running program.

NOTE The Resume command appears only for those platforms that support it. If your platform does not support this command, you must stop the current debugging session and start a new session.

Killing Program Execution

Use the **Kill** command to completely terminate program execution and end the debugging session. This behavior differs from stopping a program, as stopping temporarily suspends execution.

-  Select **Debug > Kill** or click the Kill button to kill program execution.

The debugger terminates program execution and ends the debugging session.

Running a Program

Use the **Run** command to execute a program normally, without debugger control.

-  Select **Project > Run** or click the Run button to begin program execution.

The debugger does not control program execution as the program runs.

Restarting the Debugger

Use the **Restart** command after stopping program execution. The debugger goes back to the beginning of the program and begins execution again. This behavior is equivalent to killing execution, then starting a new debugging session.

Select **Debug > Restart** to restart the debugger.

Symbol Hint

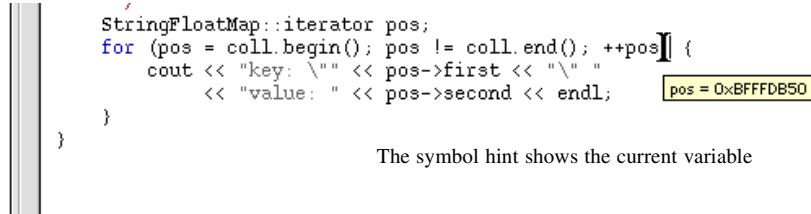
The symbol hint shows information about variable values. This information appears automatically while the debugger is active.

Select the [**Show variable values in source code**](#) option in the [**Display Settings**](#) preference panel to use the symbol hint.

Working with the Debugger

Contextual Menus

Figure 17.2 Symbol hint



Toggling the Symbol Hint

Turn on the symbol hint to view information about program variables in source views.

1. Click **Edit > Preferences**.

The IDE Preferences window appears.

2. Select **Display Settings** in the IDE Preference Panels list.

3. Check or clear the **Show variable values in source code** checkbox.

Check the checkbox to use the symbol hint. Clear the checkbox to stop using the symbol hint.

4. Click **Apply** or **Save** to confirm your changes to the preference panel.

5. Close the IDE Preferences window.

Using the Symbol Hint

During a debugging session, use the symbol hint to view information about program variables.

To use the symbol hint, rest the cursor over a variable in a source view. After a brief pause, the symbol hint appears and shows the current variable value.

Contextual Menus

The contextual menu provides a shortcut to frequently used menu commands.

The available menu commands change, based on the context of the selected item.

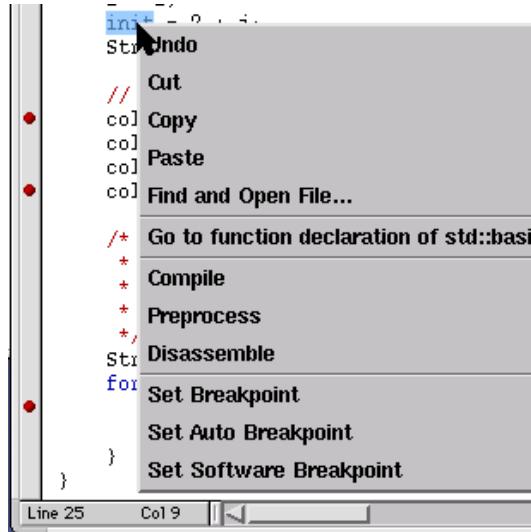
Sample uses of the contextual menu for debugging tasks include:

- changing the format of variables displayed in variable panes
- manipulating breakpoints and the program counter in source panes
- viewing memory in separate windows

TIP Experiment using the contextual menu in various IDE windows to discover additional features.

[Figure 17.3](#) shows a sample contextual menu in a source view.

Figure 17.3 Contextual menus



Using Contextual Menus

Use contextual menus to apply context-specific commands to selected items. Right-click, Control-click, or click and hold on an item to open a contextual menu for that item. The contextual menu appears, displaying menu commands applicable to the selected item.

Multi-core Debugging

The IDE allows simultaneous debugging of multiple projects. This feature provides multi-core debugging capability for some embedded processors. By configuring each project to operate on a single core, the IDE can debug multiple cores by debugging multiple projects.

Configuring multi-core debugging involves these tasks:

- configuring specific target settings for each project
- for some cores, specifying a configuration file for initializing multi-core debugging

For more information, see the *Targeting* documentation.

Data Viewer Plug-ins

Data Viewers are plug-ins that include an user interface to show a custom view of data. These plug-ins are often platform specific. Data editors are data viewers that also let you modify and write data.

The IDE will keep a registry of plug-ins that can view particular types. The plug-ins will register themselves with the IDE and indicate which formats and platforms they support. When a variable or memory address is selected, you can choose the plugin from the Data menu.

A Data Viewer plugin may also designed without a custom user interface. This type of viewer would override the built in debugger methods of showing a variable value as text and parsing an edited value back into variable data.

External Builds Support

The IDE will perform the following tasks on external makefiles:

- Build
- Debug
- File Management in Project Manager window
- Source Browsing
- Error Lookup

The IDE can build an external makefile and debug its output. A linker plugin will enable the IDE to manage a command line and targets associated with a makefile. The command line is executed when a build step is initiated. The linker plugin will also supply the executable to use for debugging.

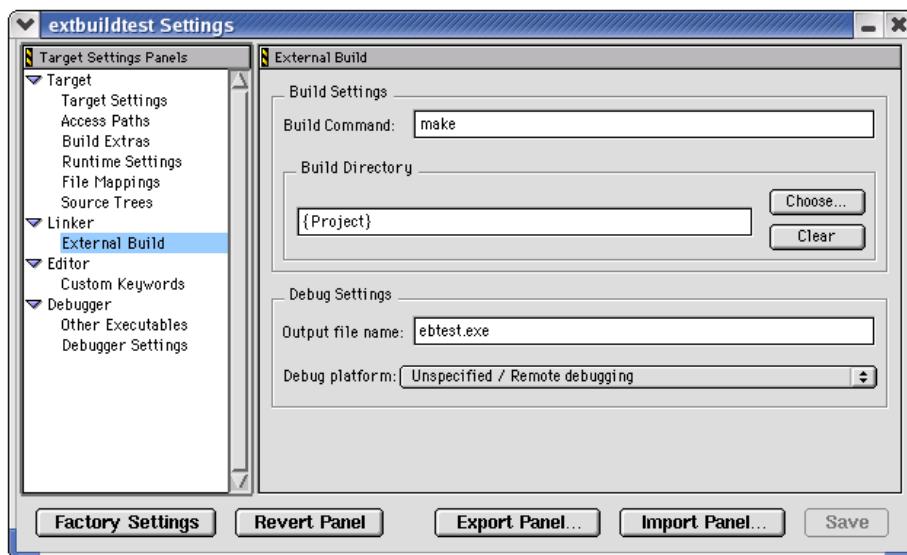
The linker plugin will provide a preference panel named External Build that is used to configure a target. The preference panel provides text fields for you to configure the command line for the target (which will enable building), specify the working directory and the output file used to launch a debugging session, and the debug platform.

The linker plugin is generic so that it can be used regardless of the target CPU and OS. The list of available debugger preference panels will be updated by the IDE when you select the debug platform.

Working with the Debugger

External Builds Support

Figure 17.4 External Build Target panel



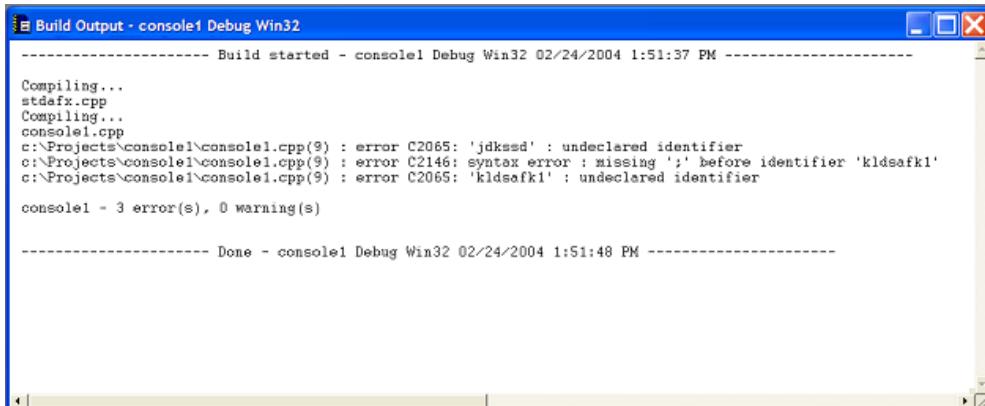
The External Build Target preference panel will collect the following information:

- Build command line to be executed in the build step
The command line will be sent to the OS shell and will contain all parameters and/or switches that are necessary for proper building of the make file.
- Build directory in which command line will be executed.
- Output file name - Executable to be launched in the debug step. The file will be relative to the output directory specified in the Target Settings preference panel.
- Debug platform - The debugger platform represents the combination of OS and CPU that your build is targeting. “Unspecified/Remote debugging” is the default, which indicates you have not specified a debug platform. In most cases, not specifying a platform will result in not being able to debug. However, some platforms may allow debugging if no additional debugger preference panel is used. If only one platform entry exists with the “Unspecified” option, then it will become the default entry.

After the IDE converts the makefile into a CodeWarrior project, source files can be added in the project manager window. Files that appear in the project manager will be parsed by the language parser and will supply Source Browsing information, such as code completion.

When a build step is initiated, the linker plugin will gather output after the command line begins executing. Output is directed to the IDE and displayed in a read-only Build Output Window. A build output window is displayed for each target. The build output window can be displayed manually by selecting the menu command **View > Build Output** (Windows) or **Window > Build Output** (Linux/Solaris/Mac). This command is enabled for targets that use the external build linker.

Figure 17.5 Build Output Window



If multiple build steps are performed on the same target, the output from each build step will be appended to the build output window. Each block of output will be separated by “----Build started----” and “----Done----” tags.

The build output window will allow users to navigate directly to the location of a compilation error. Double-click a line that contains an error message or press Enter when the cursor is in the line. If the IDE determines that a valid error message exists on the selected line, the source file is opened to the line on which the error occurred.

Click the right mouse button in the build output window to display a context menu.

- The Copy command will copy selected text to the clipboard. If no text is selected then the line that contains the cursor will be copied.

Working with the Debugger

External Builds Support

- The Clear All command will clear contents of output window.
- The Go To Error command will navigate to the error location. This is identical to double-clicking.

External Build Wizard

A wizard prompts you for information used to create projects based on external make files. The wizard collects data about the make file and creates a CodeWarrior project with a single target. The target is then configured to build the user-specified make file.

The wizard can be launched by selecting **File > New...** and selecting External Build Wizard. The New... dialog will collect the name and location of the project before launching the wizard.

Figure 17.6 External Build Wizard - page 1 of 2

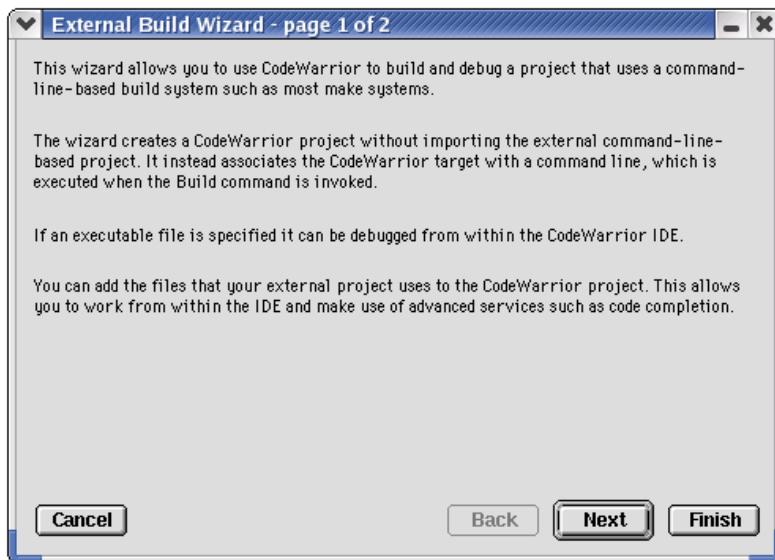
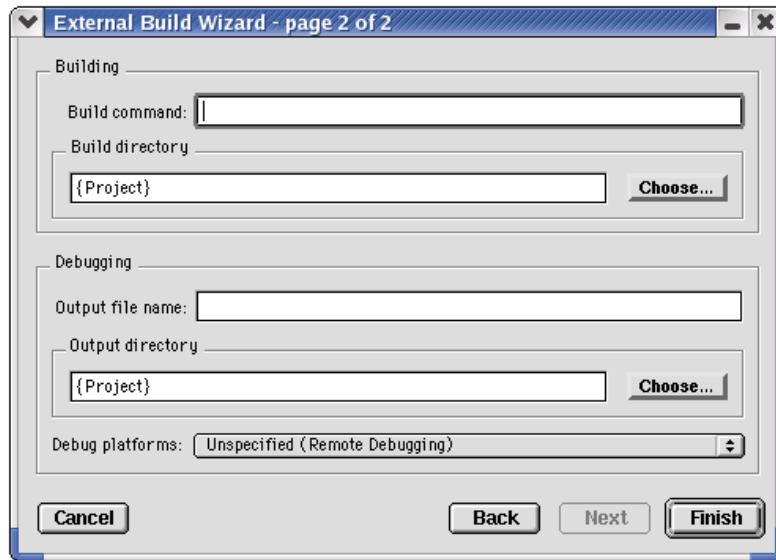


Figure 17.7 External Build Wizard - page 2 of 2



If the Output filename entry is blank, you can still finish the wizard, but no debugging can be done until you enter an output file in the External Build target panel. You can also finish the wizard if the Debug platform choice is set to "Unspecified". Although, no debugging can be performed until you specify a debug platform in the External Build target panel.

Completing the wizard will generate a new CodeWarrior IDE project and configure it for use with the external make file. The wizard will automate these tasks:

- Create project with single target named "External makefile"
- Set Linker to external make file linker
- Define settings in External Build target panel based on data collected from wizard

Working with the Debugger

External Builds Support

Manipulating Program Execution

This chapter explains how to use breakpoints, watchpoints, and eventpoints to manipulate execution of your program in the CodeWarrior™ IDE:

- Breakpoints—halt program execution on a line of source code that you specify. You can set a breakpoint that always halts program execution, or you can set a breakpoint that halts program execution if a condition that you specify is true.
- Eventpoints—perform a task during program execution on a line of source code that you specify. Eventpoints can play sounds, run scripts, log data, and perform other operations.
- Watchpoints—halt program execution after a location in memory is accessed
- Special breakpoints—these internal breakpoints halt program execution in special cases, such as halting program execution at the `main()` function or for a C++ exception.

After you set these items in your source code, you start a debugging session to use them. As program execution arrives at each of these items, the debugger can halt execution, perform a task, or update data.

This chapter contains these sections:

- [“Breakpoints” on page 222](#)
- [“Eventpoints” on page 233](#)
- [“Watchpoints” on page 244](#)
- [“Special Breakpoints” on page 248](#)

Manipulating Program Execution

Breakpoints

Breakpoints

You use *breakpoints* to halt program execution on a specific line of source code. After you set a breakpoint at a key point in the program, you can halt its execution, examine its current state, and check register and variable values. You can also change values and alter the flow of normal program execution. Setting breakpoints helps you debug your program and verify its efficiency.

You can use these types of breakpoints:

- regular breakpoints—halt program execution
- conditional breakpoints—halt program execution after meeting a condition that you specify
- temporary breakpoints—halt program execution and then remove the breakpoint that caused the halt

You can also create breakpoint templates to simplify the process of setting complex breakpoints. A *breakpoint template* has all the properties of a breakpoint, except for its location in source code. After you define a breakpoint template, you can have the debugger use the template as the basis for each breakpoint you set in your source code.

Breakpoints have *enabled* and *disabled* states. [Table 18.1](#) explains these states.

Table 18.1 Breakpoints—states

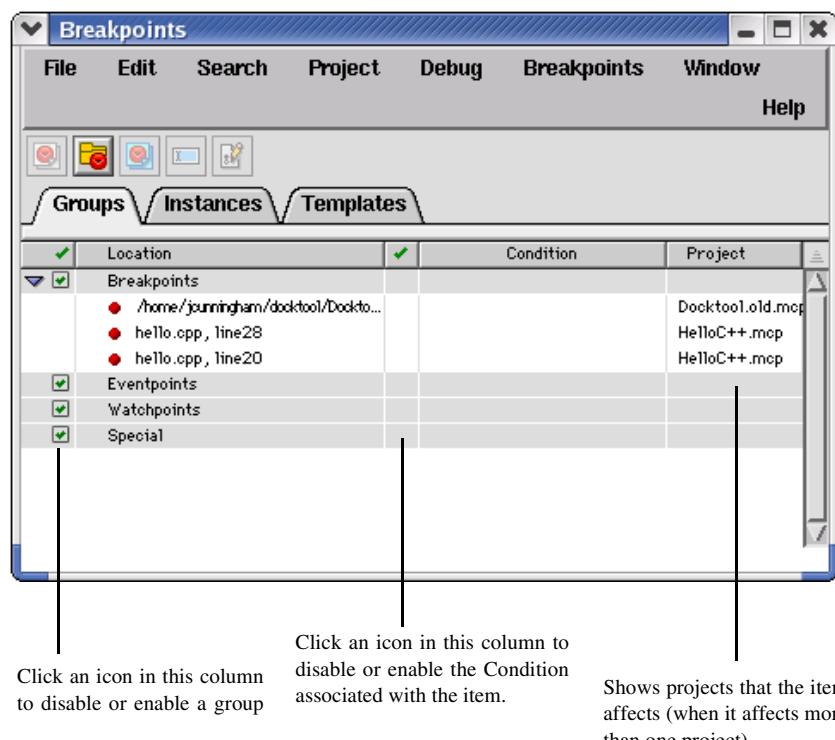
State	Icon	Explanation
Enabled		Indicates that the breakpoint is currently enabled. The debugger halts program execution at an enabled breakpoint. Click the icon to disable the breakpoint.
Disabled		Indicates that the breakpoint is currently disabled. The debugger does not halt program execution at a disabled breakpoint. Click the icon to enable the breakpoint.

Breakpoints Window

Use the **Breakpoints** window to set breakpoints. [Figure 18.1 on page 223](#) shows this window. [Table 18.2 on page 224](#) explains items in the window.

You can change the sort order of items in the Breakpoints window by clicking the column titles. Click the sort order button next to the rightmost column title to toggle between ascending and descending sort order.

Figure 18.1 Breakpoints window



Manipulating Program Execution

Breakpoints

Table 18.2 Breakpoints window—items

Item	Icon	Explanation
Create Breakpoint Template		Click to create a new breakpoint template in the Templates page. You can create complex breakpoints based on properties you define in the breakpoint template.
Create Breakpoint Group		Click to create a new group in the Groups page of the Breakpoints window. Clicking this button is equivalent to clicking Breakpoints > Create Breakpoint Group .
Set Default Breakpoint Template		Click to designate the selected item in the Templates page as the default breakpoint template. The debugger uses this template as the basis for creating new breakpoints. Clicking this button is equivalent to clicking Breakpoints > Set Default Breakpoint Template with the Breakpoints window frontmost.
Rename Breakpoint		Click to rename the selected item in the Breakpoints window. Clicking this button is equivalent to clicking Breakpoints > Rename Breakpoint with the Breakpoints window frontmost.
Breakpoint Properties		Click to view more information about the selected breakpoint, such as name, associated condition, and number of hits during execution. Clicking this button is equivalent to clicking Breakpoints > Breakpoint Properties with the Breakpoints window frontmost.
Groups tab		Click to display the Groups page. This page lets you work with breakpoints, eventpoints, watchpoints, and internal breakpoints.

Table 18.2 Breakpoints window—items (*continued*)

Item	Icon	Explanation
Instances tab		Click to display the Instances page. This page lets you set breakpoints, eventpoints, and watchpoints on a per-thread or per-process basis.
Templates tab		Click to display the Templates page. This page lets you define breakpoint templates and specify a default breakpoint template.
Active		These items affect program execution. Click the icon to make inactive.
Inactive		These items do not affect program execution. Click the icon to make active.

Opening the Breakpoints Window

Use the **Breakpoints** window to view a list of breakpoints currently set in your projects.

To open the Breakpoints window, click **View > Breakpoints** or **Window > Breakpoints Window**.

NOTE Double-click a breakpoint in the Breakpoints window to display its associated source-code line in an editor window.

Saving the Contents of the Breakpoints Window

You can save the contents of the Breakpoints window. This feature is useful for saving sets of breakpoint data, then later re-opening those sets.

To save contents of the Breakpoints window, click **File > Save** or **File > Save As**. Clicking **File > Save As** lets you specify the name and path to save the file that stores the contents.

Working with Breakpoints

This section explains how to work with breakpoints in your source code and in the **Breakpoints** window.

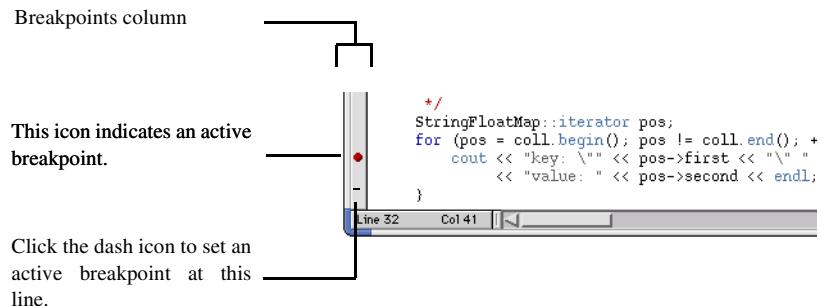
Setting a Breakpoint

Use the **Set Breakpoint** command to set a breakpoint. A regular breakpoint suspends program execution. The debugger does not execute the line of source code that contains the regular breakpoint.

The default breakpoint template in the **Templates** page of the **Breakpoints** window determines the type of breakpoint that the debugger sets. The **Auto Breakpoint** default breakpoint template defines a breakpoint that halts program execution at a line of source code. You can change the default breakpoint template to a breakpoint template that you specify.

[Figure 18.2](#) shows some source code and the Breakpoints column to the left of the source code. Breakpoint icons appear in this column.

Figure 18.2 Setting Breakpoints



To set a breakpoint at a line of source code, click the Breakpoints column next to that line. The active breakpoint icon appears in the column. After you debug the project, the debugger halts program execution at the line that has the active breakpoint icon.

TIP You can also set a breakpoint for selected results in the **Search Results** window and for selected items in the **Symbolics** window.

If you debug your project first, dash icons appear in the Breakpoints column next to source-code lines at which you can set breakpoints. Click a dash icon to set a breakpoint at that line. The dash changes to an active breakpoint icon.

NOTE Setting a breakpoint in a file affects execution of all build targets that include that file.

Viewing Breakpoint Properties

After you set a breakpoint, you can view and modify its properties. [Table 18.3](#) explains breakpoint properties.

To view properties for a breakpoint, select its name in the **Breakpoints** window and click **Breakpoints > Breakpoint Properties**.

Table 18.3 Breakpoint properties

Property	Explanation
Breakpoint Type	The type of item, such as Auto Breakpoint.
Serial number	The non-persistent serial number that uniquely identifies the item in the IDE. Use this number to identify the item in scripting languages. This number is not the same number that the debugger plug-ins use to identify the item.
Condition	The conditional expression associated with the item. This conditional expression must evaluate to true in order for the item to perform its specified action.
Hit Count	Displays the number of times that program execution arrived at the breakpoint before the program stopped.
File-Info	The path to the file that contains the item.
Name	The name of the item, which appears in the Breakpoints window. The IDE creates a default name based on the item properties, but you can change this name to a more meaningful one. Use this name to identify the item in scripting languages.
Original Process	The persistent identifier for the active process at the time you set the item. If information about the active process was not available at the time you set the item, this identifier shows the process at the time the item affected program execution.

Manipulating Program Execution

Breakpoints

Table 18.3 Breakpoint properties (*continued*)

Property	Explanation
Original-Target	The path to the build target that contains the item.
Times Hit	The number of times that this item affected program execution.
Times Left	The number of times remaining for this item to affect program execution.
Thread	The thread in which the item takes effect.
Hardware	The hardware on which to use the item. For example, set this property to Prefer Hardware to specify that the breakpoint is a hardware breakpoint.

Disabling a Breakpoint

Disable a breakpoint to prevent it from affecting program execution. The disabled breakpoint remains at the source-code line at which you set it, so that you can enable it later. Disabling the breakpoint is easier than clearing it and re-creating it from scratch.

To disable a breakpoint, select its name in the **Breakpoints** window, or click the cursor on the source-code line that contains the breakpoint, and click **Debug > Disable Breakpoint**.

- The enabled breakpoint icon changes to a disabled breakpoint icon (shown at left). The disabled breakpoint icon indicates that the breakpoint does not halt program execution.

Enabling a Breakpoint

Enable a breakpoint to have it halt program execution. Enabling a breakpoint that you previously disabled is easier than clearing it and re-creating it from scratch.

To enable a breakpoint, select its name in the **Breakpoints** window, or click the cursor on the source-code line that contains the breakpoint, and click **Debug > Enable Breakpoint**.

- ➊ The disabled breakpoint icon changes to an enabled breakpoint icon (shown at left). The enabled breakpoint icon indicates that the breakpoint halts program execution.
-

Clearing a Breakpoint

Use the **Clear Breakpoint** command to clear a breakpoint.

To clear a breakpoint in source code, click the cursor on the source-code line that contains the breakpoint and click **Debug > Clear Breakpoint**. You can also click the active breakpoint icon in the **Breakpoints** column to clear the breakpoint.

To clear a breakpoint in the **Breakpoints** window, select its name from the list in the **Groups**, **Instances**, or **Templates** pages and press Delete.

Clearing All Breakpoints

Use the **Clear All Breakpoints** command to clear all breakpoints from your projects.

To clear all breakpoints, click **Debug > Clear All Breakpoints**. The **Breakpoints** window reflects your changes.

Setting a Temporary Breakpoint

Use the **Temporary Breakpoint** command to set temporary breakpoints. Unlike a regular breakpoint that halts execution each time you debug a project, a *temporary breakpoint* halts execution only once. The debugger removes the temporary breakpoint after halting program execution. Setting a temporary breakpoint is equivalent to using the **Run To Cursor** command.

To set a temporary breakpoint at a line of source code, Alt-click or Option-click the dash icon next to that line. The dash icon changes to an active breakpoint icon. After you debug the project, the debugger halts program execution at the line that has the active breakpoint icon. After execution halts, the active breakpoint icon reverts to a dash icon.

Setting a Conditional Breakpoint

Use the **Condition** column of the **Breakpoints** window to set a conditional breakpoint. A *conditional breakpoint* has an associated conditional expression. The debugger evaluates the expression to determine whether to halt program execution at that breakpoint.

A conditional breakpoint behaves in two different ways:

- If the expression evaluates to true (a non-zero value), the debugger halts program execution.
- If the expression evaluates to false (a zero value), program execution continues without stopping.

Follow these steps to set a conditional breakpoint:

1. Set a breakpoint that you want to associate with a conditional expression.
2. Depending upon operating system, click **View > Breakpoints** or **Window > Breakpoints Window**.
3. In the **Groups or Instances** pages of the Breakpoints window, find the breakpoint that you want to associate with a conditional expression.
4. In the **Condition** column adjacent to a specific breakpoint, double-click to display a text box in the blank area.
5. Enter an expression in the text box.

During subsequent debugging sessions, the debugger evaluates the expression to determine whether to halt program execution at the conditional breakpoint.

NOTE Alternatively, drag-and-drop an expression from a source view or from the Expression window into the Breakpoints window.

To signal a breakpoint to happen after the nth execution of the instruction, you can enter the keywords "Hit Count" in the condition text box. For example, enter Hit Count > 5 for the breakpoint action to occur after the instruction has executed five times.

Setting a thread-specific conditional breakpoint

CodeWarrior's debugger supports thread-specific breakpoints. Depending on what the protocol supports, there are several different ways it's supported. You can setup a special condition on a breakpoint. Enter `mwThreadID == threadID`, where `mwThreadID` is a keyword recognized by the core debugger and `threadID` is the number that represents the ID of the thread that you want to stop.

Working with Breakpoint Templates

This section explains how to define breakpoint templates, specify a default template, and delete templates.

A *breakpoint template* defines all properties of a breakpoint except for its location in source code. For example, you can define a breakpoint template that stops execution only 10 times, and only if an associated conditional expression evaluates to false.

The *default breakpoint template* is the breakpoint template that the debugger uses as the basis for new breakpoints that you set. For example, if you define a breakpoint template named **Thread Break**, you can specify it as the default breakpoint template. After you do this, the **Thread Break** template properties apply to all new breakpoints that you set in your source code.

The initial default breakpoint template is **Auto Breakpoint**, which defines the regular breakpoint that halts program execution at a line of source code. You can change the default breakpoint template from **Auto Breakpoint** to any of your breakpoint templates. You can also change the default breakpoint template back to

Auto Breakpoint.

Creating a Breakpoint Template

Use the **Templates** page of the **Breakpoints** window to define breakpoint templates. You define a breakpoint template by using an existing breakpoint as a starting point.

To define a breakpoint template, follow these steps:

1. Set a breakpoint in your source code.

Manipulating Program Execution

Breakpoints

2. Click **View > Breakpoints** or **Window > Breakpoints Window**.
The **Breakpoints** window appears.
3. Click the **Groups** tab.
4. Select the name of the breakpoint that you just set.
The debugger gives the breakpoint a default name that includes the name of the file in which you set the breakpoint and the line at which you set the breakpoint.
5. Click the **Create Breakpoint Template** button in the toolbar of the Breakpoints window.
6. Click the **Templates** tab of the Breakpoints window.

The new breakpoint template appears in this page with the name **New Template**.

You can rename the breakpoint template by selecting it and clicking **Breakpoints > Rename Breakpoint**, or clicking the **Rename Breakpoint** button in the Breakpoints window toolbar.

NOTE You cannot rename the **Auto Breakpoint** template.

Deleting a Breakpoint Template

Use the **Templates** page of the **Breakpoints** window to delete breakpoint templates that you no longer need.

To delete a breakpoint template, follow these steps:

1. Click **View > Breakpoints** or **Window > Breakpoints Window**.
The **Breakpoints** window appears.
2. Click the **Templates** tab of the Breakpoints window.
3. Select the breakpoint template that you want to delete.
4. Click **Edit > Delete** or **Edit > Clear**.

NOTE You cannot delete the **Auto Breakpoint** template, because it defines the regular breakpoint.

Specifying the Default Breakpoint Template

Use the **Templates** page of the **Breakpoints** window to specify the default breakpoint template. The debugger uses this template as the basis for creating new breakpoints in your source code.

The initial default breakpoint template is **Auto Breakpoint**, which defines the regular breakpoint. You can specify any one of your breakpoint templates, or **Auto Breakpoint**, as the default breakpoint template.

To specify the default breakpoint template, follow these steps:

1. Click **View > Breakpoints** or **Window > Breakpoints Window**.

The **Breakpoints** window appears.

2. Click the **Templates** tab of the Breakpoints window.
3. Select the breakpoint template that you want to specify as the default breakpoint template.
4. Click **Breakpoints > Set Default Breakpoint Template** or click the Set Default Breakpoint Template icon in the Breakpoints window toolbar.

The debugger now uses the breakpoint template that you specified as the basis for creating new breakpoints in your source code.

Eventpoints

You use *eventpoints* to perform a task when program execution arrives at a specific line of source code or when an associated conditional expression evaluates to true. You can set an eventpoint that performs a task such as running a script, playing a sound, or collecting trace data. An eventpoint is equivalent to a breakpoint that performs a task other than halting program execution.

You can use several kinds of eventpoints. The Breakpoints column represents these eventpoints with various icons. You can set more than one eventpoint on the same line of source code. The Breakpoints column shows all eventpoints that you set for each line. [Table 18.4](#) explains the eventpoints and shows their corresponding icons.

Manipulating Program Execution

Eventpoints

Table 18.4 Eventpoints

Eventpoint	Icon	Explanation
Log Point		Logs or speaks a string or expression and records messages to the Log window
Pause Point		Pauses execution long enough to refresh debugger data
Script Point		Runs a script, application, or other item
Skip Point		Skip execution of a line of source code
Sound Point (Windows OS)		Plays a sound
Trace Collection Off		Stops collecting trace data for the Trace window
Trace Collection On		Starts collecting trace data for the Trace window

You can also create breakpoint templates to simplify the process of setting complex eventpoints. Creating a breakpoint template for an eventpoint is nearly identical to creating a breakpoint template for a breakpoint. The difference is using an eventpoint instead of a breakpoint as the starting point for creating the breakpoint template.

Eventpoints have *enabled* and *disabled* states. [Table 18.5](#) explains these states.

Table 18.5 Eventpoints—states

State	Icon	Explanation
Enabled	See Table 18.4	Indicates that the eventpoint is currently enabled. The debugger performs the specified task at an enabled eventpoint. Click the icon to disable the eventpoint.
Disabled		Indicates that the eventpoint is currently disabled. The debugger does not perform the specified task at a disabled eventpoint. Click the icon to enable the eventpoint.

TIP You can set an eventpoint in the Thread window and for selected variables in the Symbolics window.

Log Point

A Log Point logs or speaks a string or expression. A Log Point can also record messages to the Log window. You can configure the message that appears in the log window.

Setting a Log Point

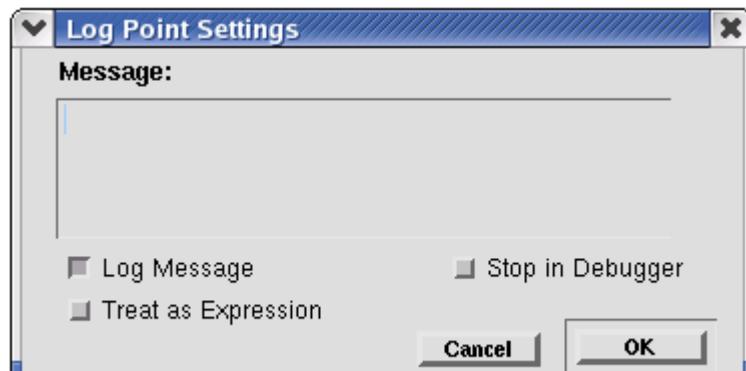
To set a Log Point, follow these steps:

1. Click the cursor on the line of source code at which you want to set the Log Point.
2. Click **Debug > Set Eventpoint > Set Log Point**.

The **Log Point Settings** window appears ([Figure 18.3](#)).

3. Enter the text of your log message in the **Message** text box.

Figure 18.3 Log Point Settings window



4. Check at least one of these checkboxes:
 - **Log Message**—check to have the IDE display your message in a Message window when program execution reaches the Log Point

Manipulating Program Execution

Eventpoints

- **Speak Message (Windows OS)**—check to have the IDE use the sound capabilities of the host operating system to speak the message that you enter in the **Message** text box.

NOTE (Windows) Install the Speech software development kit (SDK) in order to have the **Speak Message** feature work correctly.

- **Treat as Expression**—check to have the IDE evaluate the text you enter in the Message text box as an expression. For example, if you enter the name of a variable in the Message text, the debugger writes the value of that variable in the console output window.
 - **Stop in Debugger**—check to stop program execution in the debugger
5. Click the **OK** button to confirm your settings.

Example use: If you want to display the value of a variable each time some code is executed, set a log point, check the Log Message and Treat as expression boxes and enter the variable name in the edit box, then click OK.

Clearing a Log Point

To clear a Log Point, follow these steps:

1. Select the Log Point that you want to clear.

Click the cursor on the line of source code that has the Log Point, or select the Log Point by name in the **Breakpoints** window.

2. Click **Debug > Clear Eventpoint > Clear Log Point**.

Pause Point

A Pause Point suspends program execution long enough to refresh debugger data. For example, without setting a pause point, you must wait for the debugger to halt program execution before it can refresh data. Setting a Pause Point, however, lets you pause the debugging session to give the debugger time to refresh the data.

Setting a Pause Point

To set a Pause Point, follow these steps:

1. Click the cursor on the line of source code at which you want to set the Pause Point.
 2. Click **Debug > Set Eventpoint > Set Pause Point**.
-

Clearing a Pause Point

To clear a Pause Point, follow these steps:

1. Select the Pause Point that you want to clear.

Click the cursor on the line of source code that has the Pause Point, or select the Pause Point by name in the **Breakpoints** window.

2. Click **Debug > Clear Eventpoint > Clear Pause Point**.

Script Point

A Script Point runs a script, application, or other item. After you set a Script Point at a line of source code, its associated action occurs when program execution arrives at that line. For example, you can set a Script Point that performs these actions:

- (Windows) execute a file as if you had used a Windows command line
 - (Mac OS) launch an AppleScript or application
-

Setting a Script Point

To set a Script Point, follow these steps:

1. Click the cursor on the line of source code at which you want to set the Script Point.
2. Click **Debug > Set Eventpoint > Set Script Point**.

The **Script Point Settings** window appears ([Figure 18.4 on page 238](#)).

Manipulating Program Execution

Eventpoints

Figure 18.4 Script Point Settings window



3. Use the list box to specify **Commands** or **Script File**.

Specify **Commands** (Windows) if you intend to enter a command line that executes a file. Specify **Script File** if you intend to enter a path to a script file.

4. Enter the text of your Script Point in the text box.

Enter a command line or a path to a script file.

5. Check **Stop in Debugger** if you want to stop program execution in the debugger.

6. Click the **OK** button to confirm your settings.

Clearing a Script Point

To clear a Script Point, follow these steps:

1. Select the Script Point that you want to clear.

Click the cursor on the line of source code that has the Script Point, or select the Script Point by name in the **Breakpoints** window.

2. Click **Debug > Clear Eventpoint > Clear Script Point**.

Skip Point

A Skip Point prevents the debugger from executing a line of source code. This eventpoint is useful when you are aware of a line that you need to fix, but would like to go ahead and debug the rest of the program. You can set a Skip Point at that line and have the debugger execute the rest of the project without executing that particular line.

NOTE Skip Points do not work with the Java programming language.

Setting a Skip Point

To set a Skip Point, follow these steps:

1. Click the cursor on the line of source code at which you want to set the Skip Point.
 2. Click **Debug > Set Eventpoint > Set Skip Point**.
-

Clearing a Skip Point

To clear a Skip Point, follow these steps:

1. Select the Skip Point that you want to clear.
Click the cursor on the line of source code that has the Skip Point, or select the Skip Point by name in the **Breakpoints** window.
2. Click **Debug > Clear Eventpoint > Clear Skip Point**.

Sound Point (Windows OS)

A Sound Point is an audible alert. You can set a Sound Point so that when you step or run through code, the IDE plays a sound when program execution arrives at the line that has a Sound Point. Unlike a Log Point set to **Speak Message**, which speaks the message you specify, the Sound Point plays a simple notification sound.

Setting a Sound Point

To set a Sound Point, follow these steps:

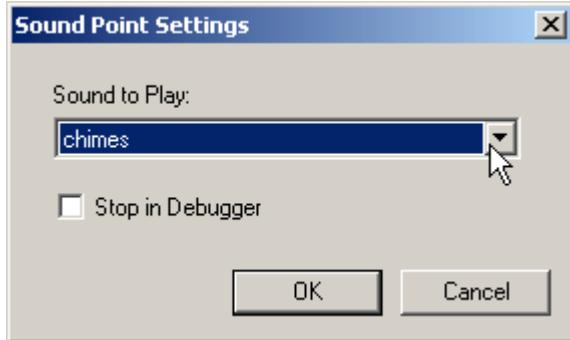
1. Click the cursor on the line of source code at which you want to set the Sound Point.
2. Click **Debug > Set Eventpoint > Set Sound Point**.

The **Sound Point Settings** window appears ([Figure 18.5 on page 240](#)).

Manipulating Program Execution

Eventpoints

Figure 18.5 Sound point settings



3. Use the **Sound to Play** list box to specify the notification sound that you want the IDE to play when program execution arrives at the Sound Point.
 4. Check **Stop in Debugger** if you want to stop program execution in the debugger.
 5. Click the **OK** button to confirm your settings.
-

Clearing a Sound Point

To clear a Sound Point, follow these steps:

1. Select the Sound Point that you want to clear.

Click the cursor on the line of source code that has the Sound Point, or select the Sound Point by name in the **Breakpoints** window.

2. Click **Debug > Clear Eventpoint > Clear Sound Point**.

Trace Collection Off

A Trace Collection Off eventpoint stops the collection of trace data. This eventpoint is useful when you want trace collection to stop when program execution reaches a line of source code that you specify.

Setting a Trace Collection Off Eventpoint

To set a Trace Collection Off eventpoint, follow these steps:

1. Click the cursor on the line of source code at which you want to set the Trace Collection Off eventpoint.
 2. Click **Debug > Set Eventpoint > Set Trace Collection Off**.
-

Clearing a Trace Collection Off Eventpoint

To clear a Trace Collection Off eventpoint, follow these steps:

1. Select the Trace Collection Off eventpoint that you want to clear.

Click the cursor on the line of source code that has the Trace Collection Off eventpoint, or select the Trace Collection Off eventpoint by name in the **Breakpoints** window.

2. Click **Debug > Clear Eventpoint > Clear Trace Collection Off**.

Trace Collection On

A Trace Collection On eventpoint starts the collection of trace data. This eventpoint is useful when you want trace collection to start when program execution reaches a line of source code that you specify.

Setting a Trace Collection On Eventpoint

To set a Trace Collection On eventpoint, follow these steps:

1. Click the cursor on the line of source code at which you want to set the Trace Collection On eventpoint.
 2. Click **Debug > Set Eventpoint > Set Trace Collection On**.
-

Clearing a Trace Collection On Eventpoint

To clear a Trace Collection On eventpoint, follow these steps:

1. Select the Trace Collection On eventpoint that you want to clear.

Click the cursor on the line of source code that has the Trace Collection On eventpoint, or select the Trace Collection On eventpoint by name in the **Breakpoints** window.

2. Click **Debug > Clear Eventpoint > Clear Trace Collection On**.
-

Working with Eventpoints

This section explains how to work with eventpoints in your source code and in the **Breakpoints** window.

Viewing Eventpoint Properties

After you set an eventpoint, you can view and modify its properties.

To view properties for an eventpoint, select its name in the **Breakpoints** window and click **Breakpoints > Breakpoint Properties**.

Disabling an Eventpoint

Disable an eventpoint to prevent it from performing its specified action. The disabled eventpoint remains at the source-code line at which you set it, so that you can enable it later. Disabling the eventpoint is easier than clearing it and re-creating it from scratch.

To disable an eventpoint, follow these steps:

1. Select the eventpoint that you want to disable.

Select the eventpoint by name in the **Breakpoints** window, or click the cursor on the source-code line that contains the eventpoint.

2. Click **Debug > Disable Eventpoint**.

The **Disable Eventpoint** menu appears.

3. From the menu, click the **Disable *Eventpoint*** command, where *Eventpoint* is the type of eventpoint that you want to disable.

- The enabled eventpoint icon changes to a disabled eventpoint icon (shown at left). The disabled eventpoint icon indicates that the eventpoint does not perform its specified action.

Enabling an Eventpoint

Enable an eventpoint to have it perform its specified action during program execution. Enabling an eventpoint that you previously disabled is easier than clearing it and re-creating it from scratch.

To enable an eventpoint, follow these steps:

1. Select the eventpoint that you want to enable.

Select the eventpoint by name in the **Breakpoints** window, or click the cursor on the source-code line that contains the eventpoint.

2. Click **Debug > Enable Eventpoint**.

The **Enable Eventpoint** menu appears.

3. From the menu, click the **Enable *Eventpoint*** command, where *Eventpoint* is the type of eventpoint that you want to enable.

The disabled eventpoint icon changes to its original eventpoint icon ([Table 18.4 on page 234](#)). The enabled eventpoint icon indicates that the eventpoint will perform its specified action.

Setting a Conditional Eventpoint

Use the **Condition** column of the **Breakpoints** window to set a conditional eventpoint. A *conditional eventpoint* has an associated conditional expression. The debugger evaluates the expression to determine whether the eventpoint performs its specified action.

A conditional eventpoint behaves in two different ways:

- If the expression evaluates to true (a non-zero value), the eventpoint performs its specified action.
- If the expression evaluates to false (a zero value), the eventpoint does not perform its specified action.

Follow these steps to set a conditional eventpoint:

1. Set an eventpoint that you want to associate with a conditional expression.
2. Click **View > Breakpoints** or **Window > Breakpoints Window**.

The **Breakpoints** window appears.

3. In the **Groups** or **Instances** pages of the Breakpoints window, find the eventpoint that you want to associate with a conditional expression.
4. Double-click the **Condition** column in align with the eventpoint.
5. Enter an expression in the **Condition** text box.

Manipulating Program Execution

Watchpoints

During subsequent debugging sessions, the debugger evaluates the expression to determine whether the eventpoint performs its specified action.

NOTE Alternatively, drag-and-drop an expression from a source view or from the Expression window into the Breakpoints window.

Watchpoints

You use *watchpoints* (sometimes referred to as access breakpoints or memory breakpoints) to halt program execution when your program reads or writes to a specific memory location. You can then examine the call chain, check register and variable values, and step through your code. You can also change values and alter the flow of normal program execution.

NOTE You cannot set watchpoint on local variable, because the debugger cannot detect watchpoints for variables stored on the stack or in registers.

You can create breakpoint templates to simplify the process of setting complex watchpoints. Creating a breakpoint template for a watchpoint is nearly identical to creating a breakpoint template for a breakpoint. The difference is using a watchpoint instead of a breakpoint as the starting point for creating the breakpoint template.

Watchpoints have *enabled* and *disabled* states. [Table 18.6](#) explains these states.

Table 18.6 Watchpoints—states

State	Icon	Explanation
Enabled		Indicates that the watchpoint is currently enabled. The debugger halts program execution at an enabled watchpoint. Click the icon to disable the watchpoint.
Disabled		Icon disappears, indicating that the watchpoint is currently disabled. The debugger does not halt program execution at a disabled watchpoint. Click the icon to enable the watchpoint.

NOTE For most hardware debug targets, you may set a maximum of one watchpoint at any time. However, different hardware targets may let you set two or more. Consult your targeting manual for watchpoint information specific to your target.

Setting a Watchpoint

Use the **Set Watchpoint** command to set a watchpoint. A watchpoint suspends program execution when the memory location that you specify changes value. The debugger does not execute the line of source code that contains the watchpoint.

NOTE The watchpoint implementation differs between CodeWarrior products, and the steps below may not completely describe your watchpoint setting process. For example, some products let you specify additional watchpoint parameters such as whether you are setting a read or a write watchpoint.

To set a watchpoint on a memory range, follow these steps:

1. Click **Project > Debug**.

A debugging session starts.

2. Click **Data > View Memory**.

A **Memory** window appears.

3. Select a range of bytes in the Memory window.

Do not double-click the range of bytes.

4. Click **Debug > Set Watchpoint**.

An underline appears beneath the selected range of bytes, indicating that you set a watchpoint in that range.

TIP You can change the color of the watchpoint underline in the **Display Settings** panel of the **IDE Preferences** window

To set a watchpoint on a variable in the thread window or variable window, follow these steps:

Manipulating Program Execution

Watchpoints

1. Make the thread window or variable window active.
 2. Right-click the variable in the variables pane.
A drop-down menu appears
 3. Select **Set Watchpoint**.
-

Viewing Watchpoint Properties

After you set a watchpoint, you can view and modify its properties.

To view properties for a watchpoint, select its name in the **Breakpoints** window and click **Breakpoints > Breakpoint Properties**.

Disabling a Watchpoint

Disable a watchpoint to prevent it from affecting program execution. The disabled watchpoint remains at the memory location at which you set it, so that you can enable it later.

To disable a watchpoint, select its name in the **Breakpoints** window, or select the range of bytes in the **Memory** window at which you set the watchpoint, and click **Debug > Disable Watchpoint**.

The enabled watchpoint icon disappears, which indicates a disabled watchpoint.

Enabling a Watchpoint

Enable a watchpoint to have it halt program execution when its associated memory location changes value. Enabling a watchpoint that you previously disabled is easier than clearing it and re-creating it from scratch.

To enable a watchpoint, select its name in the **Breakpoints** window, or select the range of bytes in the **Memory** window at which you set the watchpoint, and click **Debug > Enable Watchpoint**.

- ➊ The enabled watchpoint icon appears (shown at left), which indicates an enabled watchpoint.

Clearing a Watchpoint

Use the **Clear Watchpoint** command to clear a watchpoint.

To clear a watchpoint in the Memory window, select range of bytes at which you set the watchpoint and click **Debug > Clear Watchpoint**.

To clear a watchpoint in the **Breakpoints** window, select its name from the list in the **Groups** or **Instances** pages and press Delete.

Clearing All Watchpoints

Use the **Clear All Watchpoints** command to clear all watchpoints from your projects.

To clear all watchpoints, click **Debug > Clear All Watchpoints**. The **Breakpoints** window reflects your changes.

Setting a Conditional Watchpoint

Use the **Condition** column of the **Breakpoints** window to set a conditional watchpoint. A *conditional watchpoint* has an associated conditional expression. The debugger evaluates the expression to determine whether to halt program execution at that watchpoint.

A conditional watchpoint behaves in two different ways:

- If the expression evaluates to true (a non-zero value), the debugger halts program execution when the memory location associated with the watchpoint changes value.
- If the expression evaluates to false (a zero value), program execution continues without stopping.

Follow these steps to set a conditional watchpoint:

1. Set a watchpoint that you want to associate with a conditional expression.
2. Click **View > Breakpoints** or **Window > Breakpoints Window**.

The **Breakpoints** window appears.

3. In the **Groups** or **Instances** pages of the Breakpoints window, find the watchpoint that you want to associate with a conditional expression.

Manipulating Program Execution

Special Breakpoints

4. Double-click the **Condition** column adjacent to the watchpoint.
5. Enter an expression in the **Condition** text box.

During subsequent debugging sessions, the debugger evaluates the expression to determine whether to halt program execution at the conditional breakpoint.

NOTE Alternatively, drag-and-drop an expression from a source view or from the Expression window into the Breakpoints window.

Special Breakpoints

Special breakpoints halt program execution for very specific reasons:

- program execution arrives at the beginning of the function `main()`
- a C++ or Java exception occurs
- an event occurs that the debugger plug-in defines as a break event

You cannot change or delete special breakpoints, but you can enable and disable them.

Disabling Special Breakpoints

Disable special breakpoints to prevent them from affecting program execution.

-  To disable special breakpoints, click the Active icon to the left of the **Special** group in the **Groups** page of the **Breakpoints** window.

The active icon changes to an inactive icon, which indicates that the special breakpoints are disabled.

Enabling Special Breakpoints

Enable special breakpoints to have them halt program execution.

-  To enable special breakpoints, click the Inactive icon (shown at left) to the left of the **Special** group in the **Groups** page of the **Breakpoints** window.

The inactive icon changes to an active icon, which indicates that the special breakpoints are enabled.

Working with Variables

This chapter explains how to work with variables in a CodeWarrior™ IDE debugging session. The following windows show various types of information about variables.

- Global Variables window—shows information about global and static variables in your project
- Variable window—shows information for an individual variable in your project
- Expressions window—shows variable values and lets you form calculation expressions based on those values

This chapter contains these sections:

- [“Global Variables Window” on page 249](#)
- [“Variable Window” on page 251](#)
- [“Expressions Window” on page 254](#)

Global Variables Window

The **Global Variables** window shows all global and static variables for each process that you debug. You can open separate Global Variables windows for each process in the same build target. Use the window to observe changes in variable values as the program executes.

[Figure 19.1 on page 250](#) shows the Global Variables window. [Table 19.1 on page 250](#) explains the items in the window.

Working with Variables

Global Variables Window

Figure 19.1 Global Variables window

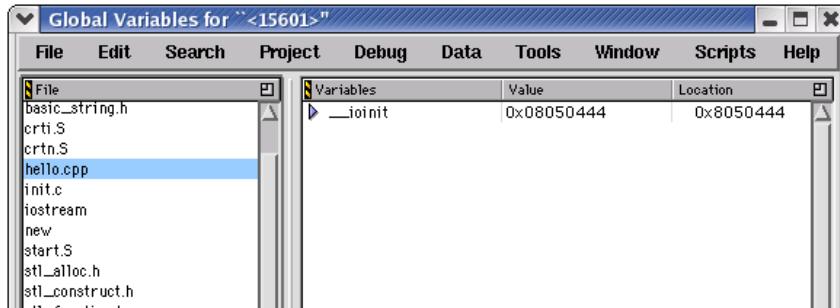


Table 19.1 Global Variables window—items

Item	Explanation
File	Lists source files that declare global or static variables. Click a source file to view its static variables. Click Global Variables to view all global variables declared in the program.
Variables	Lists variables according to the file selected in the File pane. Double-click a variable to display it in a separate Variable window.

Opening the Global Variables Window

Use the Global Variables window to display global variables declared in a program or static variables declared in source files that comprise the program.

To open the Global Variables window, select **View > Global Variables** or **Window > Global Variables Window**.

Viewing Global Variables for Different Processes

You can open a separate Global Variables window for each process that the same parent application creates.

To open the Global Variables window for a particular process, follow these steps:

1. Click **Project > Debug**.

A debugging session starts.

2. In the Thread window toolbar, use the Process list box to specify the process that has the global variables that you want to examine.
3. Click **View > Global Variables** or **Window > Global Variables Window**.

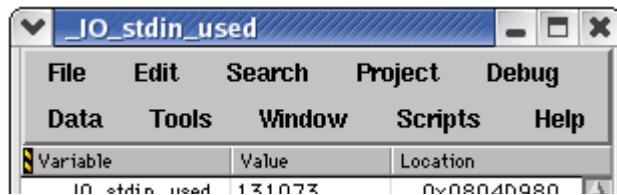
Repeat these steps for each process that has global variables that you want to examine.

Variable Window

A Variable window allows manipulation of a single variable or variable hierarchy used in source code. For a local variable, the window closes after program execution exits the routine that defines the variable.

[Figure 19.2](#) shows the Variable window.

Figure 19.2 Variable window



Opening a Variable Window

1. Select a variable in any window pane that lists variables.
2. Open a Variable window:
 - Select **Data > View Variable**, or
 - Double-click the variable.

A Variable window appears. Double-click a value to change it.

TIP Use Variable windows to monitor individual variables independently of other windows. For example, use a Variable window to continue monitoring a variable that leaves the current scope of program execution in the Thread window.

Working with Variables

Variable Window

Alternatively, use a contextual menu to open a variable window, as explained in [Table 19.2](#).

Table 19.2 Opening a Variable window by using a contextual menu

On this host...	Do this...
Windows	Right-click the variable and select View Variable .
Macintosh	Control-click the variable and select View Variable .
Solaris	Click and hold on the variable, then select View Variable .
Linux	Click and hold on the variable, then select View Variable .

Manipulating Variable Formats

You can change the way the Variables window displays data. For example, you can add labels to variable data so that those labels appear in the Variables window and clarify the displayed data.

For example, suppose you have the structure defined in [Listing 19.1](#).

Listing 19.1 Sample structure definition

```
struct Rect {  
    short          top;  
    short          left;  
    short          bottom;  
    short          right;  
};
```

The Variables window might show an instance of the `Rect` structure like this:

myRect 0x000DCEA8

You can create an XML file that defines a new way to display the structure, as shown in [Listing 19.2](#).

Listing 19.2 Sample variable format definition

```
<variableformats>
  <variableformat>
    <osname>osWin32</osname>
    <runtimename>runtimeWin32</runtimename>
    <typename>Rect</typename>
    <expression>
      "{T: " + ^var.top +
      " L: " + ^var.left +
      " B: " + ^var.bottom +
      " R: " + ^var.right +
      "}{H: " + (^var.bottom - ^var.top) +
      " W: " + (^var.right - ^var.left) + "}"
    </expression>
  </variableformat>
</variableformats>
```

Given this new variable format definition, the Variables window now shows the same `myRect` instance like this:

```
myRect {T: 30 L: 30 B: 120 R: 120}{H: 90 W: 90}
```

To manipulate variable formats, you place an XML file that defines the new format inside the `VariableFormats` directory at

`CodeWarrior/Bin/Plugins/Support/VariableFormats/`

where `CodeWarrior` is the path to your CodeWarrior installation.

The IDE reads the XML files in this directory to determine how to display variable data. [Table 19.3](#) explains the main XML tags that the IDE recognizes.

Table 19.3 Variable formats—XML tags

Tag	Explanation
<code>variableformats</code>	A group of variable format records.
<code>variableformat</code>	An individual variable format record.
<code>osname</code>	The operating system that defines the scope of this record.
<code>runtimename</code>	The runtime that defines the scope of this record.

Working with Variables

Expressions Window

Table 19.3 Variable formats—XML tags (*continued*)

Tag	Explanation
typename	The name of the Type that this record will format.
expression	The expression that reformats the variable display. The IDE evaluates this expression to determine the format that it applies to the variable. The IDE replaces all occurrences of the ^var placeholder with the name of the variable.

Expressions Window

The **Expressions** window helps you monitor and manipulate these kinds of items:

- global and local variables
- structure members
- array elements

[Figure 19.3](#) shows the Expressions window. [Table 19.4](#) explains items in window.

Figure 19.3 Expressions window

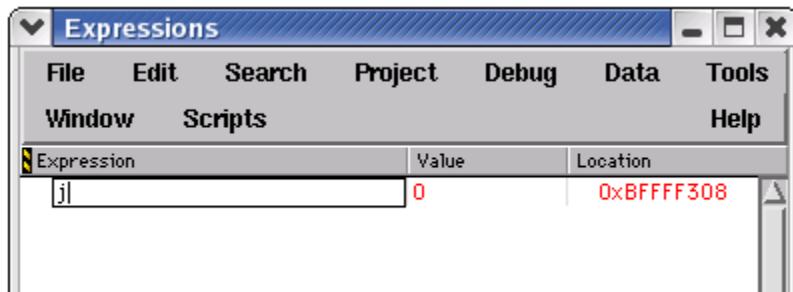


Table 19.4 Expressions window—items

Item	Explanation
Expression column	Lists expressions and expression hierarchies. Click the hierarchical controls to expand or collapse the expression view.
Value column	Shows the current value of each corresponding expression. Double-click a value to change it.

Opening the Expressions Window

Use the Expressions window to inspect frequently used variables as their values change during a debugging session.

To open the Expressions window, select **View > Expressions** or **Window > Expressions Window**.



Alternatively, click the Expressions button in the Thread window toolbar to open the Expressions window.

Adding Expressions

The Expressions window handles various ways of adding expressions for inspection.

To add an expression to the Expressions window, do this:

- Select the desired expression and choose **Data > Copy to Expression**, or
- Use the contextual menu with a selected expression, or
- Drag and drop an expression from another window into the Expressions window.

The Expressions window reflects the added expression. Drag expressions within the window to reorder them.

Working with Variables

Expressions Window

Adding a Constant Value to a Variable

You can enter an expression in the Expressions window that adds a constant value to a variable. Suppose `x` is a short integer type in the variable context of some function scope in C++ code. You can enter the expression `x+1` and the IDE computes the resulting value just as you would compute it on a calculator.

1. Select the variable to which you want to add a constant value.

For example, select `x`.

2. Enter an expression that adds a constant value to the variable.

For example, append `+1` to `x` so that the resulting expression is `x+1`.

The IDE adds the constant value to the variable and displays the result in the Expressions window.

Making a Summation of Two Variables

You can enter an expression in the Expressions window that computes the sum of two variables. Suppose `x` is a short integer type in the variable context of some function scope in C++ code. You can enter the expression `x+y` and the IDE computes the resulting value just as you would compute it on a calculator.

1. Select the variable to which you want to add another variable.

For example, select `x`.

2. Enter an expression that adds a second variable to the first variable.

For example, append `+y` to `x` so that the resulting expression is `x+y`.

The IDE computes the sum of the two variables and displays the result in the Expressions window.

Removing Expressions

The Expressions window handles various ways of removing expressions that no longer require inspection.

To remove an expression from the Expressions window:

- Select the expression and choose **Edit > Delete** or **Edit > Clear**, or

- Select the expression and press the Backspace or Delete key.

The Expressions window updates to reflect the removed expression.

NOTE Unlike the Variable window, the Expressions window does not remove a local variable after program execution exits the routine that defines the variable.

Working with Variables

Expressions Window

Working with Memory

This chapter explains how to work with variables in a CodeWarrior™ IDE debugging session. The following windows show various types of information about variables:

- Memory window—shows the memory that your project manipulates as it executes
- Array window—shows the contents of arrays that your project manipulates as it executes
- Registers window—shows the register contents of a processor
- Register Details window—shows a graphical representation of processor registers and explains register contents
- Cache window—shows processor or instructor cache data
- Trace window—shows collected trace information

This chapter contains these sections:

- [“Memory Window” on page 259](#)
- [“Array Window” on page 264](#)
- [“Registers Window” on page 266](#)
- [“Register Details Window \(Windows OS\)” on page 270](#)

Memory Window

The Memory window manipulates program memory content in various data types. Use this resizable window to perform these tasks:

- View memory
- Change individual memory bytes
- Set watchpoints

Working with Memory

Memory Window

CAUTION Arbitrarily changing memory contents could degrade the stability of the IDE, another program, or the operating system itself. Understand the consequences of manipulating memory.

[Figure 20.1](#) shows the Memory window. [Table 20.1 on page 260](#) explains the items in the window.

Figure 20.1 Memory window

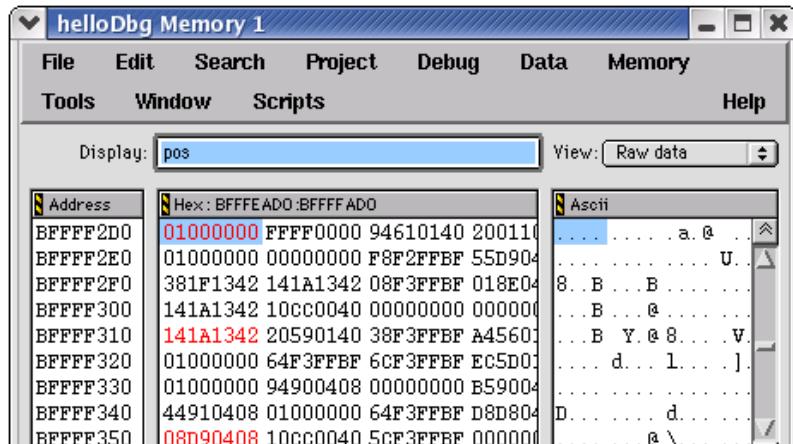


Table 20.1 Memory window—items

Item	Icon	Explanation
Display		Enter a symbol representing the starting address of memory to display. Valid symbols include addresses and non-evaluating expressions, such as <code>main</code> or <code>x</code> .
View	<input type="button" value="Raw data"/>	Select the data format in which to view memory contents.
Memory Space (for processors that support multiple memory spaces)		Choose the memory space in which to view selected variables or source code.

Table 20.1 Memory window—items (*continued*)

Item	Icon	Explanation
Previous Memory Block		Click to view the preceding block of memory.
Next Memory Block		Click to view the succeeding block of memory.
Address		Displays a contiguous range of memory addresses, beginning with the address entered in the Display field.
Hex		Displays a hexadecimal representation of the memory addresses shown in the Address pane.
Ascii		Displays an ASCII representation of the memory addresses shown in the Address pane.
Word Size	Word Size: <input type="button" value="32"/>	Select the bit size of displayed words.
Page (for processors that support multiple pages)	Page: <input type="button" value="X Memory"/>	Select the memory-space page in which to view source code.

Viewing and Changing Raw Memory

Use the **View Memory** command to view and change the raw contents of memory.

1. Select an item or expression that resides at the memory address to be examined.
2. Choose **Data > View Memory**.

A new Memory window appears.

Working with Memory

Memory Window

3. Select **Raw data** from the **View** list pop-up.

The contents of memory at the selected location appears in both hexadecimal and ASCII.

Scroll through memory by selecting the **Address**, **Hex**, or **ASCII** pane of the Memory window and then use the up and down arrow keys. Display a different memory location by changing the expression in the **Display** field.

Change the word size displayed in the Memory window by using the **Word Size** list pop-up. The choices are 8, 16, and 32 bits.

Change the contents of a particular memory location by double-clicking on that location in either the hexadecimal or ASCII pane of the Memory window. Replace the current value by entering a hexadecimal value in the **Hex** pane or a string of ASCII characters in the **ASCII** pane.

Alternatively, use a contextual menu to view and change memory, as explained in [Table 20.2](#).

Table 20.2 Opening a Memory window by using a contextual menu

On this host...	Do this...
Windows	Right-click the item and select View Memory .
Macintosh	Control-click the item and select View Memory .
Solaris	Click and hold on the item, then select View Memory .
Linux	Click and hold on the item, then select View Memory .

Viewing Memory Referenced by a Pointer

Use the **View Memory** command to inspect memory referenced by a pointer; including an address stored in a register.

1. Select a pointer in a source window.
2. Choose **Data > View Memory**.

A new Memory window appears.

3. Select **Raw data** from the **View** list pop-up.

The contents of memory referenced by the pointer appears in both hexadecimal and ASCII.

Viewing Different Memory Spaces

Use the **Page** list pop-up to view a particular memory space.

NOTE This feature is available only for processors that support multiple memory spaces.

1. Select the name of a variable or function in a source window.
2. Choose **Data > View Memory**.

A Memory window appears.

3. Select a memory space from the **Page** list pop-up.
4. Select **Raw data** from the **View** list pop-up if inspecting a variable. Select **Disassembly**, **Source**, or **Mixed** from the **View** list pop-up if inspecting source code.

The Memory window displays the selected memory-space page.

Setting a Watchpoint in the Memory Window

To set a Watchpoint using the **Memory** window, follow these steps:

1. **Run/Debug** your program.
2. Choose **Data > View Memory**.

This opens the **Memory** window.

3. Select a range of bytes in the **Memory** window.

Do not double-click the range of bytes.

4. Choose **Debug > Set Watchpoint**.

NOTE A red line appears under the selected variable in the Variable window, indicating that you have set a Watchpoint. You can change the color

Working with Memory

Array Window

of this line in the **Display Settings** panel of the IDE Preferences window (**Edit > IDE Preferences**).

Clearing Watchpoints from the Memory window

To clear a Watchpoint from the Memory window, follow these steps:

1. Select a range of bytes in the Memory window.
2. Choose **Debug > Clear Watchpoint**.

To clear *all* Watchpoints from the Memory window:

1. Open the Memory window.
You do not have to select a range of bytes.
2. Choose **Debug > Clear All Watchpoints**.

NOTE All Watchpoints clear automatically when the target program terminates or the debugger terminates the program. Watchpoints will reset next time the program runs.

Array Window

An Array window allows manipulation of a contiguous block of memory, displayed as an array of elements. The window lists array contents sequentially, starting at element 0.

The Array window title shows the base address bound to the array. The base address can bind to an address, a variable, or a register. An array bound to a local variable closes after the routine that defines the variable returns to the calling routine.

For array elements cast as structured types, a hierarchical control appears to the left of each element. Use these hierarchical controls to expand or collapse the display of each element's contents.

[Figure 20.2](#) shows an Array window. [Table 20.3](#) explains the items in the window.

Figure 20.2 Array window

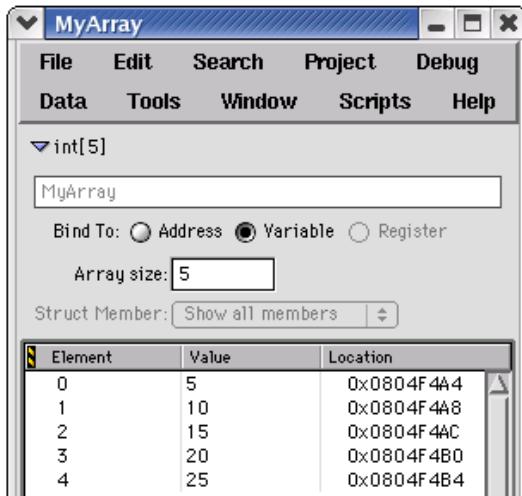


Table 20.3 Array window—items

Item	Icon	Explanation
Hierarchical control		Click to collapse the view of the information pane.
Bind To		Select the base address of the array: Address , Variable , or Register .
Array size	<input type="text" value="5"/> Array size: 5	Enter the number of elements to display in the Array window.
Struct Member	<input type="button" value="Show all members"/>	Select a specific member to show in each element, or show all members.
Element		Shows the array elements in a hierarchical list.
Value		Shows the value of each array element.
Location		Shows the address in memory of each array element.

Working with Memory

Registers Window

Opening an Array Window

Use the **View Array** command to manipulate a memory block in an Array window.

1. Select the array that you want to view.
2. Select **Data > View Array**.

A new Array window appears.

TIP Drag and drop a register or variable name into an Array window to set the base address. Use the **View Memory As** command to interpret memory displayed in an Array window as a different type.

Alternatively, use a contextual menu to open an Array window, as explained in [Table 20.4 on page 266](#).

Table 20.4 Opening an Array window by using a contextual menu

On this host...	Do this...
Windows	Right-click the array and select View Array .
Macintosh	Control-click the array and select View Array .
Solaris	Click and hold on the array, then select View Array .
Linux	Click and hold on the array, then select View Array .

Registers Window

The **Registers** window reveals a hierarchical view of these register types:

- general registers—contents of the central processing unit (CPU) of the host computer
- floating-point unit (FPU) registers—contents of the FPU registers
- registers specific to the host computer

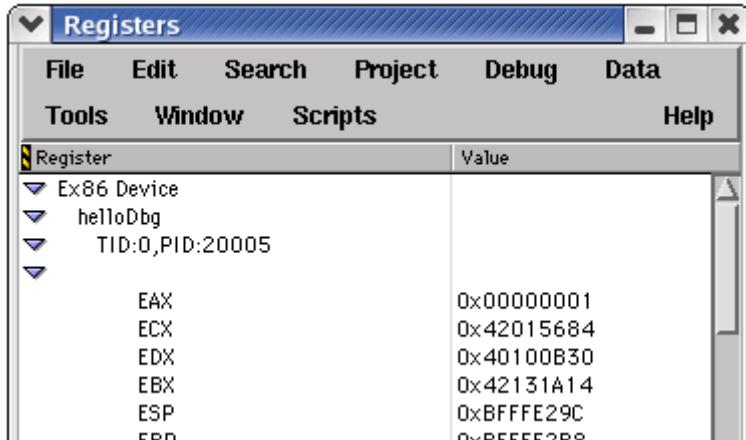
You can use the Register window to perform these tasks:

- expand the hierarchical items in the window and view their contents

- select and modify register values
- view documentation for individual registers (depending on the register)

[Figure 20.3](#) shows a sample Registers window.

Figure 20.3 Registers Window



General Registers

The **General Registers** are the register contents of the central processing unit (CPU) of the host computer. The exact listing of these registers depends on the host CPU and current build target. See the *Targeting* documentation for additional information.

FPU Registers

The **FPU Registers** are the register contents of the floating-point unit (FPU) of the host computer. The exact listing of these registers depends on the host FPU and current build target. See the *Targeting* documentation for additional information.

Host-specific Registers

The Registers window also lists additional register contents for registers specific to the host. The exact listing of these registers depends on the host computer and current build target. See the *Targeting* documentation for additional information.

Working with Memory

Registers Window

Opening the Registers Window

Open the **Registers** window to inspect and modify various register contents.

[Table 20.5](#) explains how to open the Registers window.

Table 20.5 Opening the Registers window

On this host...	Do this...
Windows	Select View > Registers
Macintosh	Select Window > Registers Window
Solaris	Select Window > Registers Window
Linux	Select Window > Registers Window

Viewing Registers

View registers to inspect and modify their contents.

1. Open the **Registers** window.
2. Expand the hierarchical list to view register groups.

Expanding the list shows the register groups that you can view or change.

3. Expand a register group.

Expanding a group shows its contents, by register name and corresponding value.

Changing Register Values

Change register values during program execution in order to examine program behavior.

1. Open the **Registers** window.
2. Expand the hierarchical list to view the names and corresponding values of the register that you want to modify.

3. Double-click the register value that you want to change.

The value highlights.

4. Enter a new register value.

5. Press **Enter** or **Return**.

The register value changes.

Changing Register Data Views

Change register data views to see register contents in a different format. For example, you can change the view of a register from binary to hexadecimal format.

1. Open the **Registers** window.

2. Expand the hierarchical list to view the names and corresponding values of the register.

3. Select the register value that you want to view in a different format.

The value highlights.

4. Select **Data > View as *format***, where *format* is the data format in which you want to view the register value. The register value changes format.

Available formats depend on the selected register value.

5. Select **Data > View as Default** to restore the original data format.

Alternatively, you can use a contextual menu to change the data format, as explained in [Table 20.6 on page 269](#).

Table 20.6 Changing the data format by using a contextual menu

On this host...	Do this...
Windows	Right-click the register value and select View as <i>format</i> .
Macintosh	Control-click the register value and select View as <i>format</i> .
Solaris	Click and hold on the register value and select View as <i>format</i> .
Linux	Click and hold on the register value and select View as <i>format</i> .

Opening Registers in a Separate Registers Window

Open registers in a separate Register Window to narrow the scope of registers that appear in a single window.

1. Open the **Registers** window.
2. Expand the hierarchical list to view the register or register group that you want to view in a separate Registers window.
3. Double-click the register or register group.
4. A new Registers window opens.

The new Registers window lists the name and value of the register that you double-clicked, or the names and values of the register group that you double-clicked.

Register Details Window (Windows OS)

The **Register Details** window lets you view detailed information about individual bits of registers from 2 bits to 32 bits in size. This window shows information for both system registers and memory-mapped registers. To open the Register Details window, click **View > Register Details** or **Window > Register Details Window**.

The Register Details window has fields that describe the register, its bitfields, and the values of those bitfields. XML files in the **Registers** folder of your CodeWarrior installation provide the information that appears in the window. The Registers folder is inside the **Support** folder. The Support folder is inside the **Plugins** folder of your CodeWarrior installation.

[Figure 20.4 on page 271](#) shows the Register Details window. [Table 20.7 on page 271](#) explains items in the window.

Figure 20.4 Register Details window

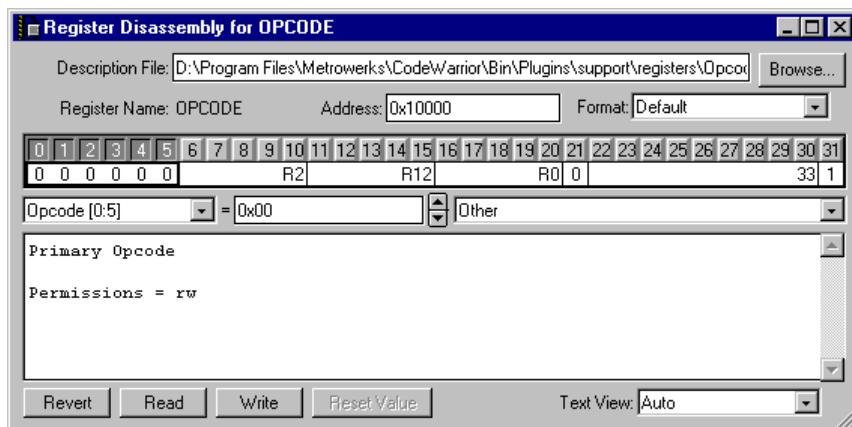


Table 20.7 Register Details window—items

Item	Icon	Explanation
Description File text box		Enter the name or full path to the XML file for the register you want to view, or click the Browse button to open a dialog box that you can use to specify the file.
Register Name		Shows the name of the register depicted in the window.
Address text box		Enter the starting address of the register values that you want to see in the Register Display. An error message appears if you enter an invalid starting address.
Format list box		Specify the data format for bit values in the Register Display: <ul style="list-style-type: none"> • Binary • Character • Decimal • Unsigned Decimal • Hexadecimal • Default—have the IDE determine the best format

Working with Memory

Register Details Window (Windows OS)

Table 20.7 Register Details window—items (*continued*)

Item	Icon	Explanation
Register Display		Shows a depiction of the register that you specify in the Description File text box, including individual register bits and their values.
Bitfield Name list box	<input type="button" value="Opcode [0:5]"/> 	Specify a bitfield to highlight in the Register Display. The Description portion of the window reflects available information for the bitfield. Select None to have the Description portion of the window reflect information for the entire register and not a bitfield in that register.
Bit Value text box	<input type="text" value="0x00"/>	Shows the current value of the bits in the Bitfield Name list box, according to the format that you specify in the Format list box. Click the spin buttons to increment or decrement the current value, or enter a new value in the text box. Changing the value changes only the Register Display. You must click the Write button to write the new value to the register itself.
Bit Value Modifier list box	<input type="text" value="Other"/>	Specify a new value for the selected bitfield, or view a brief explanation of specific bitfield values. Changing the value changes only the Register Display. You must click the Write button to write the new value to the register.
Description		Shows a description of the register or a selected bitfield in the register. Use the Description File text box to specify the register. Use the Text View list box to view specific register information, such as register descriptions, bitfield descriptions, and register details.

Table 20.7 Register Details window—items (*continued*)

Item	Icon	Explanation
Revert button		Click to change a modified value in the Register Display to its original value. If you clicked the Write button to write a new value to the register, you cannot revert that value.
Read button		Click to have the Register Display reflect current bit values from the register itself.
Write button		Click to write the bit values in the Register Display to the register. After you write new values to the register, you cannot revert them.
Reset Value button		Click to restore the default value for the selected bitfield. The IDE disables this button if the selected bitfield does not have a default value.
Text View list box		Use to specify information that appears in the Description portion of the window.

Description File

Enter in this text box the name of the register that you want to see in the Register Display of the **Register Details** window. Alternatively, enter the full path to the register description file on your computer, or click the **Browse** button to open a dialog box that lets you specify the register description file. The text box is not case sensitive.

After you enter a name or path, the debugger searches for a matching register description file in the **Registers** folder of your CodeWarrior installation and the project access paths. If the debugger finds a matching file, the Register Display updates the information 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, you can:

- type Opcode in the **Description File** text box and press Enter or Return, or
- type the full path to the opcode.xml file in the Registers folder and press Enter or Return.

Working with Memory

Register Details Window (Windows OS)

The debugger matches your entry with the `opcode.xml` file in the Registers folder. The Register Display in the Register Details window updates its information to show Opcode register details.

The debugger also updates the Register Display to show the current values in the register. If the debugger fails to update the display, an error message appears.

Register Display

This display shows the current contents of 32 bits of register data, starting at the address that you specify in the **Address** text box. The data appears according to the format that you specify in the **Format** list box.

The Register Display groups the 32 bits of data into 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 portion of the Register Details window.

Text View

Use this list box to change the information that appears in the Description portion of the Register Details window:

- **Auto**—select to have the IDE determine which information to display in the window
- **Register Description**—select to show information about the entire register, such as the name of the register itself and the meaning of its contents
- **Bitfield Description**—select to show information about the selected bitfield in the [Register Display](#), such as the name of the bitfield and its access permissions
- **Register Details**—select to show in-depth information about the current register, such as its name, its bit values, and bit-value explanations

Working with Debugger Data

This chapter explains how to work with data that the CodeWarrior™ IDE debugger generates. The following windows show various types of debugger data.

- Symbolics window—shows information that the debugger generates for a program
- Processes window—shows individual processes and tasks that the debugger can control
- Log window—shows messages generated during the debugging session

This chapter contains these sections:

- [“Symbolics Window” on page 275](#)
- [“System Browser Window” on page 278](#)
- [“Log Window” on page 281](#)

Symbolics Window

The **Symbolics** window displays information that the debugger generates for the active file. Symbolics information includes data about program variables, functions, data structures, and source files.

Select whether you want browser data generated by the compiler or the language parser, by choosing **Edit > targetname Settings...** and selecting the **Build Extras** target settings panel. Select the option from the Generate Browser Data From dropdown list. Symbolics information will be generated during the next build or debugging session.

[Figure 21.1 on page 276](#) shows the Symbolics window. [Table 21.1 on page 276](#) explains items in the window.

Working with Debugger Data

Symbolics Window

Figure 21.1 Symbolics window

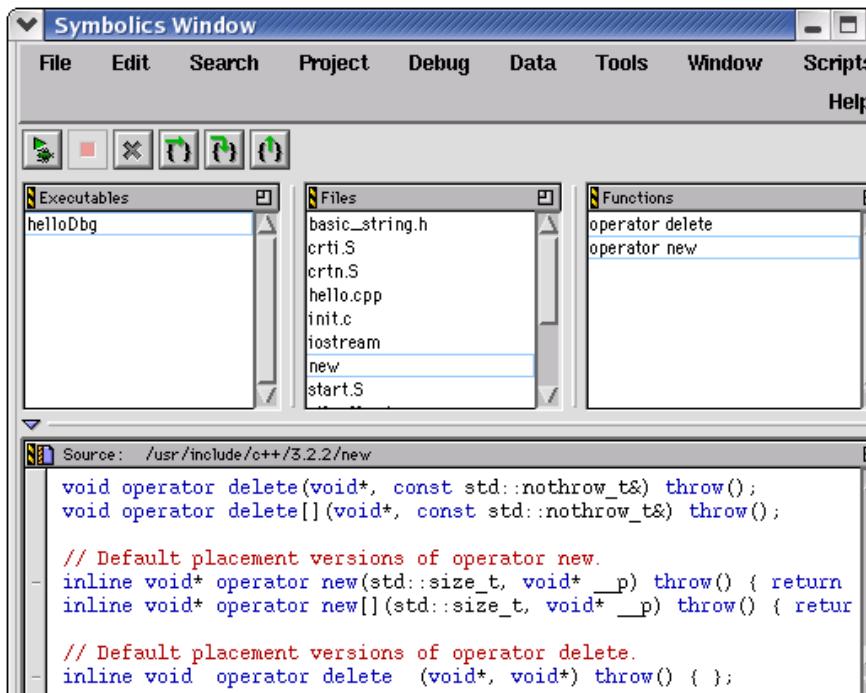


Table 21.1 Symbolics window—items

Item	Icon	Explanation
Debugger toolbar		Contains buttons that represent common debugging commands, such as stepping through code.
Executables pane		Lists recently used executable files that contain symbolics information.
Files pane		Lists source files in build target being debugged, for selected executable file.

Table 21.1 Symbolics window—items (*continued*)

Item	Icon	Explanation
Functions pane		Lists functions declared in the file selected in the Files pane.
Source pane		Displays source code in the file selected in the Files pane.

Opening the Symbolics Window

The Symbolics window displays information generated by the IDE for a file.

To open the Symbolics window, do one of these tasks:

- Select **View > Symbolics** or **Window > Symbolics window**.
- Open a symbolics file. The IDE typically appends .xSYM or .iSYM, to the names of these files.
- Open an executable file for which the IDE previously generated symbolics information. The IDE typically appends .exe or .app to these files.



Alternatively, click the Symbolics button in the Thread window toolbar to open the Symbolics window.

Using the Executables Pane

The **Executables** pane lists recently opened executable files for which the IDE generated symbolics information.

To use the pane, select an executable file in the list. The Files pane updates to display information for the selected executable file.

Using the Files Pane

For the selected executable file, the **Files** pane lists the source files in the build target being debugged.

To use the pane, select a file in the list. The Functions pane and Source pane update to display information for the selected file.

Working with Debugger Data

System Browser Window

Using the Functions Pane

The **Functions** pane lists functions declared in the selected file in the Files pane.

To use the pane, select a function in the list. The Source pane updates to display source code for the selected function.

Using the Source Pane

The **Source** pane displays source code for the selected function in the Functions pane, using the fonts and colors specified in the IDE Preferences window.

To use the pane, select a function in the Functions pane. The corresponding source code appears in the Source pane.

If the selected function does not contain source code, the Source pane displays the message **Source text or disassembly not available**.

NOTE Use the Source pane in the Symbolics window to view source code, copy source code, and set breakpoints. Use an editor window to modify the source code. Use a Thread window to view the currently executing statement.

System Browser Window

The **System Browser** window shows system level information about processes executing on various machines, like the host computer or the hardware under debugger control. The window shows this information:

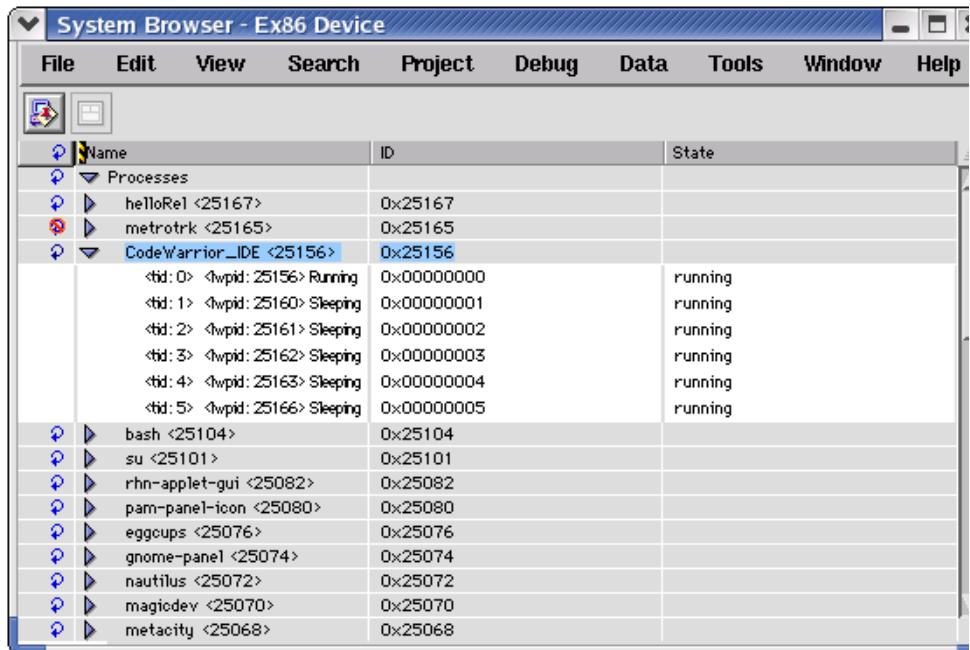
- running processes
- tasks for selected processes
- some hidden processes

Click on the expand icon for a process to view all tasks assigned to the selected process. Processes under debugger control appear in bold. Double-click a task to open it in a new Thread window, or choose the task name and click the Stack Crawl Window button.

[Figure 21.2](#) shows the System Browser window. [Table 21.2 on page 280](#) explains items in the window.

NOTE If the System Browser window does not show processes for a specific machine, you will need to start a debugging session for that machine. For example, you might need to debug a project that runs on external hardware in order to see executing processes for that hardware.

Figure 21.2 System Browser window



Working with Debugger Data

System Browser Window

Table 21.2 System Browser window—items

Item	Icon	Explanation
Attach to Process		Click to have the debugger control the selected process.
Stack Crawl window		Click to open a Thread window for the selected process.
Refresh		This icon indicates that information for selected item is periodically updated. Click this icon to toggle between refresh and no refresh.
No Refresh		This icon indicates that information is not updated for the selected item.
Expand		Click to expand a process and list related tasks.

Opening the System Browser Window

Use the **System Windows** or **System** menu command to view and manipulate active processes on a selected machine. If multiple machines are available, select each machine from the System Windows submenu to display multiple System Browser windows. If you choose a machine that is already open, the existing window will be brought to the front.

NOTE The System Browser window appears on platforms that support it.

[Table 21.3](#) explains how to open the System Browser window.

Table 21.3 Opening the System Browser window

Menu Bar Layout	Do this...
Windows	Select View > System
Macintosh (also applies to Linux/Solaris)	Select Window > System Windows

Attaching Debugger to a Process

Click the **Attach to Process** button to assign a selected process to a new debugging session. This assignment allows the debugger to control processes that it does not otherwise recognize. For example, you can click the Attach to Process button to assign dynamic link libraries or shared libraries to the debugger.

1. Select a process to attach to the debugger.

2. Click **Attach to Process** 

3. Select an executable to attach to the process.

4. Click **OK** to display the Stack Crawl (Thread) window for the process.

The debugger assumes control of the selected process. Processes under debugger control appear in bold.

Log Window

The **Log** window displays messages during program execution. Select the **Log System Messages** option in the **Debugger Settings** panel to activate the Log window.

The IDE allows you to save Log window contents to a `.txt` (text) file and copy text from the Log window to the system clipboard.

Windows-hosted Log window messages include:

- Dynamic Link Library (DLL) loading and unloading
- debugging `printf()` messages

Macintosh-hosted Log window messages include:

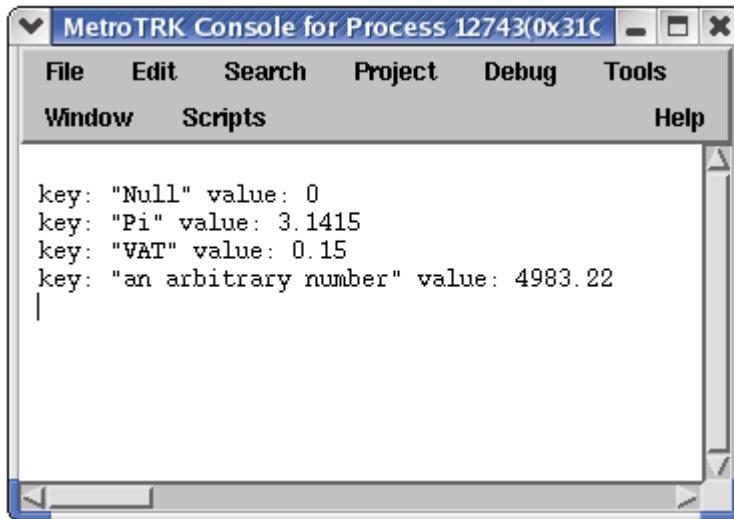
Working with Debugger Data

Log Window

- PowerPC™ code fragments
- DebugStr() messages

[Figure 21.3](#) shows a Windows-hosted Log window.

Figure 21.3 Log window



Opening the Log Window

Use the **Debugger Settings** preference panel to enable the message logging option. The Log window records these types of messages for a program during a debugging session:

- the start of new tasks
 - routine entry and exit
 - Windows: DLL loading and unloading, and debug printf() messages
 - Macintosh: PowerPC code-fragment loading and DebugStr() messages
1. Select the **Log System Messages** option in the **Debugger Settings** target settings preference panel.

2. Select **Project > Debug**.

The Log window appears. It allows you to select, copy, and save logged text to a file for later analysis. See the *Targeting* documentation for additional information.

Working with Debugger Data

Log Window

Working with Hardware Tools

This chapter explains the CodeWarrior™ IDE hardware tools. Use these tools for board bring-up, test, and analysis.

NOTE Some of the IDE features described in this chapter are not supported by all products, such as, the Flash programmer window, hardware diagnostic window, and logic analyzer. Some screen captures in this chapter were taken on a Windows PC and used as examples only, actual appearance will vary slightly on other host platforms.

This chapter contains these sections:

- [“Flash Programmer Window” on page 285](#)
- [“Hardware Diagnostics Window” on page 297](#)
- [“Working with a Logic Analyzer” on page 309](#)
- [“Trace Window” on page 314](#)
- [“Cache Window” on page 315](#)
- [“Profile Window” on page 316](#)
- [“Command Window” on page 317](#)

Flash Programmer Window

The **Flash Programmer** window lists global options for the flash programmer hardware tool. These preferences apply to every open project file.

[Figure 22.1 on page 286](#) shows the Flash Programmer window. [Table 22.1 on page 286](#) explains the items in the window.

To open the Flash Programmer window, click **Tools > Flash Programmer**.

The Flash Programmer window contains these panels:

Working with Hardware Tools

Flash Programmer Window

- [Target Configuration](#)
- [Flash Configuration](#)
- [Program / Verify](#)
- [Erase / Blank Check](#)
- [Checksum](#)

Figure 22.1 Flash Programmer window

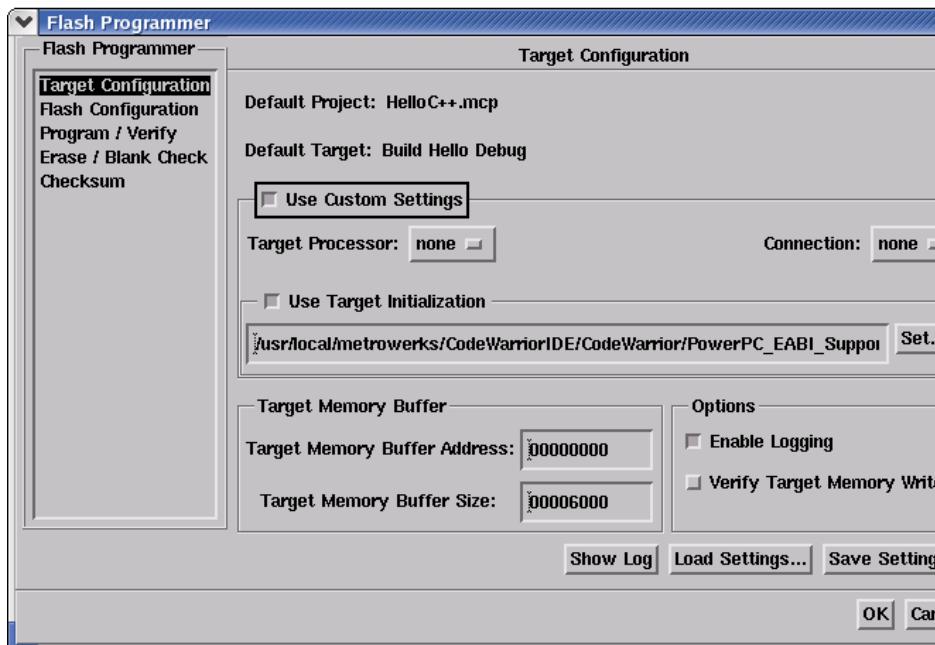


Table 22.1 Flash Programmer window—items

Item	Explanation
Flash Programmer pane	Shows a list of panel names. Click a panel name to display that panel.
Show Log	Click to display a text file that logs flash programmer actions. Check the Enable Logging checkbox in the Options group to enable this button.

Table 22.1 Flash Programmer window—items (*continued*)

Item	Explanation
Load Settings	Click to restore previously saved settings for the current panel.
Save Settings	Click to save settings for the current panel to a file.
OK	Click to save changes to all panels and close the window.
Cancel	Click to discard changes to all panels and close the window.

Target Configuration

The **Target Configuration** panel configures general flash programmer settings. [Figure 22.2 on page 287](#) shows the Target Configuration panel. [Table 22.2 on page 288](#) explains items in the panel.

Figure 22.2 Target Configuration panel



Working with Hardware Tools

Flash Programmer Window

Table 22.2 Target Configuration panel—items

Item	Explanation
Default Project	Shows the current default project in the IDE.
Default Target	Shows the default build target. Clear the Use Custom Settings checkbox to have the IDE use connection settings from the build target for connecting to the hardware.
Use Custom Settings checkbox.	<p>Check to specify the connection information that you want to use for connecting to the hardware. In this case, the IDE can connect to the hardware without using settings from a project.</p> <p>Clear to use the connection information stored in the default project for connecting to the hardware. You cannot clear the checkbox if you do not have an active default project or default target.</p> <p>Connection information includes the information that you specify in the Target Processor list box, the Connection list box, and the Use Target Initialization text box.</p>
Target Processor text/list box	Use to specify the hardware processor.
Connection list box	Use to specify the method that the IDE uses to connect to the hardware.
Use Target Initialization checkbox and text box	<p>Check to specify an initialization file for the hardware connection. Enter the initialization file path in the text box, or click the Browse button to open a dialog box that you can use to specify the initialization file path.</p> <p>Clear if you do not want to use an initialization file for the hardware connection.</p>
Target Memory Buffer Address text box	<p>Specify the starting address of an area in RAM that the flash programmer can use as a scratch area. The flash programmer must be able to access this starting address through the remote connection (after the hardware initializes).</p> <p>The flash programmer should not modify any memory location other than the target memory buffer and flash memory.</p> <p>For example, the flash programmer uses the target memory buffer to download and execute the flash device driver.</p>
Target Memory Buffer Size text box	<p>Specify the size of an area in RAM that the flash programmer can use as a scratch area, starting at the address you specify in the Target Memory Buffer Address text box.</p> <p>The flash programmer should not modify any memory location other than the target memory buffer and flash memory.</p>

Table 22.2 Target Configuration panel—items (*continued*)

Item	Explanation
Enable Logging checkbox	<p>Check to have the IDE generate detailed status information during flash operations. Checking this checkbox enables the Show Log button.</p> <p>Clear to disable logging of detailed status information during flash operations. Clearing this checkbox disables the Show Log button.</p> <p>Click the Show Log button to view the status information.</p>
Verify Target Memory Writes checkbox	<p>Check to have the IDE verify all write operations to the hardware RAM by reading the result of each write operation.</p> <p>Clear to have the IDE perform write operations without verifying them.</p>

Flash Configuration

The **Flash Configuration** panel configures settings for the flash device on the hardware device. [Figure 22.3](#) shows the Flash Configuration panel. [Table 22.3 on page 290](#) explains the items in the panel.

Working with Hardware Tools

Flash Programmer Window

Figure 22.3 Flash Configuration panel

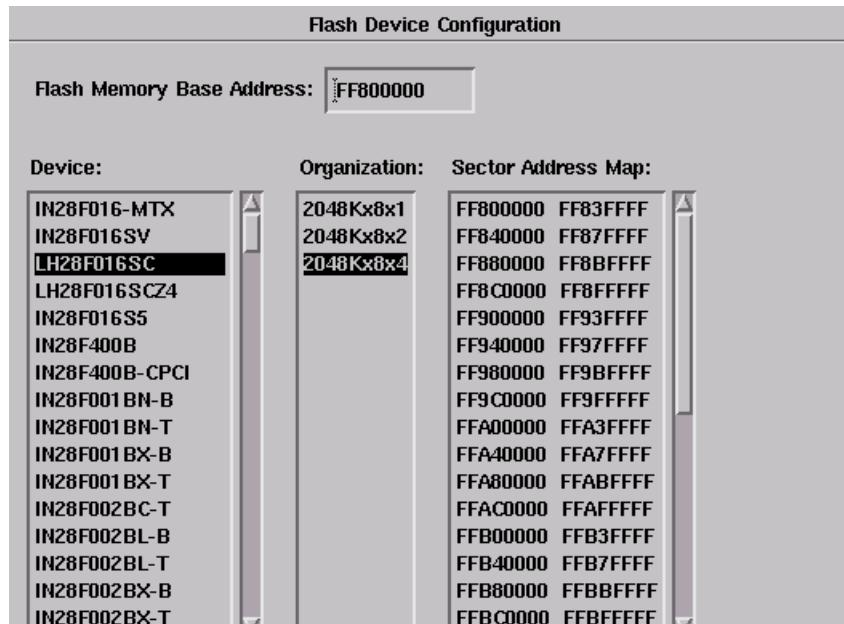


Table 22.3 Flash Configuration panel—items

Item	Explanation
Flash Memory Base Address text box	Enter the base address of the beginning of flash memory on the hardware device. Enter the address based on the perspective of the hardware device.
Device pane	Shows an alphabetical list of supported flash device types. Select a device type from this pane. Your selection determines the contents of the Organization and Sector Address Map panes.

Table 22.3 Flash Configuration panel—items (*continued*)

Item	Explanation
Organization pane	<p>Shows a list of supported layouts of flash memory in the hardware design, based on your selection in the Device pane. Each list item is of the form <i>ChipCapacityxDataBusWidthxNumberOfChipsInLayout</i>. Select an organization from this pane. Your selection determines the contents of the Sector Address Map pane.</p> <p>For example, 2048Kx8x2 indicates a chip capacity of 2048 kilobytes, a byte-wide interface to the data bus, and a 2-chip hardware layout.</p> <p>For hardware layouts of 2 or more chips, assume an interleaved organization. For example, for a 2048Kx16x2 organization, there are 2 chips on a 32-bit bus, and each chip provides 16 bits of data.</p>
Sector Address Map pane	<p>Shows a map of sector addresses that reflects your selections in the Device and Organization panes and your entry in the Flash Memory Base Address text box. This map is for informational purposes only.</p>

Program / Verify

The **Program / Verify** panel lets you program an image into flash memory and verify the programming operation. [Figure 22.4 on page 292](#) shows the Program / Verify panel. [Table 22.4 on page 292](#) explains the items in the panel.

Working with Hardware Tools

Flash Programmer Window

Figure 22.4 Program / Verify panel

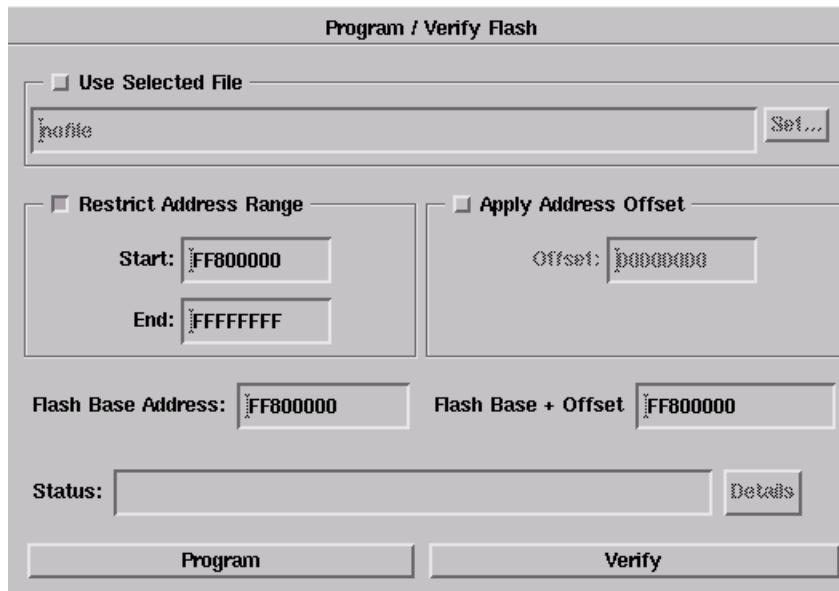


Table 22.4 Program / Verify panel—items

Item	Explanation
Use Selected File	<p>Check to specify a file to program into flash memory. Enter the file path in the text box, or click the Browse button to locate the file path.</p> <p>Clear to have the IDE program flash memory with the file that the default build target determines.</p> <p>The file determines the address to which the IDE programs flash memory. If you specify a file that does not contain address information, such as a binary file, the IDE programs flash memory at address zero. Check the Apply Address Offset checkbox to specify an address offset from zero.</p>
File Type	Select the file type. Options are: Auto Detect, Binary/Raw Format, Elf Format, or Motorola S-Record Format.

Table 22.4 Program / Verify panel—items (*continued*)

Item	Explanation
Restrict Address Range	<p>Check to use the Start and End text boxes to specify the address range in which you want the IDE to program flash data. If you use a binary file to program flash data, the flash programmer ignores data outside the address range that you specify.</p> <p>Clear to have the IDE determine the address range in which to program flash data.</p>
Start text box	<p>Enter the starting address of the range that you want the flash programmer to use for programming flash data.</p> <p>Check the Restrict Address Range checkbox to enable this text box.</p>
End text box	<p>Enter the ending address of the range that you want the flash programmer to use for programming flash data.</p> <p>Check the Restrict Address Range checkbox to enable this text box.</p>
Apply Address Offset checkbox	<p>Check to specify an offset at which to program flash data. The IDE adds this offset to the starting address that the file specifies. The flash programmer begins programming flash data at the starting address plus the offset.</p> <p>Clear to have the flash programmer begin programming flash data at the starting address that the file specifies. In this case, the IDE does not add an offset to the starting address.</p>
Offset text box	<p>Enter the offset to add to the starting address that the file specifies. The flash programmer begins programming flash data at the resulting address.</p> <p>Check the Apply Address Offset checkbox to enable this text box.</p>
Flash Base Address	Shows the base address of the beginning of flash memory on the hardware device. This address is the same address that you specify in the Flash Memory Base Address text box of the Flash Configuration panel.
Flash Base + Offset	Shows the resulting address of adding the offset value that you specify in the Offset text box to the Flash Base Address value. The flash programmer begins programming flash data at this resulting address.
Status	Shows flash programmer progress information. Click the Details button to show more thorough progress information.

Working with Hardware Tools

Flash Programmer Window

Table 22.4 Program / Verify panel—items (continued)

Item	Explanation
Program button	Click to have the flash programmer program flash data into the hardware device. The Status reflects flash programmer progress. The flash programmer does not check for blank flash memory before it begins programming the flash data.
Verify button	Click to have the IDE verify the data that the flash programmer programmed into the hardware device. The verify operation reads the flash data from the hardware device and compares that data against the image file on disk. The Status reflects flash programmer progress.

Erase / Blank Check

The **Erase / Blank Check** panel lets you erase an image from flash memory and check for blank memory. [Figure 22.5](#) shows the Erase / Blank Check panel.

[Table 22.5 on page 295](#) explains items in the panel.

Figure 22.5 Erase / Blank Check panel

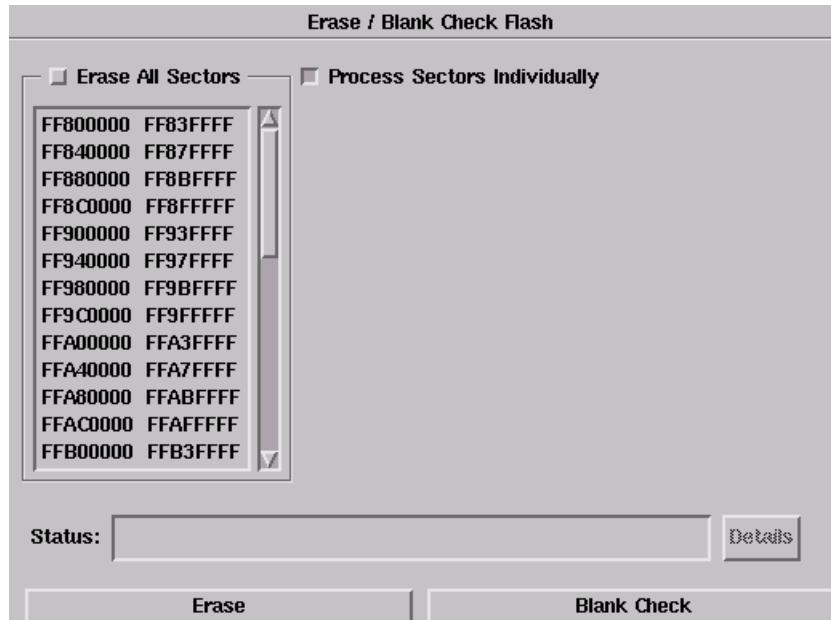


Table 22.5 Erase / Blank Check panel—items

Item	Explanation
All Sectors checkbox and list	Check to apply the erase or blank check operation to the entire flash memory. Clear to specify sectors that you want to erase or check for blanks. Select sectors in the list below the checkbox.
Erase Sectors Individually checkbox	Check to have the flash programmer ignore chip erase commands and erase each individual sector instead. Clear to have the flash programmer obey chip erase commands and erase all sectors at once. Check the All Sectors checkbox to enable this checkbox.
Status	Shows flash programmer progress information. Click the Details button to show more thorough progress information.
Erase button	Click to have the flash programmer erase the sectors that you specified. The Status reflects flash programmer progress.
Blank Check button	Click to have the flash programmer perform these tasks: <ul style="list-style-type: none"> • upload the sectors that you specified to the hardware device • compare the uploaded sectors against 0xFF • report the values that do not match 0xFF. The Status reflects flash programmer progress.

Checksum

The **Checksum** panel lets you calculate checksum values. [Figure 22.6 on page 296](#) shows the Checksum panel. [Table 22.6 on page 296](#) explains items in the panel.

Working with Hardware Tools

Flash Programmer Window

Figure 22.6 Checksum panel

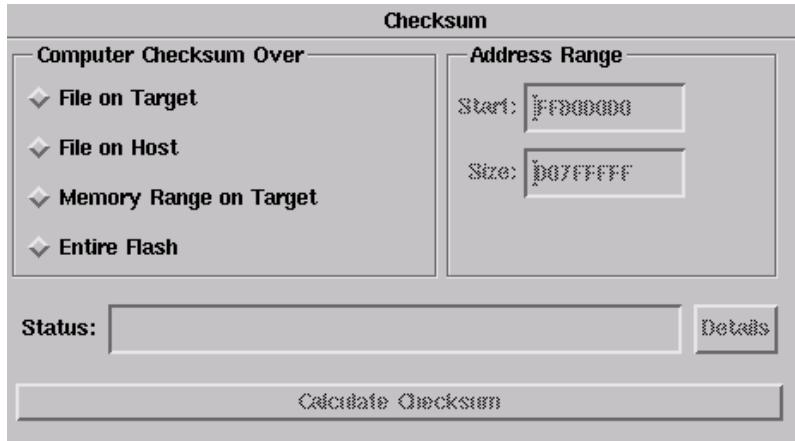


Table 22.6 Checksum panel—items

Item	Explanation
File on Target	Select to have the flash programmer read the file that you specified in the Use Selected File text box of the Program / Verify panel. The flash programmer reads this file to determine the required memory regions of the flash device for the checksum operation. The Restrict Address Range and Apply Address Offset information that you specify in the Program / Verify panel also apply to this option button.
File on Host	Select to have the flash programmer read the file on the host computer. The flash programmer reads this file to determine the required memory regions of the flash device for the checksum operation. The Restrict Address Range and Apply Address Offset information that you specify in the Program / Verify panel also apply to this option button.
Memory Range on Target	Select to have the flash programmer read the range that you specify in the Start and Size values in the Address Range group. The flash programmer uses this memory range for the checksum operation.
Entire Flash	Select to have the flash programmer read the entire contents of flash memory. The flash programmer uses this data for the checksum operation.

Table 22.6 Checksum panel—items (*continued*)

Item	Explanation
Start text box	Enter the starting address of the range that you want the flash programmer to use for the checksum operation. Select Memory Range on Target option to enable this text box.
Size text box	Enter the size of the address range that you want the flash programmer to use for the checksum operation. This size is relative to the starting address that you specify in the Start text box. Select Memory Range on Target option to enable this text box.
Status	Shows flash programmer progress information. Click the Details button to show more thorough progress information.
Calculate Checksum	Click to have the flash programmer calculate the checksum according to your specifications. At the end of the checksum operation, the Status shows the calculated checksum.

Hardware Diagnostics Window

The **Hardware Diagnostics** window lists global options for the hardware diagnostic tools. These preferences apply to every open project file.

[Figure 22.7 on page 298](#) shows the Hardware Diagnostics window. [Table 22.7 on page 298](#) explains items in the window.

To open the Hardware Diagnostics window, click **Tools > Hardware Diagnostics**.

The Hardware Diagnostics window has these panels:

- [Configuration](#)
- [Memory Read / Write](#)
- [Scope Loop](#)
- [Memory Tests](#)

Working with Hardware Tools

Hardware Diagnostics Window

Figure 22.7 Hardware Diagnostics window

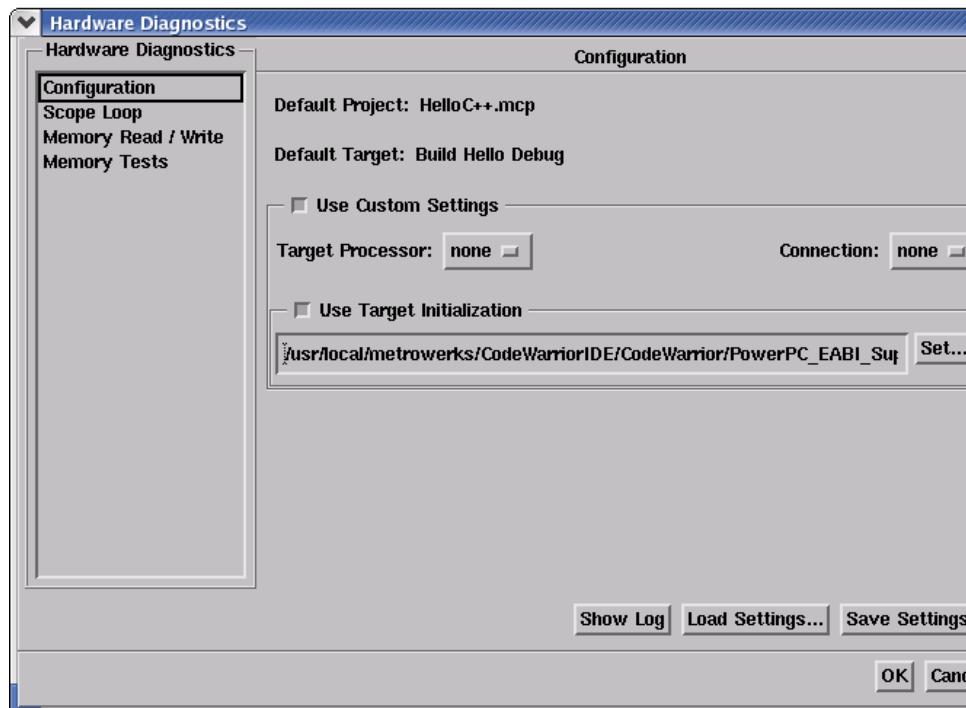


Table 22.7 Hardware Diagnostics window—items

Item	Explanation
Hardware Diagnostics pane	Shows a list of panel names. Click a panel name to display that panel in the Hardware Diagnostics window.
Load Settings	Click to restore previously saved settings for the current panel.
Save Settings	Click to save settings for the current panel to a file.
OK button	Click to save changes to all panels and close the window.
Cancel button	Click to discard changes to all panels and close the window.

Configuration

The Configuration panel configures general flash programmer settings. [Figure 22.8](#) shows the Configuration panel. [Table 22.8](#) explains items in the panel.

Figure 22.8 Configuration panel

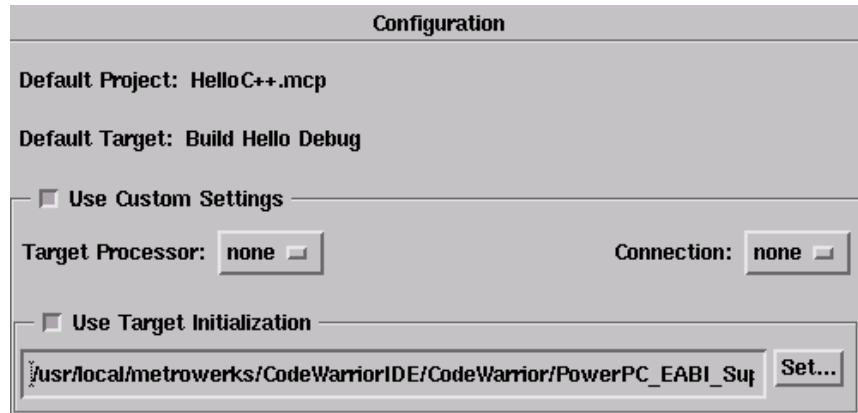


Table 22.8 Configuration panel—items

Item	Explanation
Default Project	Shows the current default project in the IDE.
Default Target	Shows the default build target in the IDE. Clear the Use Custom Settings checkbox to have the IDE use the connection settings from the build target for diagnosing the hardware.
Use Custom Settings checkbox.	Check to specify the connection information that you want to use for diagnosing the hardware. In this case, the IDE can connect to the hardware without using settings from a project. Clear to use the connection information stored in the default project for connecting to the hardware. You cannot clear the checkbox if you do not have an active default project or default target. Connection information includes information that you specify in the Target Processor list box, the Connection list box, and the Use Target Initialization text box.

Working with Hardware Tools

Hardware Diagnostics Window

Table 22.8 Configuration panel—items (*continued*)

Item	Explanation
Target Processor text/list box	Use to specify the hardware processor.
Connection list box	Use to specify the method that the IDE uses to connect to the hardware.
Use Target Initialization checkbox and text box	Check to specify an initialization file for the hardware connection. Enter the initialization file path in the text box, or click the Browse button to locate the initialization file path. Clear if you do not want to use an initialization file for the hardware connection.

Memory Read / Write

The **Memory Read / Write** panel configures diagnostic tests for performing memory reads and writes over the remote connection interface. [Figure 22.9 on page 300](#) shows the Memory Read / Write panel. [Table 22.9 on page 301](#) explains items in the panel.

Figure 22.9 Memory Read / Write panel

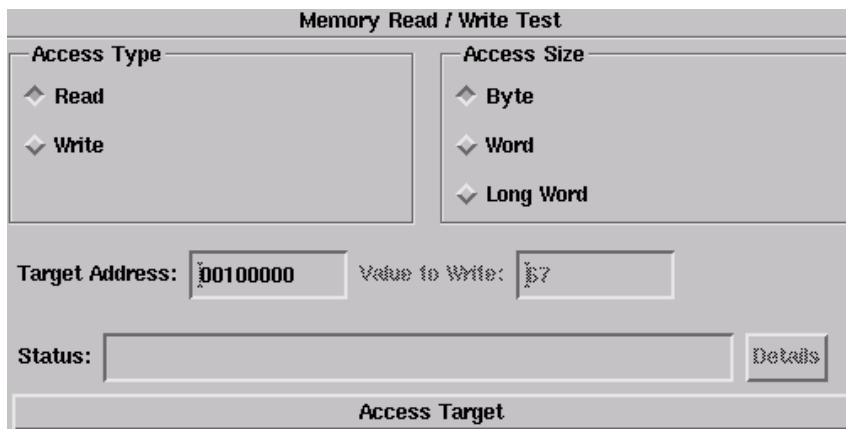


Table 22.9 Memory Read / Write panel—items

Item	Explanation
Read	Select to have the hardware diagnostic tools perform read tests.
Write	Select to have the hardware diagnostic tools perform write tests.
Byte	Select to have the hardware diagnostic tools perform byte-size operations.
Word	Select to have the hardware diagnostic tools perform word-size operations.
Long Word	Select to have the hardware diagnostic tools perform long-word-size operations.
Target Address	Specify the address of an area in RAM that the hardware diagnostic tools should analyze. The tools must be able to access this starting address through the remote connection (after the hardware initializes).
Value to Write	Specify the value that the hardware diagnostic tools write during testing. Select the Write option to enable this text box.
Status	Shows hardware diagnostic progress information. Click the Details button to show more progress information.
Access Target button	Click to have the hardware diagnostic tools perform specified tests. The Status shows test results.

Scope Loop

The **Scope Loop** panel configures diagnostic tests for performing repeated memory reads and writes over the remote connection interface. The tests repeat until you stop them. By performing repeated read and write operations, you can use a scope analyzer or logic analyzer to debug the hardware device.

[Figure 22.10 on page 302](#) shows the Scope Loop panel. [Table 22.10 on page 302](#) explains items in the panel.

After the first 1000 operations, the **Status** shows the estimated time between operations.

Working with Hardware Tools

Hardware Diagnostics Window

NOTE For all values of **Speed**, the time between operations depends heavily on the processing speed of the host computer.

For **Read** operations, the Scope Loop test has an additional feature. During the first read operation, the hardware diagnostic tools store the value read from the hardware. For all successive read operations, the hardware diagnostic tools compare the read value to the stored value from the first read operation. If the Scope Loop test determines that the value read from the hardware is not stable, the diagnostic tools report the number of times that the read value differs from the first read value.

Figure 22.10 Scope Loop panel

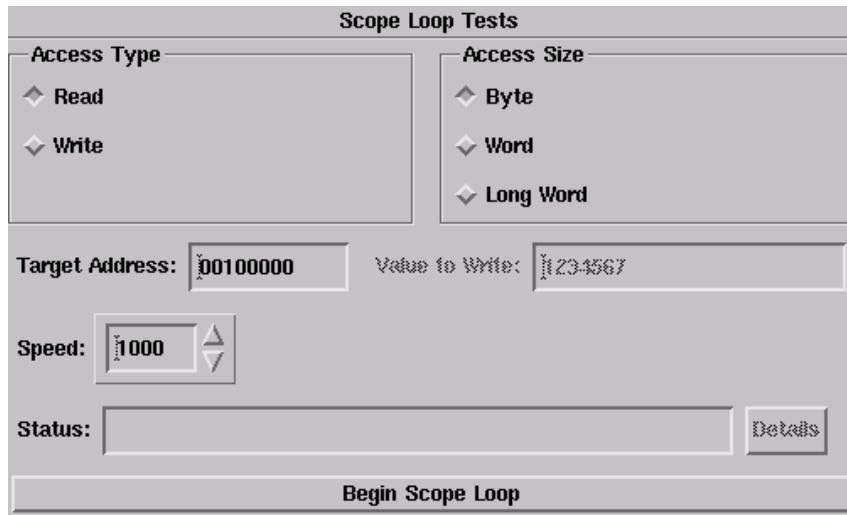


Table 22.10 Scope Loop panel—items

Item	Explanation
Read	Select to have the hardware diagnostic tools perform read tests.
Write	Select to perform write tests.
Byte	Select to have the hardware diagnostic tools perform byte-size operations.

Table 22.10 Scope Loop panel—items (*continued*)

Item	Explanation
Word	Select to perform word-size operations.
Long Word	Select to perform long-word-size operations.
Target Address	Specify the address of an area in RAM that the hardware diagnostic tools should analyze. The tools must be able to access this starting address through the remote connection (after the hardware initializes).
Value to Write	Specify the value that the hardware diagnostic tools write during testing. Select the Write option to enable this text box.
Speed slider	Move to adjust the speed at which the hardware diagnostic tools repeat successive read and write operations. Lower speeds increase the delay between successive operations. Higher speeds decrease the delay between successive operations.
Status	Shows hardware diagnostic progress information. Click the Details button to show more thorough progress information.
Begin Scope Loop button	Click to have the hardware diagnostic tools perform your specified tests. The Status shows test results.

Memory Tests

The **Memory Tests** panel lets you perform three different tests on the hardware:

- [Walking Ones](#)
- [Address](#)
- [Bus Noise](#)

[Figure 22.11 on page 304](#) shows the Memory Tests panel. [Table 22.11 on page 304](#) explains items in the panel.

You can specify any combination of tests and number of passes to perform. For each pass, the hardware diagnostic tools perform the tests in turn, until all passes are complete. The tools tally memory test failures and display them in a log window after all passes are complete. Errors resulting from memory test failures do not stop the testing process, however, fatal errors immediately stop the testing process.

Working with Hardware Tools

Hardware Diagnostics Window

Figure 22.11 Memory Tests panel

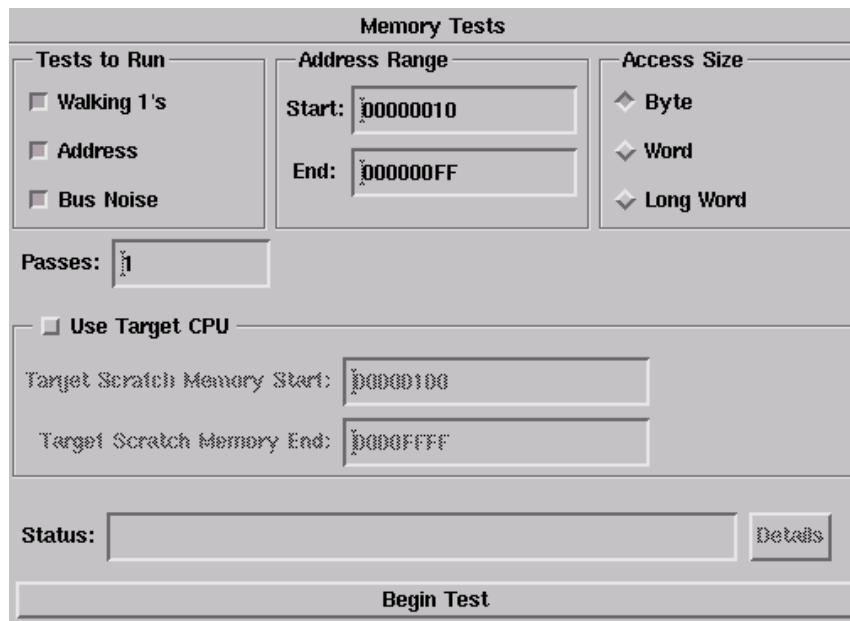


Table 22.11 Memory Tests panel—items

Item	Explanation
Walking 1's	Check to have the hardware diagnostic tools perform the Walking Ones test. Clear to have the diagnostic tools skip the Walking Ones test.
Address	Check to have the hardware diagnostic tools perform the Address test. Clear to have the diagnostic tools skip the Address test.
Bus Noise	Check to have the hardware diagnostic tools perform the Bus Noise test. Clear to have the diagnostic tools skip the Bus Noise test.
Start:	Enter the starting address of the range that you want to test.
End:	Enter the ending address of the range that you want to test.
Byte	Select to have the hardware diagnostic tools perform byte-size operations.

Table 22.11 Memory Tests panel—items (*continued*)

Item	Explanation
Word	Select to have the hardware diagnostic tools perform word-size operations.
Long Word	Select to have the hardware diagnostic tools perform long-word-size operations.
Passes	Enter the number of times that you want to repeat the specified tests.
Show Log	Click to display a text file that logs memory test actions.
Use Target CPU	<p>Check to have the hardware diagnostic tools download the test code to the hardware device. Enter in the Target Scratch Memory Start and Target Scratch Memory End text boxes the memory range that you want to use on the hardware device. The CPU on the hardware device executes the test code in this memory range.</p> <p>Clear to have the hardware diagnostic tools execute the test code through the remote connection interface.</p> <p>Execution performance improves greatly if you execute the test code on the hardware CPU, but requires that the hardware has enough stability and robustness to execute the test code.</p>
Target Scratch Memory Start	Specify the starting address of an area in RAM that the hardware diagnostic tools can use as a scratch area. The tools must be able to access this starting address through the remote connection (after the hardware initializes).
Target Scratch Memory End	Specify the ending address of an area in RAM that the hardware diagnostic tools can use as a scratch area. The tools must be able to access this address through the remote connection (after the hardware initializes).
Status	Shows memory test progress information. Click the Details button to show more thorough progress information.
Begin Test	Click to have the hardware diagnostic tools perform the memory tests that you specified. The Status reflects memory test progress.

Walking Ones

This test detects these memory faults:

- Address Line—The board or chip address lines are shorting or stuck at 0 or 1. Either condition could result in errors when the hardware reads and writes to the memory location. Because this error occurs on an address line, the data may end up in the wrong location on a write operation, or the hardware may access the wrong data on a read operation.
- Data Line—The board or chip data lines are shorting or stuck at 0 or 1. Either condition could result in corrupted values as the hardware transfers data to or from memory.
- Retention—The contents of a memory location change over time. The effect is that the memory fails to retain its contents over time.

The Walking Ones test includes four subtests:

- Walking Ones—This subtest first initializes memory to all zeros. Then the subtest writes, reads, and verifies bits, with each bit successively set from the least significant bit (LSB) to the most significant bit (MSB). The subtest configures bits such that by the time it sets the MSB, all bits set to a value of 1. This pattern repeats for each location within the memory range that you specify. For example, the values for a byte-based Walking Ones subtest occur in this order:

0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3F, 0x7F, 0xFF
- Ones Retention—This subtest immediately follows the Walking Ones subtest. The Walking Ones subtest should leave each memory location with all bits set to 1. The Ones Retention subtest verifies that each location has all bits set to 1.
- Walking Zeros—This subtest first initializes memory to all ones. Then the subtest writes, reads, and verifies bits, with each bit successively set from the LSB to the MSB. The subtest configures bits such that by the time it sets the MSB, all bits are set to a value of 0. This pattern repeats for each location within the memory range that you specify. For example, the values for a byte-based Walking Zeros subtest occur in this order:

0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x00
- Zeros Retention—This subtest immediately follows the Walking Zeros subtest. The Walking Zeros subtest should leave each memory location with

all bits set to 0. The Zeros Retention subtest verifies that each location has all bits set to 0.

Address

This test detects memory aliasing. *Memory aliasing* exists when a physical memory block repeats one or more times in a logical memory space. Without knowing about this condition, you might conclude that there is much more physical memory than what actually exists.

The address test uses a simplistic technique to detect memory aliasing. The test writes sequentially increasing data values (starting at one and increasing by one) to each successive memory location. The maximum data value is a prime number and its specific value depends on the addressing mode so as to not overflow the memory location.

The test uses a prime number of elements to avoid coinciding with binary math boundaries:

- For byte mode, the maximum prime number is 2^8 -5 or 251.
- For word mode, the maximum prime number is 2^{16} -15 or 65521.
- For long word mode, the maximum prime number is 2^{32} -5 or 4294967291.

If the test reaches the maximum value, the value rolls over to 1 and starts incrementing again. This sequential pattern repeats throughout the memory under test. Then the test reads back the resulting memory and verifies it against the written patterns. Any deviation from the written order could indicate a memory aliasing condition.

Bus Noise

This test stresses the memory system by causing many bits to flip from one memory access to the next (both addresses and data values). *Bus noise* occurs when many bits change consecutively from one memory access to another. This condition can occur on both address and data lines.

Address lines

To force bit flips in address lines, the test uses three approaches:

- Sequential—This approach works sequentially through all of the memory under test, from lowest address to highest address. This sequential approach results in an average number of bit flips from one access to the next.

Working with Hardware Tools

Hardware Diagnostics Window

- Full Range Converging—This approach works from the fringes of the memory range toward the middle of the memory range. Memory access proceeds in this pattern, where *+ number* and *- number* refer to the next item location (the specific increment or decrement depends on byte, word, or long word address mode):
 - the lowest address
 - the highest address
 - (the lowest address) + 1
 - (the highest address) - 1
 - (the lowest address) + 2
 - (the highest address) - 2
- Maximum Invert Convergence—This approach uses calculated end point addresses to maximize the number of bits flipping from one access to the next. This approach involves identifying address end points such that the values have the maximum inverted bits relative to one another. Specifically, the test identifies the lowest address with all 0x5 values in the least significant nibbles and the highest address with all 0xA values in the least significant nibbles. After the test identifies these end points, memory access alternates between low address and high address, working towards the center of the memory under test. Accessing memory in this manner, the test achieves the maximum number of bits flips from one access to the next.

Data lines

To force bit flips in data lines, the test uses two sets of static data, a pseudo-random set and a fixed-pattern set. Each set contains 31 elements—a prime number. The test uses a prime number of elements to avoid coinciding with binary math boundaries. The sets are unique to each addressing mode so as to occupy the full range of bits.

- The test uses the pseudo-random data set to stress the data lines in a repeatable but pattern-less fashion.
- The test uses the fixed-pattern set to force significant numbers of data bits to flip from one access to the next.

The subtests execute similarly in that each subtest iterates through static data, writing values to memory. The test combines the three address line approaches with the two data sets to produce six unique subtests:

- Sequential with Random Data

- Sequential with Fixed Pattern Data
- Full Range Converging with Random Data
- Full Range Converging with Fixed Pattern Data
- Maximum Invert Convergence with Random Data
- Maximum Invert Convergence with Fixed Pattern Data

Working with a Logic Analyzer

(Windows OS) This section explains how to set up your project to connect to a logic analyzer and how to use the IDE to issue commands to the logic analyzer. For more information about setting up the logic analyzer to transmit information to the IDE, refer to the *Targeting* documentation.

Configuring the Project

Use the **Analyzer Connections** target settings panel ([Figure 22.12](#)) to configure your project to connect to a logic analyzer.

Use the **Connection** list box to specify the logic analyzer connection that you want to use. Click the **Edit Connection** button to configure the parameters of the connection. [Figure 22.13 on page 311](#) shows the window that appears when you click the Edit Connection button. [Table 22.12 on page 311](#) explains options in this window.

NOTE Each build target supports only one connection to a logic analyzer. If you want your project to have more logic analyzer connections, create a build target for each additional connection.

Working with Hardware Tools

Working with a Logic Analyzer

Figure 22.12 Analyzer Connections settings panel

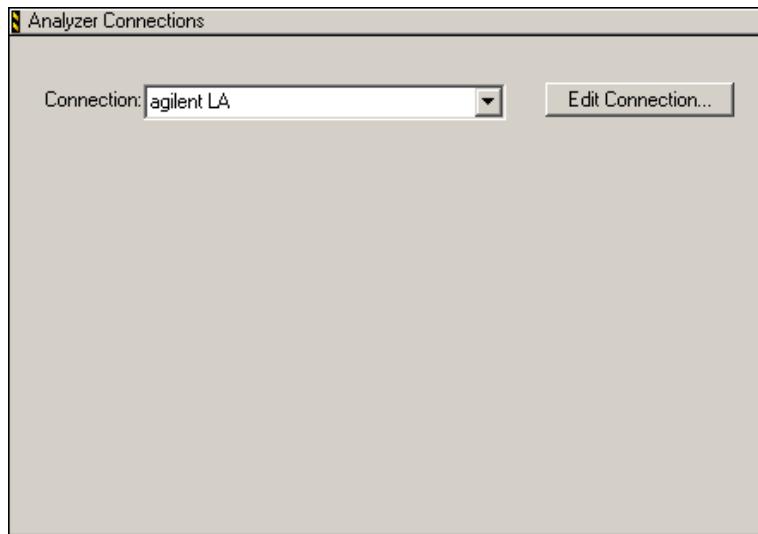


Figure 22.13 Editing a logic analyzer connection

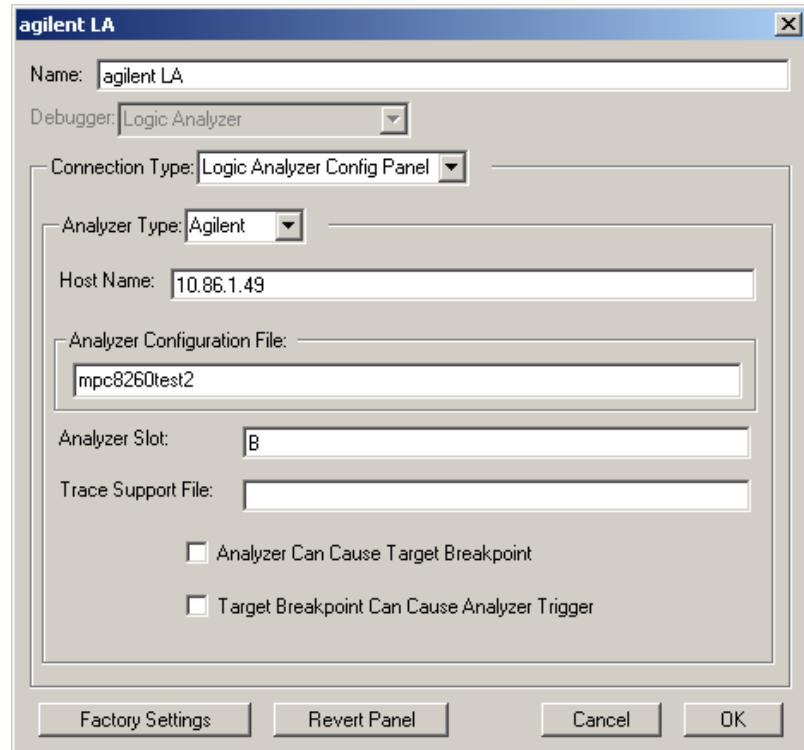


Table 22.12 Logic analyzer connection options

Option	Explanation
Name	Enter the name that you want to assign to this collection of options.
Debugger	Use to specify the debugger to use with the logic analyzer.
Connection Type	Use to specify the connection method to the logic analyzer.
Analyzer Type	Use to specify the type of logic analyzer.
Host Name	Enter the Internet Protocol (IP) address of the logic analyzer.
Analyzer Configuration File	Enter the name of the configuration file that the logic analyzer requires.

Working with Hardware Tools

Working with a Logic Analyzer

Table 22.12 Logic analyzer connection options (*continued*)

Option	Explanation
Analyzer Slot	Enter the slot name that identifies the logic analyzer location.
Trace Support File	Enter the name of the file that the logic analyzer requires to support the collection of trace data.
Analyzer Can Cause Target Breakpoint	Check to allow the logic analyzer to cause a hardware breakpoint. Clear to prevent the logic analyzer from causing a hardware breakpoint.
Target Breakpoint Can Cause Analyzer Trigger	Check to allow a hardware breakpoint to trigger the logic analyzer. Clear to prevent a hardware breakpoint from triggering the logic analyzer.

Using the Logic Analyzer

The IDE can work with a logic analyzer in these ways:

- [Connect](#)—establish a connection to the logic analyzer
- [Arm](#)—enable the logic analyzer to collect trace data
- [Disarm](#)—disable the logic analyzer from collecting trace data
- [Update Data](#)—retrieve the latest data from the logic analyzer
- [Disconnect](#)—close connection to the logic analyzer

Before you can use the IDE to work with a logic analyzer, you must use the **Analyzer Settings** target settings panel to configure a connection to the logic analyzer.

The IDE requires this information in order to correlate collected trace data with currently running source code.

Connect

This command uses the connection options that you specified in the **Analyzer Settings** target settings panel to perform these tasks:

1. Establish a connection to the logic analyzer.
2. Load the configuration file that you specified in the **Analyzer Configuration File** text box (the load process might take several minutes).

3. Requests additional information from you as required (for example, for an Agilent connection, the IDE asks you to select the machine that you want to use).
4. Retrieves all data that will appear in the Trace window.

Click **Tools > Logic Analyzer > Connect** to use this command. You cannot use this command if you are already connected to a logic analyzer.

Arm

This command instructs the logic analyzer to perform a Run All operation. This operation prepares the logic analyzer to collect trace data. Click **Tools > Logic Analyzer > Arm** to use this command. You cannot use this command if the IDE has not established a connection to the logic analyzer, or if you already armed the logic analyzer.

Disarm

This command instructs the logic analyzer to perform a Stop All operation, if it is still running. This operation stops the logic analyzer from collecting trace data. Click **Tools > Logic Analyzer > Disarm** to use this command. You cannot use this command if the IDE has not established a connection to the logic analyzer.

NOTE You must use the Disarm command in order to update trace data in the IDE.

Update Data

This command retrieves the most recent trace data from the logic analyzer in order to display that data in the Trace window of the IDE. Click **Tools > Logic Analyzer > Update Data** to use this command. The Trace window flushes its previous data and updates its display with the newly retrieved trace data. You cannot use this command until you first Disarm the logic analyzer.

NOTE The Update Data command does not update the column labels in the Trace window. If you change the labels in the logic analyzer, you must disconnect from it in the IDE and then reconnect to it. After you disconnect and reconnect, the Trace window reflects your changes to the column labels.

Disconnect

This command disconnects the IDE from the logic analyzer, if the connection still exists. Click **Tools > Logic Analyzer > Disconnect** to use this command. After you disconnect the IDE from the logic analyzer, the Trace window flushes its data. You cannot use this command if you are not currently connected to a logic analyzer.

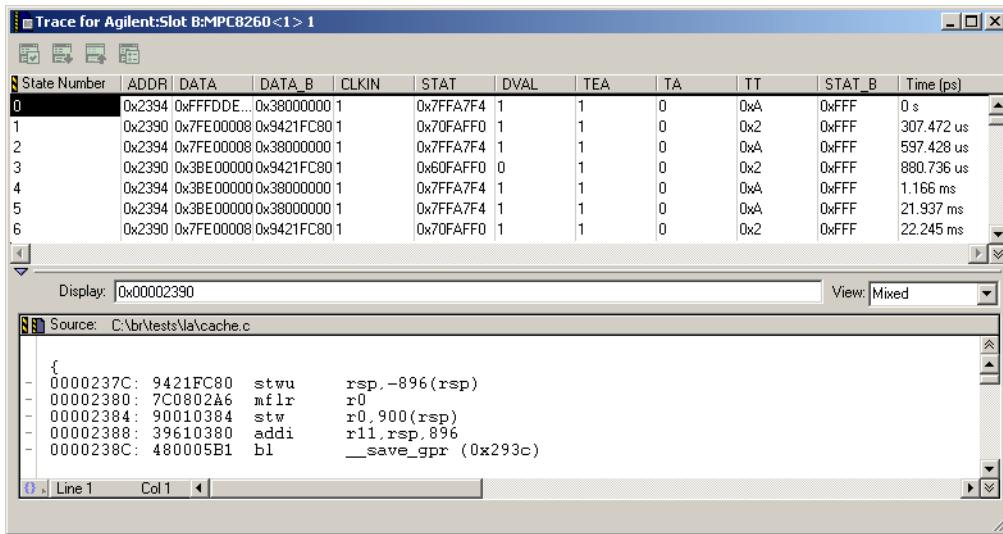
Trace Window

After you configure your project to use a logic analyzer and collect trace data, you use the **Trace** window ([Figure 22.14](#)) to view the collected data. The trace window shows up to 100,000 states or trace frames, beginning with the most recent frame.

The IDE determines the column labels that appear in the Trace window at the time it connects to the logic analyzer. If you update these labels in the logic analyzer, your changes do not appear in the Trace window if you update data. In the IDE, you must disconnect from the logic analyzer and reconnect to it in order to update the column labels in the Trace window.

After you use a logic analyzer to collect trace data, open the Trace window by clicking **Data > View Trace**.

Figure 22.14 Trace window



Cache Window

Use the **Cache** window ([Figure 22.15](#)) to view cache information for the target processor. Click **Data > View Cache** to open the Cache window.

NOTE The **View Cache** menu might have additional supported commands, depending on the target processor. For example, you might be able to click **Data > View Cache > Instruction Cache** or **Data > View Cache > Data Cache** to view these two types of cache concurrently.

Working with Hardware Tools

Profile Window

Figure 22.15 Cache window

The screenshot shows a software interface titled "PQ2 L1 Data cache". It features a toolbar with icons for file operations like Open, Save, and Print. Below the toolbar is a table with two sections: a header row and a data section. The header row contains columns for Line, Way, Address, Valid, Dirty, and Lock. The data section contains 128 rows of memory dump data, each with a corresponding line number from 0 to 127. The address column shows memory addresses starting from 007effe00. The valid, dirty, and lock columns show binary values (No or Yes). The data column displays memory contents as a series of FFFF characters. A scroll bar is visible on the right side of the data area. At the bottom of the window, there are buttons for Refresh and Write, and a "View As: Raw data" dropdown menu.

Line	Way	Address	Valid	Dirty	Lock	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
112	0	007effe00	No	No	No	7FFFFFFF	7FFFFFFF	7FFFFFFF	7FFFFFFF	00000000	00000000	00000000	7FFFFFFF
113	0	007effe20	No	No	No	7FFFFFFF							
114	0	007effe40	Yes	Yes	No	00000000	00000000	00000000	007EFE50	FFFFFFFFFF	FFFFFFFFFF	FFFFFFFFFF	FFFFFFFFFF
115	0	007effe60	Yes	Yes	No	FFFFFFFFFF							
116	0	007effe80	Yes	Yes	No	FFFFFFFFFF							
117	0	007effea0	Yes	Yes	No	FFFFFFFFFF							
118	0	007effec0	Yes	Yes	No	FFFFFFFFFF							
119	0	007effeed0	Yes	Yes	No	FFFFFFFFFF							
120	0	007efff00	Yes	Yes	No	FFFFFFFFFF	FFFFFFFFFF	FFFFFFFFFF	FFFFFFFFFF	FFFFFFFFFF	00000000	00000000	00000000
121	0	007efff20	No	No	No	7FFFFFFF							
122	0	007efff40	No	No	No	7FFFFFFF							
123	0	007efff60	No	No	No	7FFFFFFF							
124	0	007efff80	No	No	No	7FFFFFFF							
125	0	007efffa0	No	No	No	7FFFFFFF	7FFFFFFF	00000000	00000000	00000000	00000000	00000000	007EFC88
126	0	007efffc0	No	No	No	007EFC88	007EFE18	007EFE18	007EFE18	007E0EC0	007E1060	007E1200	00000190
127	0	007efffe0	No	No	No	00000194	007FFFF0	008FFFF0	009FFFF0	0080017C	00000064	FFFFFFFFFF	0000208C
0	1	00800000	No	No	No	7FFFFFFF							
1	1	00800020	No	No	No	7FFFFFFF							
2	1	00800040	No	No	No	7FFFFFFF							
3	1	00800060	No	No	No	7FFFFFFF							

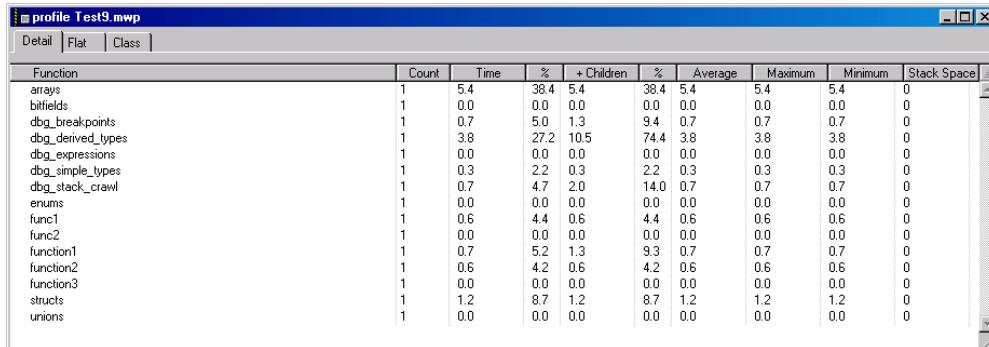
Profile Window

Use the Profile window ([Figure 22.16 on page 317](#)) to examine profile data that you collect from executing code. Examining this data helps you improve the performance of your project. You use profiler Application Programming Interface (API) calls or #pragma directives in your source code to turn on the profiler, collect profiling data, and turn off the profiler. For more information, refer to [“Profiler” on page 319](#).

NOTE The Profiler is only available if the target supports it. This feature is dependent upon support by the target-specific compiler and a profiling library.

To open the Profile window, add the appropriate API calls or #pragma directives to your source code, then debug your project. The Profile window opens automatically.

Figure 22.16 Profile window



The screenshot shows a Windows-style window titled "profile Test9.mwp". The window has tabs at the top: "Detail" (selected), "Flat", and "Class". The main area is a table with the following data:

Function	Count	Time	%	+ Children	%	Average	Maximum	Minimum	Stack Space
arrays	1	5.4	38.4	5.4	38.4	5.4	5.4	0	0
bitfields	1	0.0	0.0	0.0	0.0	0.0	0.0	0	0
dbg_breakpoints	1	0.7	5.0	1.3	9.4	0.7	0.7	0.7	0
dbg_derived_types	1	3.8	27.2	10.5	74.4	3.8	3.8	3.8	0
dbg_expressions	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
dbg_simple_types	1	0.3	2.2	0.3	2.2	0.3	0.3	0.3	0
dbg_stack_crawl	1	0.7	4.7	2.0	14.0	0.7	0.7	0.7	0
enums	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
func1	1	0.6	4.4	0.6	4.4	0.6	0.6	0.6	0
func2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
function1	1	0.7	5.2	1.3	9.3	0.7	0.7	0.7	0
function2	1	0.6	4.2	0.6	4.2	0.6	0.6	0.6	0
function3	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
structs	1	1.2	8.7	1.2	8.7	1.2	1.2	1.2	0
unions	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

Command Window

The IDE supports a command-line interface to some of its features. You can use the command-line interface together with the TCL scripting engine. You can also issue a command line that saves a log file of command-line activity.

The **Command** window in the IDE shows the standard output and standard error streams of command-line activity. [Figure 22.17 on page 317](#) shows the Command window.

Figure 22.17 Command window



Working with Hardware Tools

Command Window

Opening the Command Window

Use the **Command** window to view the standard output and standard error streams of command-line activity.

To open the Command window, click **View > Command Window**.

Issuing Command Lines

Use the **Command** window to issue command lines to the IDE. For example, enter `debug` to start a debugging session.

To issue a command line, bring forward the Command window, type the command line, and press Enter or Return. The IDE executes the command line that you entered.

If you work with hardware as part of your project, you can use the Command window to issue command lines to the IDE while the hardware is running.

NOTE Enter `help` to see a list of available commands and a brief explanation of each command. Enter `help command` to see a detailed explanation of the *command*. Detailed information is provided in the IDE Automation Guide.

Profiler

The CodeWarrior active profiler lets you analyze how processor time is distributed during your program's execution. With this information, you can determine where to concentrate your efforts to optimize your code most effectively.

This chapter contains these sections:

- [“Overview” on page 319](#)
- [“Using the Profiler” on page 323](#)
- [“Configuring” on page 329](#)
- [“Viewing Results” on page 335](#)
- [“Troubleshooting” on page 341](#)
- [“Reference” on page 343](#)

Overview

This section provides you with general information about what a profiler is, different kinds of profilers, and a typical strategy you would follow to measure program performance.

Topics discussed are:

- [What Is a Profiler?](#)—a brief description of profilers and what they do
- [Types of Profilers](#)—different kinds of profilers, their strengths and weaknesses
- [A Profiling Strategy](#)—an outline you should follow when profiling your own code
- [Profiling Code](#)—three steps to follow when profiling code

What Is a Profiler?

Speed and performance are important issues in most software projects. In most cases, code that does not work quickly does not work well.

Programmers have regularly observed that 10% of their code does 90% of the work. Reworking code to make it more efficient is not a trivial task. You should concentrate on improving that core 10% of your code first, and improve the infrequently-used code later, if at all.

How would you like to know precisely where your code spent its time? That is what a profiler does for you—it gives you clues. More than clues, the CodeWarrior profiler gives you hard and reliable data.

A good profiler analyzes the amount of time your code spends performing various tasks. Armed with this information, you can apply your efforts to improving the efficiency of core routines.

A profiler can also help you detect bottlenecks—routines your data passes through to get to other places—and routines that are just inordinately slow. Identifying these problems is the first step to solving them.

Types of Profilers

The simplest profilers count how many times a routine is called. They do not report any information about which routines are called by other routines, or the amount of time spent inside the various routines being profiled.

Clearly a good profile of the runtime performance of code requires more information than a raw count. More advanced profilers perform statistical sampling of the runtime environment. These profilers are called passive or sampling profilers.

A passive profiler divides the program being profiled into evenly-sized “buckets” in memory. It then samples the processor’s program counter at regular intervals to determine which bucket the counter is in.

The main advantage of a passive profiler is that it requires no modification to the program under observation. You just run the profiler and tell it what program to observe. Also, passive profilers distribute the overhead that they incur evenly over time, allowing the post-processing steps to ignore it. On the other hand, they cannot sample too frequently or the sampling interrupt will overwhelm the program being sampled.

Passive profilers have a significant disadvantage. Although useful, bucket boundaries do not line up with routine boundaries in the program. This makes it difficult if not impossible to determine which *routines* are heavily used. As a result, passive profilers generate a relatively low-resolution image of what is happening in the program while it runs.

In addition, because they rely on a statistical sampling technique, the program must run for a long enough period to collect a valid sample. As a result, they do not have good repeatability—that is, the results you get from different runs may vary unless the sampling period is long.

The most advanced and accurate profilers are called active profilers. The CodeWarrior profiler is an active profiler.

An active profiler tracks the precise amount of time a program spends in each individual routine, measured directly from the system clock.

To perform this magic, an active profiler requires that you modify the code of the program to be observed. An active profiler gains control at every routine entry and exit. There must be a call to the profiler at the beginning of each profiled routine. The profiler can then track how much time is spent in the routine.

This approach has significant advantages over a passive profiler. An active profiler can report high-resolution results about exactly what your program is doing. An active profiler also tracks the dynamic call tree of a program. This information can be very useful for determining the true cost of calling a routine. The true cost of a routine call is not only the time spent in the routine, it is also the time spent in its children—the subsidiary routines it calls, the routines they call, and so on to whatever depth is necessary.

Because it uses measurements and not statistical sampling, an active profiler is much more accurate and repeatable than a passive profiler.

The requirement that you must modify the actual source code might seem like a significant disadvantage. With the CodeWarrior profiler, this disadvantage is minimal. Activating the profiler for an entire program—or for a range of routines within a program—is simple. The compiler does most of the work, inserting the necessary calls to the profiler itself. You do have to recompile the project when you turn on profiling.

Finally, active profilers generate a large amount of raw information. This can lead to confusion and difficulty interpreting the results. The Profiler window that is part of the CodeWarrior profiler system handles these difficulties with aplomb. You can view and sort the data in whatever way best suits your needs.

A Profiling Strategy

You use a profiler to measure the runtime performance of your code. What is usually important is how your code's performance measures up to some standard. When approaching the problem of measuring performance, you might want to take these three steps:

1. Establish your standards.

For example, you might decide that you want the program to load in less than ten seconds, or check the spelling of a five-page document that contains no misspellings in 15 seconds. Also decide on the platform you will use for testing, since processor speeds vary.

2. Determine how to measure time.

Your measurement device may be no more complicated than a stopwatch, or you may need to add some simple code to count ticks. At this phase you want to test the code in as close to its finished form as possible, so measure time in a way that is accurate enough to suit your needs, and that has the lowest impact on your code's natural performance. You do not want to run a full-blown profile here, because profiling can add significant overhead, thus slowing down your code's raw performance.

3. Run the tests and measure results.

If you meet your performance goals, your job is done. If your code does not meet your goals, then it is time to profile your code.

Profiling Code

To profile your code, you do three things:

1. Run a profiler on the area of the code you want tested.

This might be a single routine, a group of routines that perform a task, or even the entire application. What you profile depends upon what you are testing.

2. Analyze the data collected by the profiler and improve your code.

You study the results of your profiling and look for problems and room for improvement.

The profiling process is iterative. You repeat these two steps until you achieve the performance gain you need to meet your goals.

The rest of this manual discusses how to perform these two steps—profile your code and analyze the results—using the CodeWarrior profiler system.

3. Retest your code to verify results

When you are satisfied that you have reached your goals, you have one more step to perform. You should run your original tests—without the profiler of course—to verify that your code in its natural state meets your performance goals.

The CodeWarrior profiler will help you meet those goals quickly and easily.

Using the Profiler

The CodeWarrior profiler lets you analyze how processor time is distributed during your program’s execution. With this information, you can determine where to concentrate your efforts to optimize your code most effectively.

This section discusses the following principal topics:

- [What It Does](#)—an overview of the principle features of the profiler
- [How It Works](#)—basic information on the elements of the profiler and about how to use the profiler in your own code
- [Profiling Made Easy](#)—a step-by-step guide to using the profiler

What It Does

The CodeWarrior profiler is a state-of-the-art, user-friendly, analytical tool that can profile C or C++ code.

For every project, from the simplest to the most complex, the profiler offers many useful features that help you analyze your code. You can:

- turn the profiler on and off at compile time
- profile any routine, group of routines, or an entire project
- track time spent in any routine

- track time spent in a routine and the routines it calls—its children
- track execution paths and times in a dynamic call tree
- collect detailed or summary data in a profile
- use precision time resolutions for accurate profiling
- track the stack space used by each routine

How It Works

The CodeWarrior profiler is an active profiler. The profiling system consists of three main profiler components:

- a statically-linked code library of compiled code containing the profiler
- an Application Programming Interface (API) to control the profiler
- the Profiler window to view and analyze the profile results

Details of the API are discussed in [Profiler Function Reference](#). The Profiler window is discussed in [Viewing Results](#).

The rest of this section will discuss the general profiling process. Subsequent sections describe how to carry out the profiling process for your particular target.

To use the profiler, you do these things:

- Include the correct profiler library and files in your CodeWarrior project
- Modify your source code to make use of the profiler API
- Use the API to initialize the profiler, to dump the results into a file, and to exit the profiler
- Use the Profiler window to view the results

You can profile an entire program if you wish or, adding compiler directives to your code, you can profile any individual section of your program.

You modify the original source code slightly to initialize the profiler, dump results, and exit the profiler when through. You may modify the source code more extensively if you wish to profile individual portions of your code.

Then the compiler and linker—using a profiler library—generate a new version of your program, ready for profiling. While it runs, the profiler generates data. Your program will run a little more slowly because of the profiler overhead (sometimes a *lot* more slowly), but that is taken into account in the final results. When complete, you use the Profiler window to analyze the data and determine what changes are appropriate to improve performance. You can repeat the

process as often as desired until you have turned your code into a fast, efficient, well-oiled machine.

See also

[Profiler Function Reference](#) and [Viewing Results](#)

Profiling Made Easy

This section takes you step by step through the general process of profiling an application.

To profile an application, you:

- [Add a profiler library to the project](#)
- [Turn on profiling](#)
- [Include the profiler API interface](#)
- [Initialize the profiler](#)
- [Dump the profile results](#)
- [Exit the profiler](#)

In the steps that follow, we detail precisely what to do in both C and C++. These steps may seem a little complicated. Do not be alarmed. Using the CodeWarrior profiler is actually easier than reading about how to do it.

1. Add a profiler library to the project

The code that performs the profiler magic has been compiled into libraries.

The precise library that you add to your code depends on the target for which you are profiling code and on the kind of code you are developing.

2. Turn on profiling

You can use the following methods to turn profiling on or off:

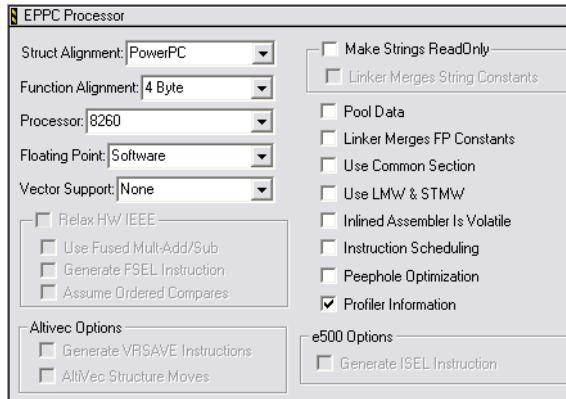
a. Project-Level Profiling

To turn on profiling for an entire project, use the project settings. In the Project Settings dialog, choose the processor you are generating code for under the Code Generation option. Check the **Profiler Information** checkbox, as shown in [Figure 23.1](#). With profiling on, the compiler generates all the code necessary so that every routine calls the profiler.

Profiler

Using the Profiler

Figure 23.1 Processor Preferences Options for PowerPC



b. Routine-Level Profiling

To profile certain routines (rather than the entire project), use the appropriate profiler API calls for your target to initialize the profiler, set up profiling, and immediately turn profiling off. You can then manually turn profiling on and off by placing profiler calls around the routine or routines you want to profile. For example, you could modify your code to look like [Listing 23.1](#).

Listing 23.1 profiling a Routine

```
void main()
{
    ...
    err = ProfilerInit(...);

    if (err == noErr)
    {
        ProfilerSetStatus(FALSE); // turn off profiling until needed.
        // more code.....

        // now you reach routine you want to profile
        ProfilerSetStatus(TRUE); // turn on profiling
        foobar(); // this routine is profiled and shows up in viewer
        ProfilerSetStatus(FALSE); // turn profiling off again

        // more code....
        ProfilerTerm();
    }
}
```

Assuming that profiling is on for an entire project, you can turn off profiling at any time. First, use an appropriate call to turn off profiling. Then use another call to turn it on. Turn it on just before calling the routine or routines you are interested in. Turn it off when those routines return. It is really that easy.

Alternatively, you can use `#pragma` statements in C/C++. These are not as useful as using profiler API calls. For example, suppose you have two routines—`foo()` and `bar()`—that each call a third utility routine, `barsoom()`. If you use compiler directives to turn on profiling for `foo()` and `barsoom()`, the result you get will include the time for `barsoom()` when called from `bar()` as well.

3. Include the profiler API interface

To use the profiler, you add at least three profiler-related calls to your code. These calls are detailed in the next three steps. The process varies slightly for the different languages and targets.

Source files that make calls to the profiler API must include the appropriate header file for your target. For example, to profile an entire application, you

Profiler

Using the Profiler

would add this line of code to the source file that includes your `main()` function:

```
#include <profiler.h>
```

TIP You do not have to include the header file in every file that contains a profiled function, only in those that actually make direct profiler API calls.

4. Initialize the profiler

At the beginning of your code, you call the appropriate function for your target. See [Profiler Function Reference](#) to find out the precise function name that you need for your specific target.

5. Dump the profile results

Obviously, if you profile code you want to see the results. The profiler dumps the results to a data file. The data is in a proprietary format understood by the profiler.

6. Exit the profiler

When you are all through with the profiler, before exiting the program you should terminate the profiler by calling the correct profiler API function. On most platforms, if you initialize the profiler and then exit the program without terminating the profiler, timers may be left running that could crash the machine.

The call to terminate the profiler stops the profiler and deallocates memory. It does not dump any information. Any collected data that has not been dumped is lost when you call the function to terminate the profiler.

Having performed these quick steps, you simply compile your program and run it. The IDE automatically opens this file in the Profiler window when the dump is complete. You can later re-open the file in the IDE to view the info again.

In summary, the process of using the CodeWarrior profiler is quite easy. You add the requisite library, turn on profiling, include the header file, initialize the profiler, dump the results, and exit. It is a remarkably painless and simple process that quickly gets you all the data you need to perform a professional-level analysis of your application's runtime behavior.

Configuring

This reference section discusses how to use the profiler libraries, APIs, and compiler options.

The sections in this section are:

- [Profiler Libraries and Interface Files](#)—the libraries and interface files that you add to your code in order to use the profiler
- [Profiling Special Cases](#)—special cases to consider when profiling code

Profiler Libraries and Interface Files

You can find all of the profiler libraries and interface files in the Profiler folder. The profiling code that actually keeps track of the time spent in a routine exists in a series of libraries. Depending upon the nature of your project and the platform for which you are writing code, you link in one or another of these libraries as appropriate. The libraries you use must match your settings in the **Target** settings panel.

The `profiler.h` file is the header file for the profiler API for C and C++. Include this file to make calls to control the profiler.

Profiling Special Cases

The profiler handles recursive and mutually recursive calls transparently. The profiler also warns you when profiling information was lost because of insufficient memory. (The profiler uses memory buffers to store profiling data.)

For leading-edge programmers, the profiler transparently handles and reliably reports the times for abnormally terminated routines exited through the C++ exception handling model (try, throw, catch) or the ANSI C library `setjmp()` and `longjmp()` routines.

This section describes special cases you may encounter while profiling your code:

- [Profiling Code with #pragma Statements](#)
- [Initializing Profiler with ProfilerInit\(\)](#)
- [Terminating Profiler with ProfilerDump\(\)](#)
- [Profiling Abnormally Terminated Functions](#)
- [Debugging Profiled Code](#)

Profiling Code with #pragma Statements

You can substitute #pragma statements for profiler API function calls to profile your C/C++ code on the function level. However, this is not as useful as the profiler calls. See “[Routine-Level Profiling](#)” on page 326 for more information.

Setting the “Generate Profiler Calls” Processor preference option sets a preprocessor variable named `__profile__` to 1. If profiling is off, the value is zero. You can use this value at compile time to test whether profiling is on.

Instead of, or in addition to, setting the option in the Processor preferences, you can turn on profiling at compile time. The C/C++ compiler supports three preprocessor directives that you can use to turn compiling on and off at will.

<code>#pragma profile on</code>	enables calls to the profiler in functions that are declared following the pragma
<code>#pragma profile off</code>	disables calls to the profiler in functions that are declared following the pragma
<code>#pragma profile reset</code>	sets the profile setting to the value selected in the preferences panel

You can use these directives to turn profiling on for any functions you want to profile, regardless of the settings in the Processor preferences. You can also turn off profiling for any function you do not want to profile.

Initializing Profiler with ProfilerInit()

At the beginning of your code, you call `ProfilerInit()` to initialize the profiler. [Table 23.1](#) shows the prototypes for `ProfilerInit()` for C/C++.

Table 23.1 ProfilerInit() Prototypes

C/C++	<pre>long ProfilerInit(ProfilerCollectionMethod method, ProfilerTimeBase timeBase, short numFunctions, short stackDepth);</pre>
-------	--------------------------------------------------------------------------------------------------------------------------------------------------

The parameters tell the profiler how this collection run is going to operate, and how much memory the profiler should allocate for its data buffers. Each parameter and its purpose is given in [Table 23.2](#).

Table 23.2 ProfilerInit() Parameters

Parameter	Purpose
<code>method</code>	collect detailed or summary data
<code>timeBase</code>	time scale to use in measurements
<code>numFunctions</code>	maximum number of routines to profile
<code>stackDepth</code>	approximate maximum depth of deepest calling tree

The collection method may be either `collectDetailed` or `collectSummary`. If you collect detailed data, you get information for the calling tree—the time in each routine and each of its children in the calling hierarchy. Summary data collects data for the time spent in each routine without regard to the calling chain. Collecting detailed data requires more memory.

The `timeBase` may be one of the following values:

- `ticksTimeBase`
- `microsecondsTimeBase`
- `timeMgrTimeBase`
- `PPCTimeBase`
- `win32TimeBase`
- `bestTimeBase`

The `bestTimeBase` option automatically selects the most precise timing mechanism available on the computer running the profiled software. Not all of these values are supported on all target platforms. Refer to the Targeting Manual for your product to determine which timebases are available for use.

The `numFunctions` parameter is the approximate number of routines to be profiled. The `stackDepth` parameter is the approximate maximum depth of your calling chain. You do not need to know the precise values ahead of time. If the profiler runs out of memory to hold data in its buffers, the profiler loses some data but notifies you of this in the results. You can then modify the parameters in the call to `ProfilerInit()` to increase the buffers and preserve all your data.

The profiler allocates buffers in the profiled application's heap based on the method of collection, the number of routines, and the depth of the calling tree. On platforms where it is possible, the profiler will allocate memory outside of the application's heap, which helps reduce the profiler's effect on the application.

The call to `ProfilerInit()` returns a non-zero error value if the call fails for any reason. Use the return value to ensure that memory was allocated successfully before continuing with the profiler. Typically you would add this call as conditionally compiled code so that it compiles and runs only if profiling is on and the call to `ProfilerInit()` was successful.

You call `ProfilerInit()` before any profiling occurs. Typically you make the call at the beginning of your code.

See also [Time and Timebases](#) and [Memory Usage](#)

Calling `ProfilerInit()` in C/C++

In C/C++, the call would be at the beginning of your `main()` function.

The call might look like this:

```
if (!ProfilerInit(collectDetailed, bestTimeBase, 20, 5))
{
// your profiled code
}
```

Of course, your parameters may vary depending upon how many routines you have and the depth of your calling chains.

Terminating Profiler with `ProfilerDump()`

The profiler dumps its data to a file when you call `ProfilerDump()`. The file appears in the current default directory, usually the project directory.

You provide a file name when you call `ProfilerDump()`. You may dump results as often as you like. You can provide a different file name for intermediate results (if you have multiple calls to `ProfilerDump()`), or use the same name. If the specified file already exists, a new file is created with an incrementing number appended to the file name for each new file. This allows the dump to be called inside a loop with a constant file name. This can be useful for dumping intermediate results on a long task.

`ProfilerDump()` does not clear accumulating results. If you want to clear results you can call `ProfilerClear()`.

A typical call to `ProfilerDump()` would be placed just before you exit your program, or at the end of the code you are profiling. The prototypes for `ProfilerDump()` are listed in [Table 23.3](#).

Table 23.3 ProfilerDump() Prototypes

C/C++	long ProfilerDump(unsigned char *filename);
-------	-------------------------------------------------

Calling ProfilerDump()

There is only one parameter: `char*`. The parameter points to a C-style string for `filename`. The IDE automatically adds a `.cwp` extension to the file name.

Profiling Abnormally Terminated Functions

The profiler correctly reports data for abnormally terminated functions that exited through the C++ exception handling model (`try`, `throw`, `catch`) or the ANSI C library `setjmp()` and `longjmp()` routines. You do not have to do anything to get this feature, it is automatic and part of the profiler's design.

However, there is a possibility of some errors in the reported results for an abnormally terminated function.

First, the profiler does not detect the abnormal termination until the next profiling call after the abnormal termination. Therefore, some additional time will be reported as belonging to the terminated function.

Second, if the next profiler event is a profiler entry, and the new stack frame for that function is larger than the frames that were abnormally exited, the profiler will not immediately detect that the original function was abnormally terminated. In that case the profiler will treat the function just entered as a child of the function abnormally terminated. The profiler will correct itself on the next profiling event without this property—that is, when the stack returns to a point smaller than it was when the abnormally terminated function exited.

Finally, remember that the profiler is not closed properly and the output file is not dumped when `exit()` is called. If you need to call `exit()` in the middle of your program and want the profiler output, call `ProfilerDump()`.

If you are using the profiler, you should always call `ProfilerTerm()` before `exit()`.

CAUTION If a program exits after calling `ProfilerInit()` without calling `ProfilerTerm()`, timers may be left running that could crash the machine.

Debugging Profiled Code

It is possible to debug code that has calls to the profiler in it. However, the profiler does interfere with stepping through code. You may find it simpler to debug non-profiled code, and profile separately. In this section, we describe what happens when you step into and out of a profiled routine. In addition, we describe the effects that stopping in the debugger has on the profile results.

See also the *CodeWarrior IDE User Guide* for more information on how to use the debugger.

Stepping into a Profiled Routine

If you step into a profiled routine you may see assembly code instead of source code. The compiler has added calls to `__PROFILE_ENTRY` at the start of the routine. This is how the profiler knows when to start counting time for the routine.

If you step through the assembly code far enough to get to the code derived from the original source code, then switch the view from source to assembly and back again, you can see the original source code.

Stepping out of a Profiled Routine

If you single-step out of a routine being profiled, you may end up in the `__PROFILE_EXIT` assembly code from the profiler library. This is how the profiler knows when to stop counting time for the routine.

Effect of Stopping on the Profile Results

If you stop in a profiled routine, the profiler counts all the time you spend in the debugger as time that routine was running. This skews the results.

CAUTION If you debug profiled code, you should not kill the code from the debugger. If you have called `ProfilerInit()` you should call `ProfilerTerm()` on exit. If you do not do so, you may crash your system.

Viewing Results

This section discusses the ways you may view the data created by the CodeWarrior profiler.

In this section you will look at:

- [What It Does](#)—the principle features of the profiler
- [How It Works](#)—the profiler interface and how you can view data
- [Finding Performance Problems](#)—use the profiler to locate problems

What It Does

The Profiler window displays profiler output for you to analyze the results of your program's execution. The profiler reads the dump files created by the calls in your code and displays the data in a form that you can use. Using the data display you can:

- sort data by any of several relevant criteria such as name, time in routine, percent of time in routine, and so forth
- open multiple profiles simultaneously to compare different versions of the profiled code
- identify trouble spots in the code
- view summary, detailed, or object-based data

How It Works

You open profile data files exactly as you open files in any application. You can use the **Open** command from the **File** menu or drop the data file's icon on the CodeWarrior IDE. Whatever approach you take, when you open a file a window appears.

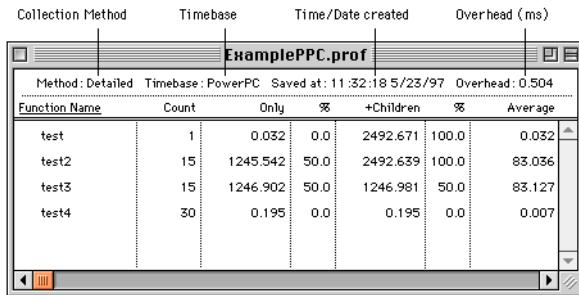
Profiler Window

The Profiler window allows you to view several elements of the profile data simultaneously, as shown in [Figure 23.2](#).

Profiler

Viewing Results

Figure 23.2 Profiler Window



Profiler Window Data Columns

The profiler window contains a series of columns containing data from the profile. All times are displayed according to the resolution of the timer that you use to profile data. The results in the window are only as precise as the timer used.

The times shown in the data columns are relative. Each time datum is reported to three decimal places. However, some time bases (most notably `ticksTimeBase`) are less precise. See [Time and Timebases](#)

[Table 23.4](#) lists each of the columns in the profiler window (from left to right) and the information that column contains.

Table 23.4 Profile Window Data Columns

Column	Contents
Function name	Routine name. (The profiler unmangles C++ function names.)
Count	Number of times this routine was called.
Time	Time spent in this routine, not counting time in routines that this routine calls.
%	Percent of total time for the Time column.
+Children	Time spent in this routine and all the routines it calls.
%	Percent of total time for the +Children column.
Average	Average time for each routine invocation: Time divided by the number of times the routine was called.

Table 23.4 Profile Window Data Columns

Column	Contents
Maximum	Longest time for an invocation of the routine.
Minimum	Shortest time for an invocation of the routine.

Sorting Data

You can view the data sorted by the value in any column. To change the sort order, click the column title. The heading becomes highlighted and data is sorted by the value in that column. Use the arrow control to change the direction of the sort (ascending/descending).

Multiple Windows

You can open any number of different profile windows simultaneously. This allows you to compare the results of different runs easily.

Window Views

In the tabs, you may choose to view the data in one of three ways: flat, detail, or class. Not all possibilities are available for all profiles.

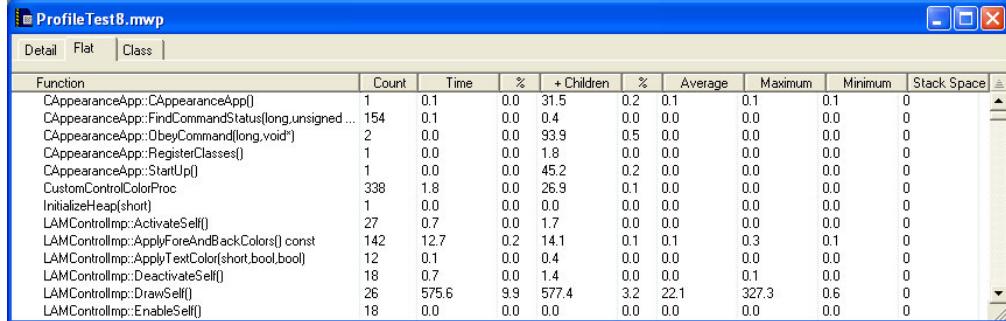
Flat View

The flat view displays a complete, non-hierarchical, flat list of each routine profiled. No matter what calling path was used to reach a routine, the profiler combines all the data for the same routine and displays it on a single line. [Figure 23.3](#) shows a flat view.

Profiler

Viewing Results

Figure 23.3 Flat View



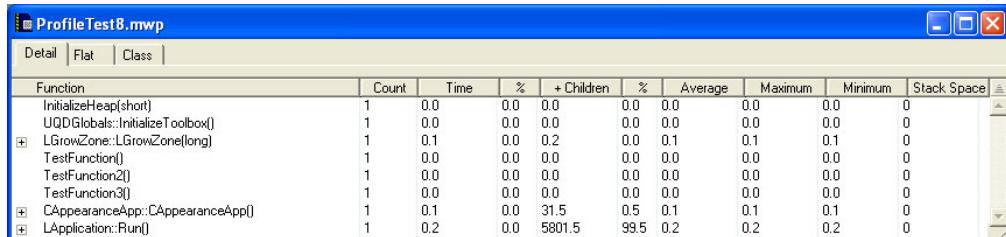
The flat view is particularly useful for comparing routines to see which take the longest time to execute. The flat view is also useful for finding a performance problem with a small routine that is called from many different places in the program. This view helps you look for the routines that make heavy demands in time or raw number of calls.

A flat view window can be displayed for any profile.

Detail View

The detail view displays routines according to the dynamic call tree as shown in [Figure 23.4](#).

Figure 23.4 Detail View



Routines that are called by a given routine are shown indented under that routine. This means that a routine may appear more than once in the profile if it called from different routines. This makes it difficult to tell how much total time was spent in a routine. However, you can use the flat view for that purpose.

The detail view is useful for detecting design problems in code; it lets you see what routines are called how often from what other routines. Armed with

knowledge of your code's underlying design, you may discover flow-control problems.

For example, you can use detailed view to discover routines that are called from only one place in your code. You might decide to fold that routine's code into the caller, thereby eliminating the routine call overhead entirely. If it turns out that the little routine is called thousands of times, you can gain a significant performance boost.

In detail view, sorting is limited to routines at the same level in the hierarchy. For example, if you sort by routine name, the routines at the top of the hierarchy will be sorted alphabetically. For each of those first-level routines, its second-level routines will be sorted alphabetically underneath it, and so on.

The detail view requires that `collectDetailed` be passed to `ProfilerInit()` when collecting the profile. If `collectSummary` is used, you cannot display the data in detailed view.

Class View

The class view displays summary information sorted by class. Beneath each class the methods are listed. This is a two-level hierarchy. You can open and close a class to show or hide its methods, just like you can in the detail view.

When sorting in class view, functions stay with their class, just like subsidiary functions in detail view stay in their hierarchical position. [Figure 23.5](#) shows the methods sorted by count.

Class view allows you to study the performance impact of substituting one implementation of a class for another. You can run profiles on the two implementations, and view the behavior of the different objects side by side. You can do the same with the flat view on a routine-by-routine basis, but the class view gives you a more natural way of accessing object-based data. It also allows you to gather all the object methods together and view them simultaneously, revealing the effect of interactions between the object's methods.

Profiler

Viewing Results

Figure 23.5 Class View

The screenshot shows a Windows application window titled "ProfileTest8.mwp". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar is a menu bar with "Detail", "Flat", and "Class" options. The main area is a table titled "Function" with the following columns: Count, Time, %, + Children, %, Average, Maximum, Minimum, and Stack Space. The table lists numerous C++ functions, many of which have a plus sign before their names, indicating they are collapsed. The "Time" column shows values like 0.0, 70.3, and 3.4. The "% Time" column shows values like 0.0, 1.2, and 0.1. The "+ Children" column shows values like 0.0, 43.1, and 3.4. The "Stack Space" column shows values like 0, 0.0, and 0.

Function	Count	Time	%	+ Children	%	Average	Maximum	Minimum	Stack Space
+ TestFunction()	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
+ TestFunction2()	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
+ InitializeHeap(short)	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
+ TestFunction3()	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
+ SiControlActionUPP::LChasingArrows::	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
+ TRegistrar<LSlider>::RearrangeObjects<7LWindow>_11UReanimatorFUIs_...	2	0.0	0.0	1.5	0.0	0.0	0.0	0.0	0
+ TRegistrar<LProgressBar>::	2	0.0	0.0	1.8	0.0	0.0	0.0	0.0	0
+ TRegistrar<LSeparatorLine>::	2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0
+ TRegistrar<AMWindowHeaderImp>::LSeparatorLine::	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
+ STUnhiliteMenu::	2	70.3	1.2	70.3	1.2	35.1	70.3	0.0	0
+ TRegistrar<LWindowHeader>::	2	0.0	0.0	2.7	0.0	0.0	0.0	0.0	0
+ TRegistrar<LWindowThemeAttachment>::	2	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0
+ TRegistrar<LChasingArrows>::UAEDesc::	2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0
+ TRegistrar<LWindow>::LComparator::	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
+ UAppleEventsMgr::	3	0.8	0.0	1.1	0.0	0.3	0.6	0.1	0
+ UCControlRegistry::	4	0.1	0.0	3.4	0.1	0.0	0.0	0.0	0

Object view will display “N/A” (Not Available) in the +Children column for classes in a collectSummary profile. This is because the detail information is missing from the file.

The class view requires that the profile contain at least one mangled C++ name. If there is none, you cannot use object view.

Finding Performance Problems

As you work with the profiler, you will see that the information provided quickly guides you to problem areas.

To look for time hogs, sort the view by either the Time column or the +Children column. Then examine routines that appear near the top of the list. These are the routines that swallow the greatest percentage of your code’s time. Any improvement in these routines will be greatly magnified in your code’s final performance.

You may also want to sort based on the number of times a routine is called. The time you save in a heavily-used routine is saved each time it is called.

If stack size is a concern in your code, you can sort based on the Stack Space column. This lets you see the largest size the stack reached during the profile.

Troubleshooting

This section answers common questions about the profiler. So if you have a problem with the profiler, consult this section first. Other users may have encountered similar difficulties, and there may be a simple solution.

Profile Times Vary Between Runs

“I’m getting different results (within 10%) in the profiler every time I run my program.”

Background

There are two potential reasons that this may be happening. Both are time-related problems. The first problem that can occur is inadequate time in the function relative to the profiler resolution. The second problem is clock resonance.

Inadequate Time in the Function

If the function time that you are trying to measure is only 10 times greater than the resolution of the timebase, you will encounter this problem.

Solution

To solve this problem, increase the number of times your function is called, then the average the profiler computes will be more accurate.

Sometimes it is helpful to pull a routine out of a program, and into a special test program which calls it many times in a loop for performance tuning purposes. However, this technique is susceptible to cache differences between the test and real program.

Clock Resonance

If the operations you are performing in your profiled code coincide with the incrementing of the profiler clock, the results can be distorted, and could show wild variations.

Solution

Avoid this problem by increasing the number of times your function is called.

Problems while Profiling Inline Functions

“My inline functions are not getting inlined when I’m profiling my code. What’s happening?”

Background

When the compiler switch for profiling is turned on, the default setting for “don’t inline functions” is changed to true. This is so that these functions will have profiling information collected for them.

Solution

Place a `#pragma dont_inline off` in your source file to turn on function inlining again. You will not collect profile information for inline functions. In effect, a function can be inlined or profiled, but not both. The profiler cannot profile an inlined function.

TIP If you use the `#pragma dont_inline off` in your code, you may see profile results for some inline functions.

When you declare an inline function, the compiler is allowed, but not required to inline the function. It is perfectly legal for the compiler to inline some functions, but not others. Data is collected only for the calls that were not inlined. The calls that were inlined have their time added into the time of the calling function.

Profiling Library Could not be Found

“While trying to profile my dynamically linked library (shared library), I get an error message saying that the profiling library could not be found.”

Background

This problem occurs when trying to use the profiling library to profile your dynamically linked library and the profiling library is not in the search path.

Solution

Add the profiling library to the search path. If you are using the CodeWarrior IDE, see the *CodeWarrior IDE User's Guide* for information on search paths.

Reference

This section contains the detailed technical reference information you may need when using the profiler.

The topics discussed include:

- [Compiler Directives](#)—handling compiler directives
- [Memory Usage](#)—understanding memory usage
- [Time and Timebases](#)—the available time resolutions
- [Profiler Function Reference](#)—a reference for all of the profiler API functions

Compiler Directives

You can control routine-level profiling using compiler directives.

The C/C++ compiler supports three preprocessor directives that you can use to turn compiling on and off at will.

#pragma profile on	Enables calls to the profiler in functions that are declared following the pragma.
#pragma profile off	Disables calls to the profiler in functions that are declared following the pragma.
#pragma profile reset	Sets the profile setting to the value selected in the preferences panel.

You can use these directives to turn profiling on for any functions you want to profile, regardless of the settings in the Processor preferences. You can also turn off profiling for any function you do not want to profile.

As there are compiler directives to turn the profiler on and off, there are also directives to test if the profiler is on. You can use these tests in your code so that you can run your program with or without the profiler and not have to modify your code each time.

In C/C++, use the `#if-#endif` clause. For example:

```
void main()
{
#if __profile__      // is the profiler on?
    if (!ProfilerInit(collectDetailed, bestTimeBase, 20, 5))
    {
#endif
        test(15);
#if __profile__
    ProfilerDump("Example.prof");
    ProfilerTerm();
}
#endif
}
```

See also [Routine-Level Profiling](#)

Memory Usage

The profiler allocates two buffers in your program's heap to hold data as it collects information about your code: one based on the number of routines, and one based on the stack depth. You pass these parameters in your call to `ProfilerInit()`.

When possible, the profiler will allocate its memory outside of your program's heap to reduce the impact of the profiler on your program. If this is not possible, the profiler's memory buffers will be allocated in your program's default heap. You must ensure that the heap is large enough to hold both your program's dynamically allocated data and the profiler's buffers.

In summary collection mode, the profiler allocates `64 bytes * numFunctions` and `40 bytes * stackDepth`.

In detailed collection mode, the profiler allocates `12 * 64 * numFunctions` bytes and `40 * stackDepth` bytes.

As an example, assume numFunctions is set to 100, and stackDepth to 10. In summary mode the profiler allocates buffers of 6,400 bytes and 400 bytes. In detailed mode it allocates buffers of 76,800 bytes and 400 bytes.

`ProfilerGetDataSizes()` lets you query the profiler for the current size of the data collected in the function and stack tables. This information can be used to tune the parameters passed to `ProfilerInit()`.

See also [“`ProfilerInit\(\)`” on page 346.](#)

Time and Timebases

The `timeBase` may be one of the following values:

- `ticksTimeBase`
- `microsecondsTimeBase`
- `timeMgrTimeBase`
- `PPCTimeBase`
- `win32TimeBase`
- `bestTimeBase`

The `bestTimeBase` option automatically selects the most precise timing mechanism available on the computer running the profiled software. Not all of these values are supported on all target platforms. Refer to the Targeting Manual for your product to determine which timebases are available for use.

When you call `ProfilerInit()`, the constant `bestTimeBase` tells the profiler to figure out the most precise timebase available on your platform and to use it.

Profiler Function Reference

This is a reference for all profiler functions mentioned in the text of this manual. The functions described in this section are:

- [ProfilerInit\(\)](#)
- [ProfilerTerm\(\)](#)
- [ProfilerSetStatus\(\)](#)
- [ProfilerGetStatus\(\)](#)
- [ProfilerGetDataSizes\(\)](#)

- [ProfilerDump\(\)](#)
- [ProfilerClear\(\)](#)

The discussion of each function includes the following attributes:

- Description: A high-level description of the function
- Prototypes: The entire C/C++ prototypes for the function
- Remarks: Implementational or other notes about the function

ProfilerInit()

`ProfilerInit()` prepares the profiler for use and turns the profiler on. The parameters tell the profiler how this collection run is going to operate, and how much memory to allocate. `ProfilerInit()` *must* be the first profiler call before you can call any other routine in the profiler API.

Prototypes

```
typedef enum {  
    collectDetailed,  
    collectSummary  
} ProfilerCollectionMethod;  
  
typedef enum {  
    bestTimeBase  
} ProfilerTimeBase;  
  
long ProfilerInit(  
    ProfilerCollectionMethod method,  
    ProfilerTimeBase bestTimeBase,  
    long numFunctions, short stackDepth);
```

Remarks

`ProfilerInit()` will allocate its memory outside of your program's heap to reduce the impact of the profiler on your program. If this is not possible, the profiler's memory buffers will be allocated in your program's default heap. You must ensure that the heap is large enough to

hold both your program's dynamically allocated data and the profiler's buffers.

`ProfilerInit()` returns an error status that indicates whether or not the profiler was able to allocate its memory buffers. If the return value is 0, then memory allocation was successful. If a non-zero value is returned, then the allocation was not successful.

The `method` and `timeBase` parameters select the appropriate profiler options. The `numFunctions` parameter indicates the number of routines in the program for which the profiler should allocate buffer storage. If the profiler is operating in detailed mode, this number is internally increased (exponentially), because of the branching factors involved. The `stackDepth` parameter indicates how many routines deep the stack can get.

A call to `ProfilerInit()` must be followed by a matching call to `ProfilerTerm()`.

ProfilerTerm()

`ProfilerTerm()` stops the profiler and deallocates the profiler's buffers. It calls `ProfilerDump()` to dump out any information that has not been dumped. `ProfilerTerm()` must be called at the end of a profile session.

```
void ProfilerTerm( void );
```

Remarks

If a program exits after calling `ProfilerInit()` you should call `ProfilerTerm()`. Failing to do so may lead to a crash on some platforms.

ProfilerSetStatus()

`ProfilerSetStatus()` lets you turn profiler recording on and off in the program. This makes it possible to profile specific sections of your code such as screen redraw or a calculation engine. The profiler output makes more sense if

the profiler is turned on and off in the same routine, rather than in different routines.

```
void ProfilerSetStatus( short on );
```

Remarks

This routine and `ProfilerGetStatus()` are the only profiler routines that may be called at interrupt time.

Pass 1 to turn recording on and 0 to turn recording off.

ProfilerGetStatus()

`ProfilerGetStatus()` lets you query the profiler to determine if it is collecting profile information.

```
short ProfilerGetStatus( void );
```

Remarks

This routine and `ProfilerSetStatus()` are the only profiler routines that may be called at interrupt time.

`ProfilerGetStatus()` returns a 1 if the profiler is currently recording, 0 if it is not.

ProfilerGetDataSizes()

`ProfilerGetDataSizes()` lets you query the profiler for the current size of the data collected in the function and stack tables. This information can be used to tune the parameters passed to `ProfilerInit()`.

Prototypes

```
void ProfilerGetDataSizes(  
    long *functionSize,  
    long *stackSize);
```

Remarks

If you have passed `collectDetailed` to `ProfilerInit()`, `ProfilerGetDataSizes()` returns the number of actual routines in the table, which may be larger than the value passed to `ProfilerInit()` in `numFunctions`. This is because the profiler multiplies `numFunctions` by 12 when it allocates the table. The multiplication is done so that you can easily switch between `collectDetailed` and `collectSummary` methods without changing the parameters.

ProfilerDump()

`ProfilerDump()` dumps the current profile information without clearing it.

```
long ProfilerDump( char* filename );
```

Remarks

This can be useful for dumping intermediate results on a long task. If the specified file already exists, a new file is created with an incrementing number appended to the filename. This allows the dump to be called inside a loop with a constant filename.

A non-zero value from `ProfilerDump()` indicates that an error has occurred.

ProfilerClear()

`ProfilerClear()` clears any profile information from the buffers.

```
void ProfilerClear( void );
```

Remarks

`ProfilerClear()` retains the settings of `collectionMethod` and `timeBase` that were set by `ProfilerInit()`. It does not deallocate the buffers.

Compilers and Linkers

This section contains these chapters:

- [Compilers](#)
- [Linkers](#)

Compilers

This chapter explains how to work with compilers in the CodeWarrior™ IDE. The IDE uses compilers to complete these tasks:

- Generate object code—the compiler translates source code into object code. Sample source code includes C++ files and Java files. Object code represents the same source instructions in a language that the computer directly understands.
- Flag syntax errors—the compiler highlights source code that generates syntax errors. Syntax errors result from failing to follow valid structure in a programming language. In C++, a common syntax error is forgetting to end a statement with a semicolon.

Read this chapter to learn more about typical tasks for working with compilers.

This chapter contains these sections:

- [Choosing a Compiler](#)
- [Compiling Projects](#)

Choosing a Compiler

Choose a compiler to determine how the IDE interprets source code. The IDE uses a *plug-in* compiler architecture. This architecture provides these features:

- Modularity—the IDE associates a specific compiler plug-in with a particular programming language or environment. For example, a compiler plug-in exists for C++ source code, and another compiler plug-in exists for Java source code.
- Flexibility—as new programming languages develop, the IDE can use new compiler plug-ins.

The IDE associates common filename extensions with various plug-in compilers. For example, most Java files have the filename extension `.java`. The IDE associates these files with the Java compiler. The **File Mappings** panel provides control over such associations.

Compiling Projects

Compile projects to process the source files that comprise a program and generate object code. The compiler flags syntax errors in the source files.

Use these tasks to compile projects:

- Compile source files.
- Set the build order or link order.
- Update a project or its files.
- Create an executable file from a project.
- Run an application created from the project.
- Remove object code.

This section explains how to perform each task.

Compiling Source Files

Use the **Compile** commands to compile source files into binary files. The IDE can compile a single file, multiple files, or all files in an open project.

1. Enable the Project window that contains the desired files to be compiled.
2. Select one or more files.
3. Choose **Project > Compile**.

The IDE compiles the selected files.

NOTE The **Project** menu contains most commands for compiling and linking projects. However, depending on the project type, some commands might be disabled or renamed.

Setting the Build and Link Order of Files

Use the **Link Order** view in the Project window to specify the order in which the compiler and linker process files. Establishing the proper link order prevents link errors caused by file dependencies. The **Link Order** view is sometimes called the **Segments** view or **Overlays** view, depending on the target.

1. Click the **Link Order** tab in a Project window.

2. Click and drag files into the desired link order.

The IDE changes the link order. The build begins at the top of the link order, processes each file, and concludes at the bottom of the link order.

NOTE The IDE uses the new link order during subsequent **Update**, **Make**, **Run**, and **Debug** operations.

Updating Projects

Use the **Bring Up To Date** command to compile, but not link, the newly added, modified, and touched files in a project. Unlike the **Make** and **Run** commands, the **Bring Up To Date** command does not produce a binary file.

1. Select the project to update.
2. Choose **Project > Bring Up To Date**.

The IDE compiles all uncompiled project files.

Making Executable Files

Use the **Make** command to compile the newly-added, modified, and touched files in a project, then link them into a binary file. Unlike the **Run** command, the **Make** command does not execute the binary file. The **Make** command is useful for creating dynamic link libraries (DLLs), shared libraries, code resources, or tools.

1. Select the project to make.
2. Choose **Project > Make**.

The IDE processes the project and creates a binary file.

Running Application Projects

Use the **Run** command to perform these tasks:

- Compile and link a project (if necessary).
- Create a standalone application.
- Change project settings (if required).

Compilers

Compiling Projects

- Save the application.
- Run the application.

Note, the **Run** command is not available if the project creates a non-executable file like a dynamic linked library (DLL), shared library, library, code resource, or tool.

1. Select the project to run.
 2. Choose **Project > Run**.
-

Synchronizing File Modification Dates

Use the **Synchronize Modification Dates** command to update the modification dates of all files stored in a project. This command is useful for handling files from a third-party editor that does not share file-status information with the IDE.

1. Select the project window.
2. Choose **Project > Synchronize Modification Dates**.

The IDE checks the file-modification dates and marks modified files for re-compilation.

Removing Object Code

Use the **Remove Object Code** command to remove binary object code stored in the project file and reduce project size.

1. Open the desired project.
2. Choose **Project > Remove Object Code**.
3. Set compaction options as desired.
 - Select **Recurse subprojects** to remove object code from all subprojects in the project file.
 - Select **Compact targets** to remove these items:
 - Target data files with the .tdt extension.
 - Browser data.
 - Dependency information.
 - Additional data cached by the IDE.

4. Select the method by which the IDE removes the object code.
 - Click **All Targets** to remove object code from all build targets.
 - Click **Current Target** to remove object code only from the active build target.

The IDE removes the specified object code from the project.

Compilers

Compiling Projects

Linkers

This chapter explains how to work with linkers in the CodeWarrior™ IDE. The IDE uses linkers to complete these tasks:

- Combine code—the linker combines source-file object code with object code from library files and other related files. The combined code represents a complete computer program.
- Create a binary file—the linker processes the complete program and generates a binary file. Sample binary files include applications and shared libraries.

Read this chapter to learn more about typical tasks for working with linkers.

This chapter contains these sections:

- [Choosing Linkers](#)
- [Linking Projects](#)

Choosing Linkers

Choose a linker to determine the binary file type produced by the IDE. This list describes common binary files:

- Applications—applications, or executable files, represent a wide body of computer programs. Common applications include word processors, web browsers, and multimedia players.
- Libraries—libraries contain code for use in developing new computer programs. Libraries simplify programming tasks and enhance re-usability.
- Specialized files—files designed for highly efficient operation in a specific context. Such files usually support a particular combination of hardware and software to perform tasks.

The IDE provides various linkers for software development. The **Target Settings** panel contains an option for selecting a linker. The IDE maps to each linker a group of recognized filename extensions. These mappings determine how the IDE interprets each file.

Linking Projects

Link projects to process object code and generate a binary file. Refer to the CodeWarrior *Targeting* documentation for more information about linkers for specific computer systems. This section explains general-purpose linker tasks.

Generating Project Link Maps

Use the **Generate Link Map** command to create a link-map file that contains function and cross-section information about the generated object code. The link map reveals the files, libraries, and functions ignored by the IDE while producing the binary output.

The IDE stores the link-map file in the project folder. The file uses the same name as the build target, with a .MAP or .xMAP extension.

1. Select the project window.
2. Choose **Edit > targetname Settings...**
3. Select the linker panel in the **Target Settings Panels** list.
4. Select the **Generate Link Map** option.
5. Click **Save**.
6. Choose **Project > Make**.

The IDE generates the link-map file.

Preferences and Target Settings

This section contains these chapters:

- [Customizing the IDE](#)
- [Working with IDE Preferences](#)
- [Working with Target Settings](#)
- [Preference and Target Settings Options](#)

Customizing the IDE

The CodeWarrior™ IDE enables you to customize menus, toolbars, and key bindings to suit your programming preferences. Use the **Customize IDE Commands** window—which consists of the Commands, Toolbar Items, and Key Bindings tabs—to build your customizations.

This chapter contains these sections:

- [“Customizing IDE Commands” on page 363](#)
- [“Customize Toolbars” on page 373](#)
- [“Customize Key Bindings” on page 379](#)

Customizing IDE Commands

You can customize the menu commands in the IDE’s menu bar, as well as control the appearance of specific menu commands, create new command groups to distinguish menu commands, and associate a command line (Windows, Solaris, and Linux) or a script or application (Mac OS) with a new menu command. The customized menu commands you create have access to IDE information, such as the current editor selection, the frontmost window, and the current project and its output file.

Select **Edit > Commands & Key Bindings** to access the Customize IDE Commands window. [Figure 26.1 on page 364](#) shows the Customize IDE Commands window. [Table 26.1 on page 364](#) explains each button in the window. See the tasks in this chapter for more detailed information.

Customizing the IDE

Customizing IDE Commands

Figure 26.1 Customize IDE Commands window

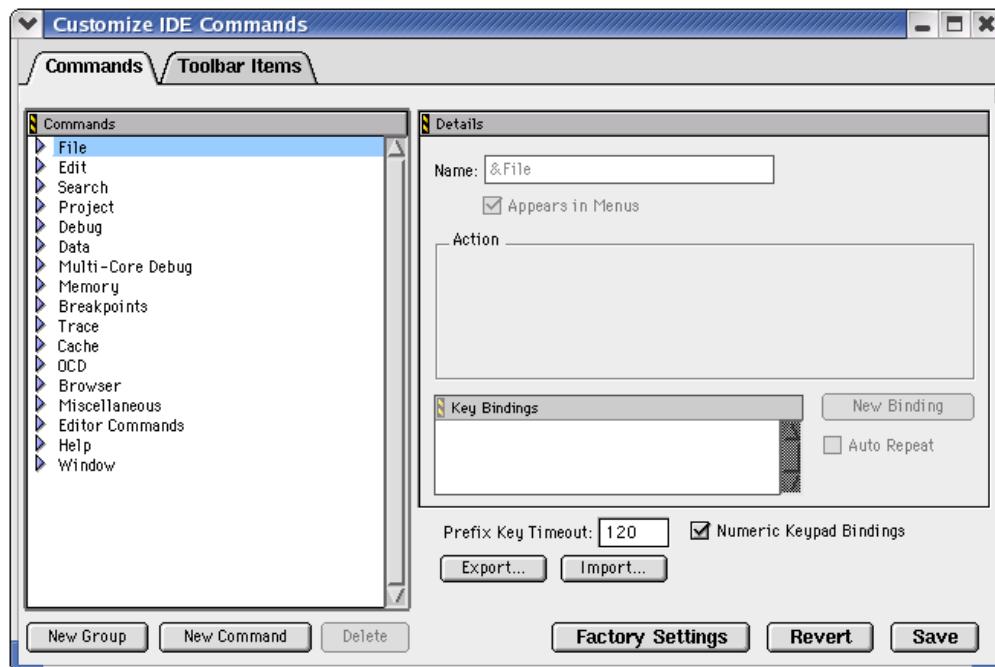


Table 26.1 Customize IDE Commands window—button overview

Button name	Explanation
New Group	Click to add a new command group to the Commands list.
New Command	Click to add a new command setting within a group.
Factory Settings	Click to restore default options for the current Customize IDE Commands (Commands and Toolbar Items) lists.
Revert	Click to restore the most recently saved options for the current Customize IDE Commands (Commands and Toolbar Items) lists.
Export	Click to save a file that contains commands and key bindings to use later in the Customize IDE Commands lists.

Table 26.1 Customize IDE Commands window—button overview (*continued*)

Button name	Explanation
Import	Click to open a file that contains commands and key bindings to use in the current Customize IDE Commands lists.
Save	Click to save customizations to the Customize IDE Commands list.

Commands Tab

Click the **Commands** tab at the top of the Customize IDE Commands window to display the commands view. Use this view to modify existing menu commands, and to create and remove command groups and menu commands.

Modifying Existing Commands

You can use the **Commands** tab of the Customize IDE Commands window to examine and modify existing command groups and menu commands. This view includes a Commands list. 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. Information for the selected item appears on the right-hand side of the Customize IDE Commands window. This window provides this information for each selected 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 items, 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 a menu command allows that menu command to appear under the related command group in the menu bar. Disabling the checkbox prevents the menu command from appearing in the menu bar under the command group.
- **Action**—This section shows information about the action the selected item performs. For default menu commands, this section shows the command type, such as **Command** or **Hierarchical Menu**. For customized menu commands that you create, this section lets you specify a command line

Customizing the IDE

Customizing IDE Commands

(Windows, Solaris, and Linux) or a script (Mac OS) that runs after you choose the customized menu command.

- **Key Bindings**—This area consists of the Key Bindings list, the **New Binding** button, and the **Auto Repeat** checkbox.
-

Creating a New Command Group

To create a new command group for menu commands, follow these steps:

1. Click the **New Group** button.

The IDE creates a new command group called **New Group**, adds it to the Commands list, and displays its information in the Customize IDE Commands window.

2. Rename the new command group in the **Name** field.

3. Use the **Appears in Menus** checkbox to toggle the availability of the new command group in the IDE menu bar.

Select the Appears in Menus checkbox to display the new command group in the menu bar. Clear the checkbox if you do not want the command group to appear in the menu bar.

4. Click **Save**.

The IDE saves your new command group. If you selected the **Appears in Menus** checkbox, your new command group appears in the menu bar.

Creating a New Menu Command

To create a new menu command, follow these steps:

1. Select the command group you want to contain the new menu command.

You must select an existing command group in the Commands list.

2. Click the **New Command** button.

The IDE creates a new menu command named **New Command** and places it within the selected command group. The information for the new menu command appears in the Customize IDE Commands window.

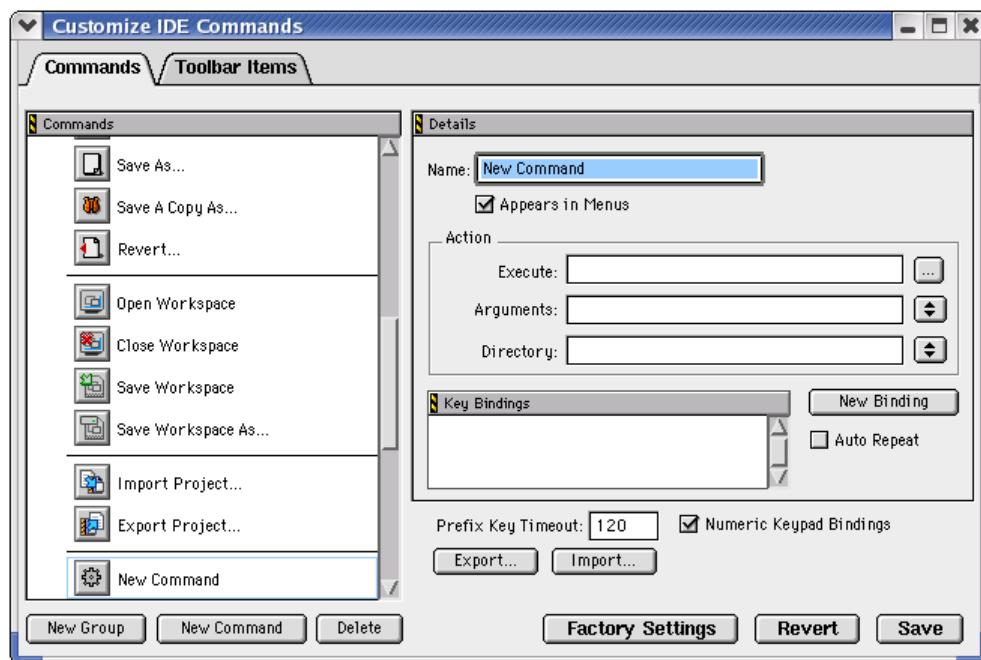
3. Enter a name for the new menu command.

You can change the default name of **New Command**. Enter a new name in the Name field of the Customize IDE Commands window.

4. Use the **Appears in Menus** checkbox to toggle the availability of the new command within its command group.
5. Define the desired Action for the new menu command.
6. Click **Save**.

The IDE saves your new menu command. If you enabled the **Appears in Menus** checkbox, the new menu command appears within the selected command group.

Figure 26.2 Command action fields



Defining Command Actions (Windows)

These fields help you associate an action with the new menu command:

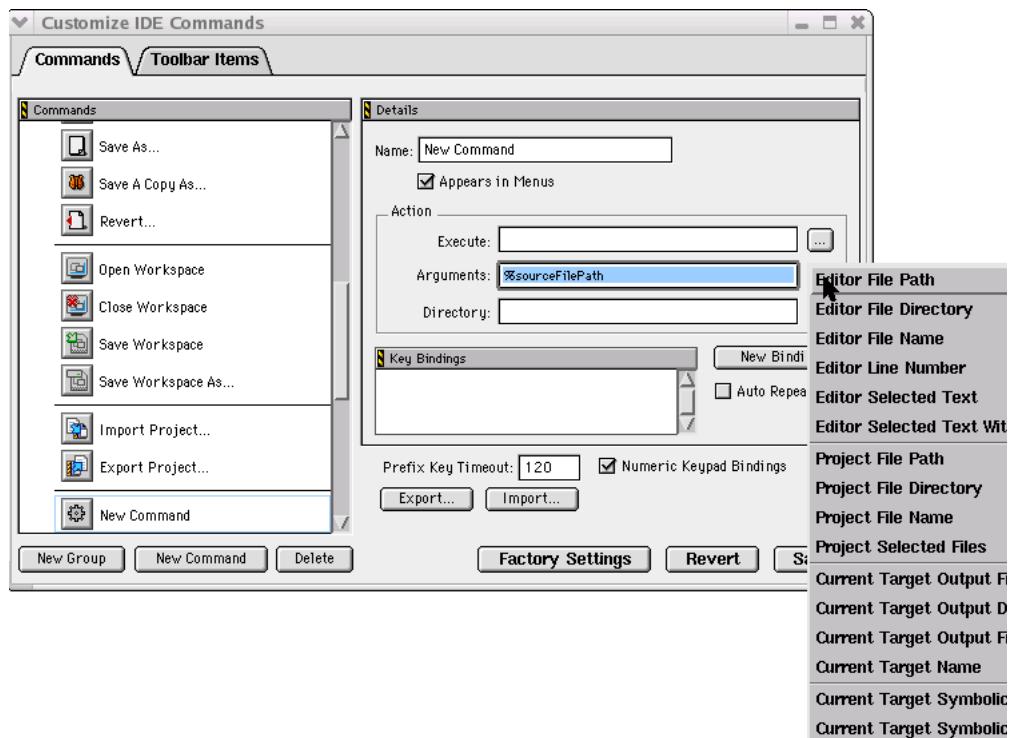
- **Execute**—Enter in this field a command to run an application. Alternatively, click the ellipsis button next to the field to select the application using a standard dialog box.
- **Arguments**—Enter the arguments that the IDE must pass to the application specified in the Execute field. Alternatively, choose the desired arguments from the pop-up menu next to the field.
- **Directory**—Enter the working directory the IDE should use when it executes the application specified in the Execute field. Alternatively, choose the desired directory from the pop-up menu next to the field.

Pre-defined Variables in Command Definitions

The IDE provides pre-defined variables for Windows, Solaris, and Linux (not Mac OS) to associate actions with commands. When you create a new command, you can use these pre-defined variables in command definitions to provide additional arguments that the IDE passes to the application (which is specified in the Execute field).

NOTE You can use variables that end with `Dir` as both argument and directory names.

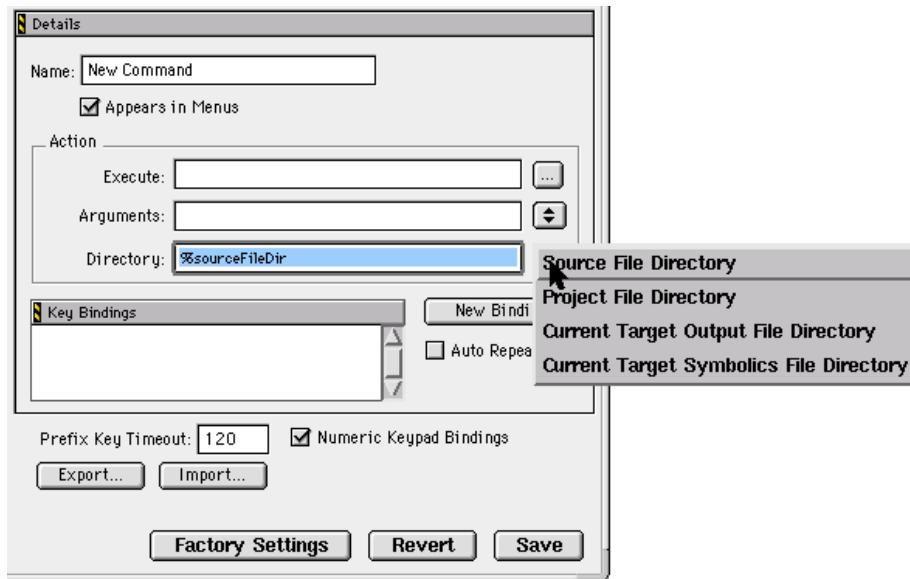
Figure 26.3 Pre-defined Arguments



Customizing the IDE

Customizing IDE Commands

Figure 26.4 Pre-defined Directory variables



[Table 26.2](#) explains the pre-defined variables for command-line arguments.

Table 26.2 Pre-defined variables in command definitions

Variable	Command-line output
%sourceFilePath	sourceFilePath is the frontmost editor window's full path.
%sourceFileDir	sourceFileDir is the frontmost editor window's directory.
%sourceFileName	sourceFileName is the frontmost editor window's filename.
%sourceLineNumber	sourceLineNumber is the line number of the insertion point in the front window.
%sourceSelection	sourceSelection is the path to a temporary file containing the currently selected text.

Table 26.2 Pre-defined variables in command definitions (*continued*)

Variable	Command-line output
%sourceSelUpdate	sourceSelUpdate is like sourceSelection , except the IDE waits for the command to finish and updates the selected text with the contents of the file.
%projectFilePath	projectFilePath is the full path of the front project window.
%projectFileDir	projectFileDir is the directory of the front project window.
%projectFileName	projectFileName is the filename of the front project window.
%projectSelectedFiles	projectSelectedFiles passes the selected filenames in the project window.
%targetFilePath	targetFilePath is the full path of the output file of the front project.
%targetFileDir	targetFileDir is the directory of the output file of the front project.
%targetFileName	targetFileName is the filename of the output file of the front project.
%currentTargetName	currentTargetName passes the name of the current target of the frontmost window.
%symFilePath	symFilePath is the full path to the symbolics file of the front project (can be the same as targetFile, such as CodeView).
%symFileDir	symFileDir is the full directory to the symbolics file of the front project (can be the same as targetFile, such as CodeView)
%symFileName	symFileName is the full filename to the symbolics file of the front project (can be the same as targetFile, such as CodeView)

Using a Pre-defined Variable

To use a pre-defined variable, follow these steps:

Customizing the IDE

Customizing IDE Commands

1. Create a new menu command.

The IDE creates a new menu command named **New Command** and places it within your selected command group. The information for the new menu command appears in the Customize IDE Commands window.

2. Enter a name for the new menu command.
3. Use the **Appears in Menus** checkbox to toggle the availability of the new command within its command group.
4. Define the Action for the new menu command.
 - a. Enter in the **Execute** field a command line to run an application.
 - b. Next to the **Arguments** field, click on the arrow icon and select an argument listed in the pop-up menu.
 - c. Next to the **Directory** field, click on the arrow icon and select a directory listed in the pop-up menu.
5. Click **Save**.

The IDE saves your new menu command with the pre-defined variables. If you enabled the **Appears in Menus** checkbox, the new menu command appears within the selected command group.

Defining Command Actions (Mac OS)

After you create a new menu command, the **Customize IDE Commands** window shows the **Run App/Script** field. This field appears in the Action section of the window.

1. Click the **Choose** button next to the field to display a standard Open dialog box.
2. Use the dialog box to select an application or script.

The IDE launches this application or script each time you choose the menu command. The path to the selected application or script appears in the **Run App/Script** field.

Deleting Command Groups and Menu Commands

You can delete the command groups and menu commands that you create for the IDE. Once removed, the command groups no longer appear in the IDE's menu

bar, and the menu commands no longer activate their associated command lines (Windows), applications or scripts (Mac OS).

NOTE If you need to temporarily remove your customized command groups and menu commands, consider exporting your settings. If you export your settings, you do not need to reconstruct them if you want them in the future.

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 **Delete**.

After clicking the **Delete** button, the selected command group or menu command disappears from the Commands list.

3. Click **Save**.

Clicking the **Save** button confirms the deletion. The IDE removes deleted command groups from its menu bar. Deleted menu commands disappear from their respective command groups.

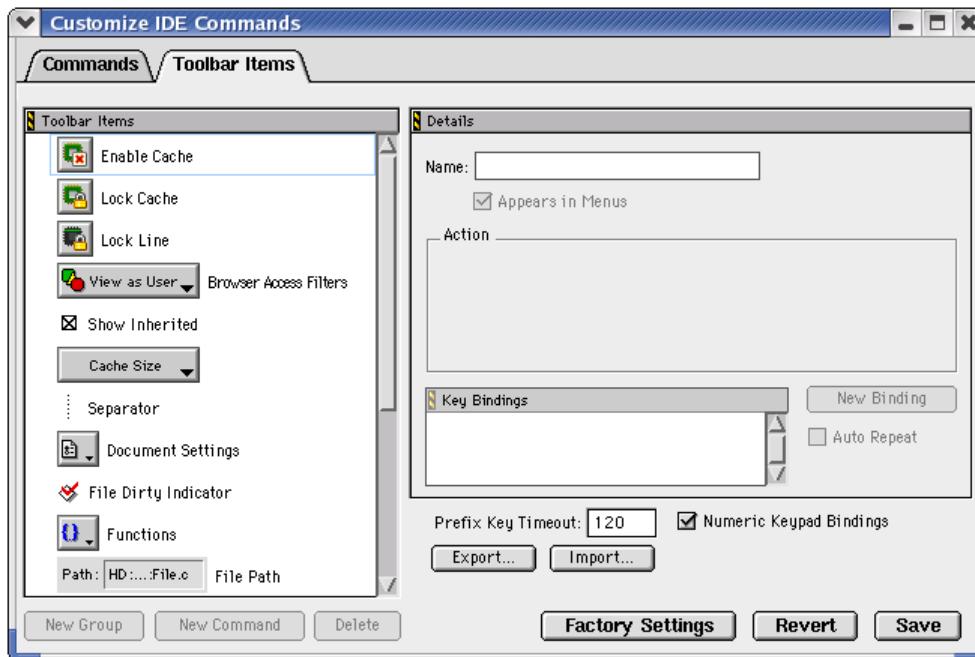
Customize Toolbars

You can customize your IDE toolbars to contain frequently used commands. The IDE toolbars contain a series of elements. Each element typically represents a menu command. After you click the element, the IDE executes the associated menu command. The toolbar can also contain elements that execute actions other than menu commands.

Customizing the IDE

Customize Toolbars

Figure 26.5 Toolbar Items tab



This section explains these topics:

- [“Kinds of Toolbars” on page 374](#)
- [“Toolbar Elements” on page 375](#)
- [“Modify a Toolbar” on page 375](#)

Kinds of Toolbars

The CodeWarrior IDE uses two toolbar types:

- Main toolbar (Windows OS)—This toolbar, also known as the floating toolbar, is always available.
- Window toolbars—These toolbars appear in particular windows, such as the Project window toolbar and the Browser window toolbar.

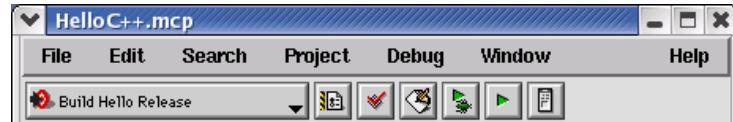
This distinction is important because you show, hide, clear, and reset the different toolbar types by using different sets of menu commands. These commands distinguish between the floating toolbar and the other window toolbars.

When you change one of these toolbar types, that change applies to every instance of that toolbar type you subsequently create. For example, if you modify the toolbar in an editor window, your changes appear in all editor windows opened thereafter.

Figure 26.6 Main toolbar (Windows OS)



Figure 26.7 Project window toolbar



Toolbar Elements

A toolbar can contain these elements:

- *Commands*—buttons that you click to execute IDE menu commands
- *Controls*—menus, such as Document Settings, Functions, Header Files, Markers, Version Control, and Current Target
- *Miscellaneous*—other elements, such as the File Dirty Indicator and File Path field
- *Scripts* (Mac OS)—buttons that you click to execute one of the scripts available through the Scripts menu

Click the **Toolbar Items** tab at the top of the Customize IDE Commands window to display the Toolbar view. Use this view to add new elements to a toolbar.

Modify a Toolbar

You can modify a toolbar in these ways:

- Add a toolbar element
- Remove a toolbar element
- Clear all elements on a toolbar
- Reset a toolbar

Customizing the IDE

Customize Toolbars

In certain circumstances there are restrictions on which elements you can add or remove from a toolbar. For example, you cannot add a second instance of an element to the toolbar.

After you modify a toolbar, the changes apply to every instance of that toolbar. For example, if you customize the Project window toolbar, those changes will affect every Project window that you open, not just the toolbar in the active Project window. Your changes do not affect windows that are already open.

TIP To display a ToolTip that names a toolbar element, rest the cursor over the element. On the Mac OS 9-hosted IDE, activate Balloon Help and rest the cursor over the element.

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 in the **Toolbar Items** view in the Customize IDE Commands window.

To add an element to a toolbar, follow these steps:

1. From the **Toolbar Items** list, select the icon next to the element that you want to add to a toolbar.

Make sure that the destination toolbar is visible.

2. Drag the element's icon from the Toolbar Items list to the destination toolbar.

On the Windows-hosted IDE, if the destination toolbar accepts the element, a framing bracket appears in the toolbar. This framing bracket shows you where the new element will appear after you release the cursor. If the destination toolbar does not accept the element, the framing bracket does not appear.

3. Release the element at the desired position.

After you release the element, the IDE inserts the element into the destination toolbar.

The toolbar might not accept an element for these reasons:

- The toolbar is full.
- The element already exists in the toolbar.
- The window does not support that element.

- The following elements can only be added to the editor window toolbar: Document Settings, Functions, Header Files, Markers, Version Control menus, File Dirty Indicator, and File Path field.
 - The **Current Target** menu element can only be added to the Project window toolbar.
-

Removing a Toolbar element

To remove an element from a toolbar, follow these steps:

1. Display a contextual menu for the button that you want to remove, as explained in [Table 26.3 on page 377](#).

Table 26.3 Displaying a contextual menu for a toolbar button

On this host...	Do this...
Windows	Right-click the button.
Macintosh	Control-click the button.
Solaris	Control-click the button.
Linux	Ctrl-click the button.

2. Select the **Remove Toolbar Item** command from the contextual menu.

The IDE removes the button from the toolbar.

Clearing All Buttons on Toolbars

You can clear all elements from a toolbar and build your own toolbar from scratch. [Table 26.4](#) explains how to clear the main (floating) toolbar and window toolbars.

Customizing the IDE

Customize Toolbars

Table 26.4 Clearing toolbars

On this host...	Do this to clear the main toolbar...	Do this to clear the window toolbar...
Windows	Select View > Toolbars > Clear Main Toolbar.	Select View > Toolbars > Clear Window Toolbar.
Macintosh	Select Window > Toolbar > Clear Floating Toolbar.	Select Window > Toolbar > Clear Window Toolbar.
Solaris	Select Window > Toolbar > Clear Floating Toolbar.	Select Window > Toolbar > Clear Window Toolbar.
Linux	Select Window > Toolbar > Clear Floating Toolbar.	Select Window > Toolbar > Clear Window Toolbar.

Reset Toolbars

Reset a toolbar to restore its default button set. [Table 26.5](#) explains how to reset the main (floating) toolbar and window toolbar by using menu commands.

Table 26.5 Resetting a toolbar by using menu commands

On this host...	Do this to reset the main toolbar...	Do this to reset the window toolbar...
Windows	Select View > Toolbars > Reset Main Toolbar.	Select View > Toolbars > Reset Window Toolbar.
Macintosh	Select Window > Toolbar > Reset Floating Toolbar.	Select Window > Toolbar > Reset Window Toolbar.
Solaris	Select Window > Toolbar > Reset Floating Toolbar.	Select Window > Toolbar > Reset Window Toolbar.
Linux	Select Window > Toolbar > Reset Floating Toolbar.	Select Window > Toolbar > Reset Window Toolbar.

Alternatively, you can use a contextual menu to reset the main toolbar or a window toolbar. Once you reset the toolbar, the IDE restores the default toolbar

button set. [Table 26.6](#) explains how to reset the main (floating) toolbar and window toolbar by using a contextual menu.

Table 26.6 Resetting a toolbar by using a contextual menu

On this host...	Do this to reset the main toolbar...	Do this to reset the window toolbar...
Windows	Right-click the toolbar and select Reset Toolbar .	Right-click the toolbar and select Reset Toolbar .
Macintosh	Control-click the toolbar and select Reset Toolbar .	Control-click the toolbar and select Reset Toolbar .
Solaris	Click and hold on the toolbar, then select Reset Toolbar .	Click and hold on the toolbar, then select Reset Toolbar .
Linux	Click and hold on the toolbar, then select Reset Toolbar .	Click and hold on the toolbar, then select Reset Toolbar .

Customize Key Bindings

You can customize the keyboard shortcuts, known as key bindings, for various commands in the CodeWarrior IDE. You can bind a set of keystrokes to virtually any command. To activate the command, type its associated key binding. Use the Customize IDE Commands window to change IDE key bindings.

You can also use the Customize IDE Commands window to look up default key bindings for specific commands, as well as change existing key bindings to better suit your needs.

Click the **Commands** tab at the top of the Customize IDE Commands window to display the Commands view. Use this view to configure key bindings for menu commands, editor actions, and other actions. You can also specify prefix keys.

This section has these topics:

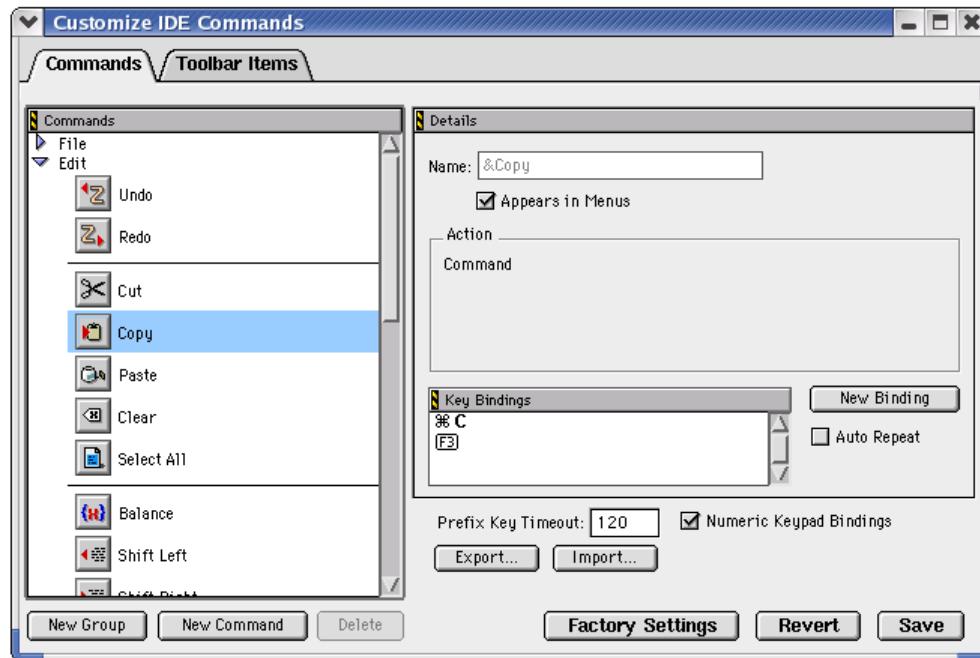
- Modifying key bindings
- Adding key bindings
- Deleting key bindings
- Setting Auto Repeat for key bindings
- Exporting commands and key bindings

Customizing the IDE

Customize Key Bindings

- Importing commands and key bindings
- Quote key prefix

Figure 26.8 Customize IDE Commands—Key Bindings



Adding Key Bindings

Use the Customize IDE Commands window to specify additional key bindings for a particular command.

To add a key binding, follow these steps:

1. From the Commands list, select the command to which you want to add a new key binding.

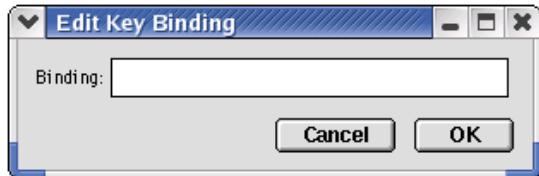
Click the hierarchical controls next to the command categories to expand them as necessary so that you can see individual commands. Select the individual command you wish to modify.

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 **New Binding**.

After clicking this button, the **Edit Key Binding** dialog box appears.

Figure 26.9 Edit Key Bindings



3. Create the key combination you would like to use for the selected command.

For example, to add the key combination Ctrl-8, hold down the **Ctrl** key and press the **8** key, then release both keys at the same time.

If you decide against the key combination that you just entered, or if you make a mistake, click **Cancel** in the **Edit Key Binding** dialog box. The IDE discards changes and returns you to the Customize IDE Commands window.

4. Click **OK** in the Edit Key Binding dialog box.

The new key binding appears in the Key Bindings list in the Customize IDE Commands window.

5. Click **Save** in the Customize IDE Commands window to save your changes.

The new key binding is now available for use in the IDE.

Exporting Commands and Key Bindings

You can export to a file the custom commands and key bindings that you use with the IDE. You can then import the file into another IDE running on a different computer in order to transfer all of your custom commands and key bindings. This process simplifies your setup on the other computer because you do not have to recreate your custom commands and key bindings manually.

Customizing the IDE

Customize Key Bindings

NOTE After you import your custom commands and key bindings into another computer, the IDE running on that computer first sets all its commands and key bindings to their default values, then imports your custom commands and key bindings.

To export your custom commands and key bindings, follow these steps:

1. Click **Export** in the Customize IDE Commands window.

After you click this button, a standard **Save** dialog box appears.

2. Select a location in which to save the `Commands&KeyBindings.mkb` file.

This file contains information about your custom commands and key bindings.

3. Click **Save**.

The IDE saves the `Commands&KeyBindings.mkb` file at the selected location.

TIP You can rename the `Commands&KeyBindings.mkb` file, but remember to preserve the `.mkb` extension. This extension identifies the file as a Metrowerks Key Bindings file. Furthermore, the Windows-hosted version of the CodeWarrior IDE uses this extension to properly recognize the commands and key bindings file.

Importing Commands and Key Bindings

You can import custom commands and key bindings from a previously exported file. `Commands&KeyBindings.mkb` is the default name of an exported file for custom commands and key bindings.

NOTE After you import your custom commands and key bindings into another computer, the IDE running on that computer first sets all its commands and key bindings to their default values, then imports your custom commands and key bindings.

To import commands and key bindings, follow these steps:

1. Click **Import** in the Customize IDE Commands window.

After you click this button, a standard **Open** dialog box appears.

2. Use the dialog box to find and open the `Commands&KeyBindings.mkb` file that you want to import.

The IDE adds the custom commands and key bindings to the Customize IDE Commands window.

Quote Key prefix

The Quote Key is a special prefix key that lets you use any character (such as a-z) as a command key without a modifier key, and still retain the ability to use that character normally, as in editor windows.

In typical use, a key equivalent involves two keys: a modifier key (such as the Ctrl key) combined with a printing key. However, the IDE does not require a modifier key.

For example, you can assign the **2** key (with no modifier) to a command. If you make this assignment, you can no longer type a 2 into your source code in the editor. This conflict occurs because the IDE now interprets the 2 as a command key instead of a printing key. The Quote Key prefix provides the solution to such conflicts.

You can configure the IDE to recognize any key as the Quote Key prefix. Despite its name, the Quote Key prefix does not have to be the key that creates the quote character ("").

After typing an assigned Quote Key prefix, the IDE interprets the next keypress as a keystroke, not as a command.

Returning to the earlier example, assume that you assign the 2 key to a command and the tilde key (~) to be your Quote Key prefix. To execute the command, you would type the 2 key. To enter the character 2 into source code, you would type the tilde key first, then the 2 key. To enter the tilde character into source code, you would press the tilde key twice.

WARNING! The Quote Key only affects the next key or key combination that you type. You must use the Quote Key once for each bound key or key combination for which you want to type.

Assigning the Quote Key prefix

To assign the Quote Key prefix:

1. Click the expand control next to the Miscellaneous command group.
Miscellaneous is part of the **Commands** list in the Customize IDE Commands window.
2. Select the **Quote Key** item.

NOTE If you want to use the numeric keypad as part of the new key binding, enable the **Numeric Keypad Bindings** checkbox in the Customize IDE Commands window.

3. Click **New Binding** to display the Edit Key Bindings dialog box.
4. Type the desired Quote Key prefix.

The keys you type appear in the dialog box. If you make a mistake or decide against the keys you typed, click **Cancel** to return to 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.

Working with IDE Preferences

This chapter explains core CodeWarrior™ IDE preference panels and provides basic information on global- and project-level preference options. Consult the *Targeting* documentation for information on platform-specific preference panels.

This chapter contains these sections:

- [“IDE Preferences Window” on page 385](#)
- [“General Panels” on page 387](#)
- [“Editor Panels” on page 399](#)
- [“Debugger Panels” on page 412](#)

Abbreviated descriptions appear in this chapter. See [“Preference and Target Settings Options” on page 449](#) for more information on preference-panel options.

IDE Preferences Window

The **IDE Preferences** window lists global IDE options. These preferences, unless superseded by a Target Settings option, apply to every open project file. Select **Edit > Preferences** to open the IDE Preferences window.

The IDE Preferences window lists preferences by group:

- **General**—configures overall IDE preferences, such as project builds, recent items, and third-party tools
- **Editor**—configures editor preferences, such as fonts, tabs, and syntax coloring
- **Debugger**—configures debugger preferences, such as window hiding during debugging sessions, low-level interactions, and variable highlighting

Working with IDE Preferences

IDE Preferences Window

Figure 27.1 IDE Preferences window

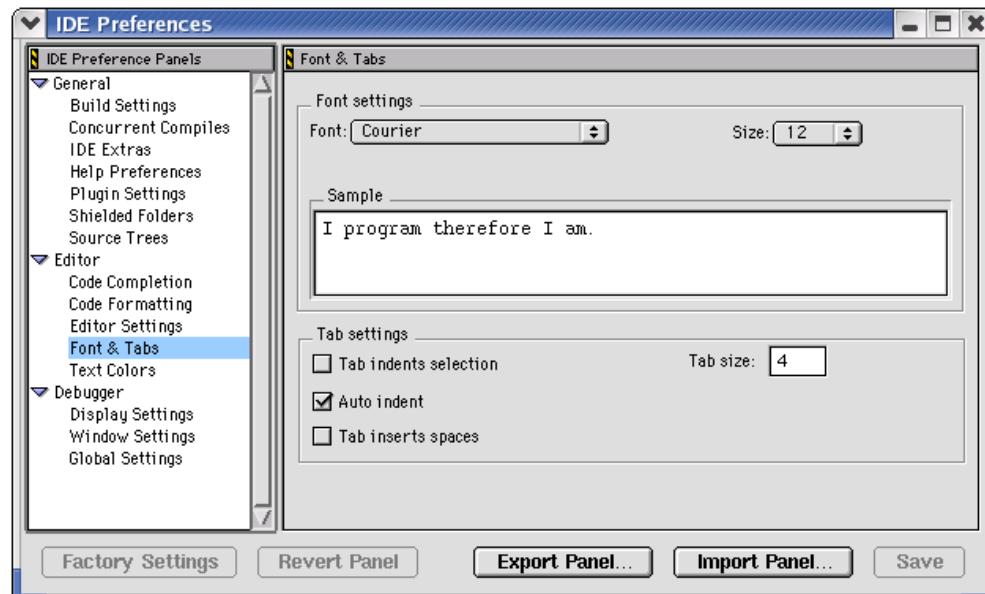


Table 27.1 IDE Preferences window

Item	Explanation
IDE Preference Panels list	Lists preference panels, organized by group. Click the hierarchical control next to a group name to show or hide individual preference panels.
Preference panel	Shows options for the selected item in the IDE Preference Panels list.
Factory Settings	Click to restore the default options for the current preference panel.
Revert Panel	Click to restore the most recently saved options for the current preference panel.
Export Panel	Click to save an XML file that contains options for the current preference panel.
Import Panel	Click to open an XML file that contains options for the current preference panel.

Table 27.1 IDE Preferences window (*continued*)

Item	Explanation
OK (Windows)	Click to save modifications to all preference panels and close the window.
Cancel (Windows)	Click to discard modifications to all preference panels and close the window.
Apply (Windows)	Click to confirm modifications to all preference panels.
Save (Macintosh, Solaris, and Linux)	Click to save modifications to all preference panels.

General Panels

The **General** section of the IDE Preference Panels defines basic options assigned to a new project.

The General preference panels available on most IDE hosts include:

- [“Build Settings” on page 387](#)
- [“Concurrent Compiles” on page 389](#)
- [“IDE Extras” on page 390](#)
- [“Help Preferences” on page 393](#)
- [“Plugin Settings” on page 394](#)
- [“Shielded Folders” on page 394](#)
- [“Source Trees” on page 396](#)

Build Settings

The **Build Settings** preference panel provides options for customizing various aspects of project builds, including:

- file actions during project builds
- memory usage to accelerate builds
- local data storage of projects stored on read-only volumes

Working with IDE Preferences

General Panels

Figure 27.2 Build Settings preference panel

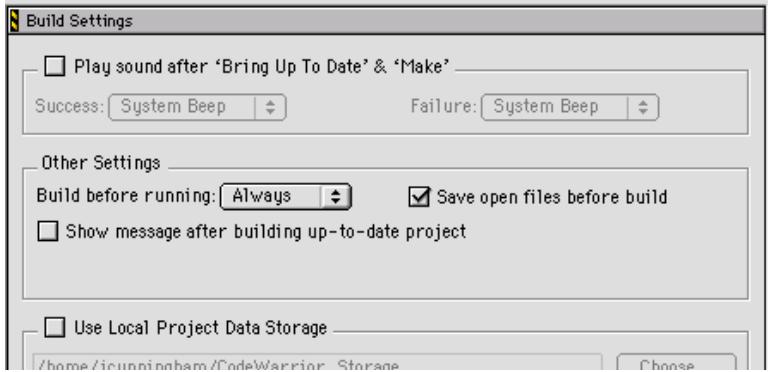


Table 27.2 Build Settings preference panel

Item	Explanation
Play sound after 'Bring Up To Date' & 'Make' (Macintosh, Solaris, and Linux)	Select to have the IDE play an alert sound after completing a Bring Up To Date or Make command.
Success (Macintosh, Solaris, and Linux)	Choose the sound the IDE plays after successfully completing a Bring Up To Date or Make command.
Failure (Macintosh, Solaris, and Linux)	Choose the sound the IDE plays after failing to complete a Bring Up To Date or Make command.
Build before running	Choose to always build the project before running it, never build the project before running it, or ask for the desired action.
Save open files before build	Select to automatically save the contents of all editor windows before starting a build.
Show message after building up-to-date project	Select to have the IDE display a message after successfully building a project.
Include file cache (Macintosh)	Enter the kilobytes of memory to allocate to the file cache used for #include files during a project build.

Table 27.2 Build Settings preference panel (*continued*)

Item	Explanation
Compiler thread stack (Windows and Macintosh)	Enter the kilobytes of memory to allocate to the stack for execution of the IDE compiler thread. Increase the size when compiling heavily optimized code.
Use Local Project Data Storage	Select to specify a location to save project data if the project is on a read-only volume. Click Choose to select the location.

Concurrent Compiles

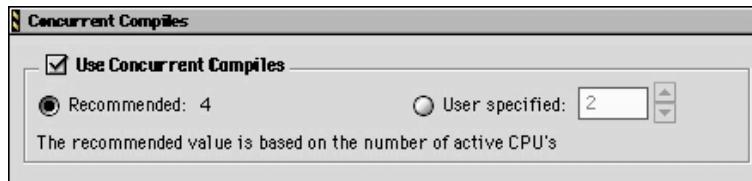
The **Concurrent Compiles** preference panel controls execution of simultaneous IDE compilation processes. The IDE lists this panel in the IDE Preference Panels list when the active compiler supports concurrency.

The IDE uses concurrent compiles to compile code more efficiently. The IDE improves its use of available processor capacity by spawning multiple compile processes, which allow the operating system to perform these tasks as needed:

- optimize resource use
- use overlapped input/output

For those compilers that support concurrency, concurrent compiles improve compile time on both single- and multiple-processor systems.

Figure 27.3 Concurrent Compiles preference panel



Working with IDE Preferences

General Panels

Table 27.3 Concurrent Compiles preference panel

Item	Explanation
Use Concurrent Compiles	Select to have the IDE run multiple compilation processes simultaneously.
Recommended	Select to allow the number of concurrent compiles suggested by the IDE.
User Specified	Select to stipulate the number of concurrent compiles.

IDE Extras

The **IDE Extras** preference panel provides options for customizing various aspects of the IDE, including:

- menu-bar layout
- the number of recent projects, document files, and symbolics files to remember
- use of a third-party editor

Figure 27.4 IDE Extras preference panel

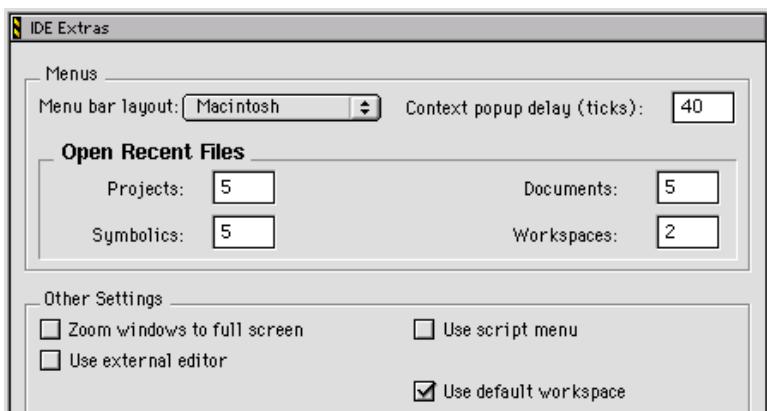


Table 27.4 IDE Extras preference panel

Item	Explanation
Menu bar layout	Choose a layout that organizes IDE menus into a typical host-platform menu bar. Restart the IDE in order for menu-bar layout changes to take effect.
Projects	Enter the number of recently opened projects for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.
Documents	Enter the number of recently opened documents for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.
Symbolics	Enter the number of recently opened symbolics files for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.
Workspaces	Enter the number of recently opened workspaces for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.
Context popup delay (Macintosh, Solaris, and Linux)	Enter the number of ticks to wait before displaying contextual menus. A tick is 1/60 of a second.
Use Third Party Editor (Windows)	Select to use a third-party text editor to edit source files.
Launch Editor (Windows)	Enter a command-line expression that runs the desired third-party text editor.
Launch Editor w/ Line # (Windows)	Enter a command-line expression that runs the desired third-party text editor and passes to that editor an initial line of text to display.
Use Multiple Document Interface (Windows)	Select to have the IDE use the Multiple Document Interface (MDI). Clear to have the IDE use the Floating Document Interface (FDI). Restart the IDE in order for interface changes to take effect.
Zoom windows to full screen (Macintosh, Solaris, and Linux)	Select to have zoomed windows fill the entire screen. Clear to have zoomed windows in a default size.

Working with IDE Preferences

General Panels

Table 27.4 IDE Extras preference panel (*continued*)

Item	Explanation
Use Script menu (Macintosh, Solaris, and Linux)	Select to display the Scripts menu in the menu bar. Clear to remove the Scripts menu from the menu bar.
Use External Editor (Macintosh, Solaris, and Linux)	Select to use a third-party text editor to edit text files in the current project. Clear to use the editor included with the IDE.
Use ToolServer menu (Classic Macintosh)	Select to display the ToolServer menu in the menu bar. Clear to remove the ToolServer menu from the menu bar.
Enable automatic Toolbar help (Classic Macintosh)	Select to display Balloon Help after resting the cursor over a toolbar icon. Clear to prevent Balloon Help from appearing.
Use default workspace	Select this option to have the IDE use the default workspace to save and restore state information. Clear this option to have the IDE always start in the same state.
Find Reference using (Macintosh)	Choose an online browser application to view reference information and definitions.
Show Code and Data Sizes (Windows)	Displays or hides Code and Data columns in project manager.

Using an External Editor on the Macintosh

To use an external editor on the Macintosh, the IDE sends AppleEvents to an alias file that points to the editor application. Manually configure the IDE to use an external editor.

1. Choose **Edit > Preferences**.

The IDE Preferences window appears.

2. Select the **IDE Extras** panel from the **IDE Preference Panels** list.
3. Select the **Use External Editor** option.
4. Click **Save**.

The IDE is now prepared to use an external editor application. To specify the external editor to use:

1. Find and open the `CodeWarrior` folder.
2. Create a folder named `(Helper Apps)` inside the `CodeWarrior` folder (if it does not already exist).
3. Make an alias of the desired editor application.
4. Place the alias file inside the `(Helper Apps)` folder.
5. Rename the alias file `External Editor`.
6. Restart the IDE in order for changes to take effect.

The IDE now uses the aliased external editor.

Help Preferences

The **Help Preferences** panel, available on the Solaris and Linux IDE hosts, specifies the browser used for viewing IDE online help.

Figure 27.5 Help Preferences panel



Table 27.5 Help Preferences panel

Item	Explanation
Browser Path	Enter a path to the browser to use for viewing IDE online help. Alternatively, use the Choose... button.
Choose...	Click to select the path to the browser to use for viewing IDE online help.

Working with IDE Preferences

General Panels

Plugin Settings

The **Plugin Settings** preference panel contains options for troubleshooting third-party IDE plug-ins.

Figure 27.6 Plugin Settings preference panel

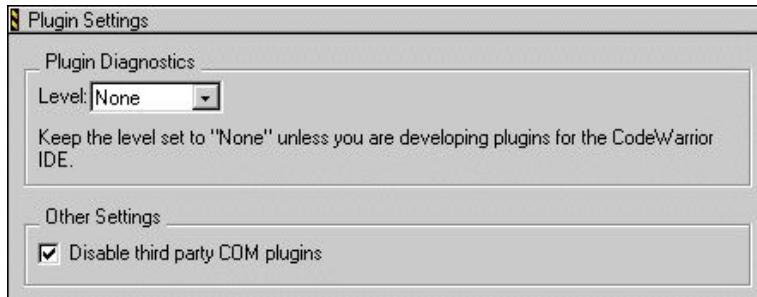


Table 27.6 Plugin Settings preference panel

Item	Explanation
Level	Choose the plug-in diagnostics level the IDE generates the next time it loads plug-ins. Restart the IDE in order for diagnostic-level changes to take effect. Options are None, Errors Only, and All Info.
Disable third party COM plugins	Select to prevent the IDE from loading third-party Common Object Model (COM) plug-ins.

Shielded Folders

The **Shielded Folder** preference panel enables the IDE to ignore specified folders during project operations and find-and-compare operations. The IDE ignores folders based on matching names with regular expressions defined in the preference panel.

NOTE If the **Access Paths** settings panel in the Target Settings window contains a path to a shielded folder, the IDE overrides the shielding and includes the folder in project operations and find-and-compare operations.

Figure 27.7 Shielded Folders preference panel

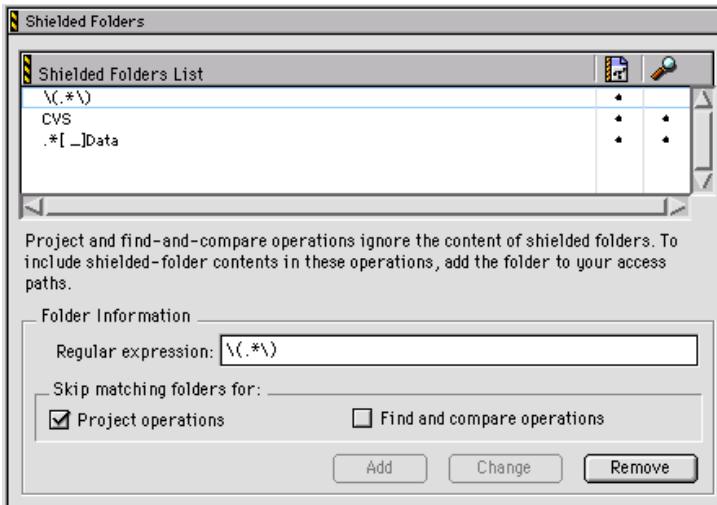


Table 27.7 Shielded Folders preference panel

Item	Icon	Explanation
Shielded folder list		Lists folders that match the specified regular expression. The IDE skips these folders during project operations, find-and-compare operations, or both.
Regular Expression		Enter the regular expression used to shield folders from selected operations.
Project operations		Select to have the IDE skip folders during project operations. A bullet appears in the corresponding column of the shielded folder list.
Find and compare operations		Select to have the IDE skip folders during find-and-compare operations. A bullet appears in the corresponding column of the shielded folder list.

Working with IDE Preferences

General Panels

Table 27.7 Shielded Folders preference panel (*continued*)

Item	Icon	Explanation
Add		Click to add the current Regular Expression field entry to the shielded folder list.
Change		Click to replace the selected regular expression in the shielded folder list with the current Regular Expression field entry.
Remove		Click to delete the selected regular expression from the shielded folder list.

Table 27.8 Default regular expressions in Shielded Folders panel

Regular Expression	Explanation
\(.*\)	Matches folders with names that begin and end with parentheses, such as the (Project Stationery) folder.
CVS	Matches folders named CVS. With this regular expression, the IDE skips Concurrent Versions System (CVS) data files.
. * [__] Data	Matches the names of folders generated by the IDE that store target data information, such as a folder named MyProject_Data.

Source Trees

Use the **Source Trees** panel to add, modify, and remove source trees (root paths) used in projects. Use source trees to define common access paths and build-target outputs to promote sharing of projects across different hosts. Source trees have these scopes:

- *Global source trees*, defined in the IDE Preferences window, apply to all projects.
- *Project source trees*, defined in the Target Settings window for a particular project, apply only to files in that project. Project source trees always take precedence over global source trees.

Except for the difference in scope, global and project source trees operate identically.

Figure 27.8 Source Trees panel

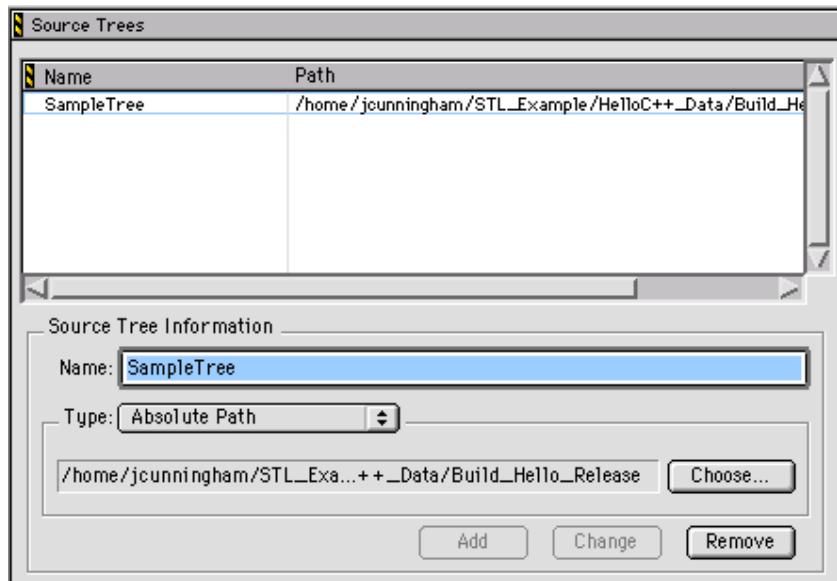


Table 27.9 Source Trees panel

Item	Explanation
Source Tree list	Contains the Name and Path of currently defined source trees.
Name	Enter a name for a new source tree or modify the name of a selected source tree.
Type	Choose the source-tree path type.
Choose	Click to select or modify a source-tree path.
Add	Click to add a new source-tree path to the Source Tree list.
Change	Click to modify the selected source-tree name or path.
Remove	Click to delete the selected source tree from the Source Tree list.

Adding Source Trees

Add source trees that define root paths for access paths and build-target output.

1. Choose **Edit > Preferences**.

The IDE Preferences window appears.

2. Select the **Source Trees** panel from the **IDE Preference Panels** list.

3. Enter in the **Name** field a name for the new source tree.

4. Choose the source tree **Type**:

- **Absolute Path**—defines a path from the root level of the hard drive to a desired folder, including all intermediate folders
- **Environment Variable**—(Windows, Solaris, and Linux) defines an environment variable in the operating environment
- **Registry Key**—(Windows) defines a key entry in the operating-environment registry

5. Enter the source-tree definition:

- For **Absolute Path**—Click **Choose** to display a subordinate dialog box. Use the dialog box to select the desired folder. The absolute path to the selected folder appears in the **Source Trees** preference panel.
- For **Environment Variable**—Enter the path to the desired environment variable.
- For **Registry Key**—Enter the path to the desired key entry in the registry.

6. Click **Add**.

The IDE adds the new source tree to the **Source Trees** list.

7. Click **OK**, **Apply**, or **Save**.

The IDE saves the source-tree changes.

Changing Source Trees

Change a source tree to update path information for a project. The IDE must be able to resolve source trees before building the project.

1. Choose **Edit > Preferences**.

2. Select the **Source Trees** panel from the **IDE Preference Panels** list.
3. Select the desired source tree in the **Source Trees** list.
4. If needed, enter a new name for the selected source tree.
5. If needed, choose a new path type for the selected source tree.
6. Click **Change**.

The IDE updates the source tree and displays changes in the **Source Trees** list. A reminder message to update source-tree references in the project appears.

7. Click **OK**, **Apply**, or **Save**.

The IDE saves the source-tree changes.

Removing Source Trees

Remove source trees that the project no longer uses. The IDE must be able to find the remaining source trees before building the project.

1. Choose **Edit > Preferences**.
2. Select the **Source Trees** panel from the **IDE Preference Panels** list.
3. Select the source tree from the **Source Trees** list.
4. Click **Remove**.

The IDE updates the **Source Trees** list. A reminder message to update source-tree references in the project appears.

5. Click **OK**, **Apply**, or **Save**.

The IDE saves the source-tree changes.

Editor Panels

The **Editor** section of the IDE Preference Panels list defines the editor settings assigned to a new project.

The Editor preference panels available on most IDE hosts include:

- [“Code Completion” on page 400](#)
- [“Code Formatting” on page 401](#)

Working with IDE Preferences

Editor Panels

- [“Editor Settings” on page 403](#)
- [“Font & Tabs” on page 406](#)
- [“Text Colors” on page 408](#)

Code Completion

The **Code Completion** preference panel provides options for customizing the IDE code-completion behavior, including:

- automatic invocation and indexing
- window positioning and appearance delay
- case sensitivity

Figure 27.9 Code Completion preference panel

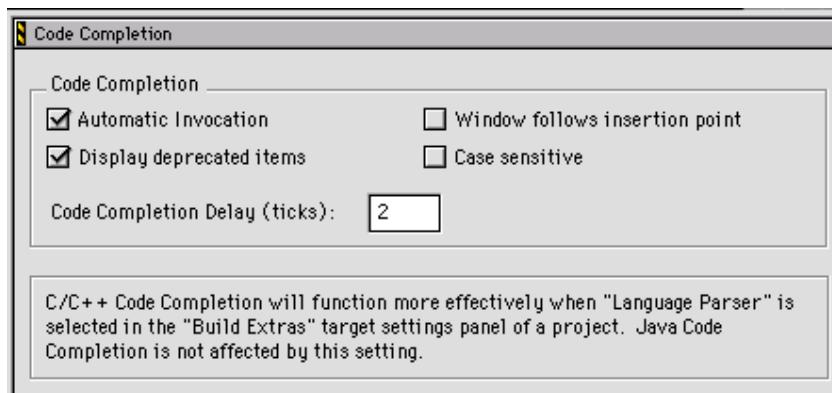


Table 27.10 Code Completion preference panel

Item	Explanation
Automatic Invocation	Select to automatically open the Code Completion window to complete programming-language symbols. Clear to manually open the window.
Window follows insertion point	Select to have the Code Completion window follow the insertion point as you edit text. Clear to leave the window in place.

Table 27.10 Code Completion preference panel (*continued*)

Item	Explanation
Display deprecated items	Select to have the Code Completion window display obsolete items in gray text. Clear to have the window hide obsolete items.
Case sensitive	Select to have the IDE consider case when completing code. Clear to have the IDE ignore case.
Code Completion Delay (ticks)	Enter the number of ticks to wait before opening the Code Completion window. A tick is 1/60 of a second.

Code Formatting

The **Code Formatting** preference panel provides options for customizing editor code-formatting behavior, including:

- indenting
- syntax placement
- brace handling

Working with IDE Preferences

Editor Panels

Figure 27.10 Code Formatting preference panel

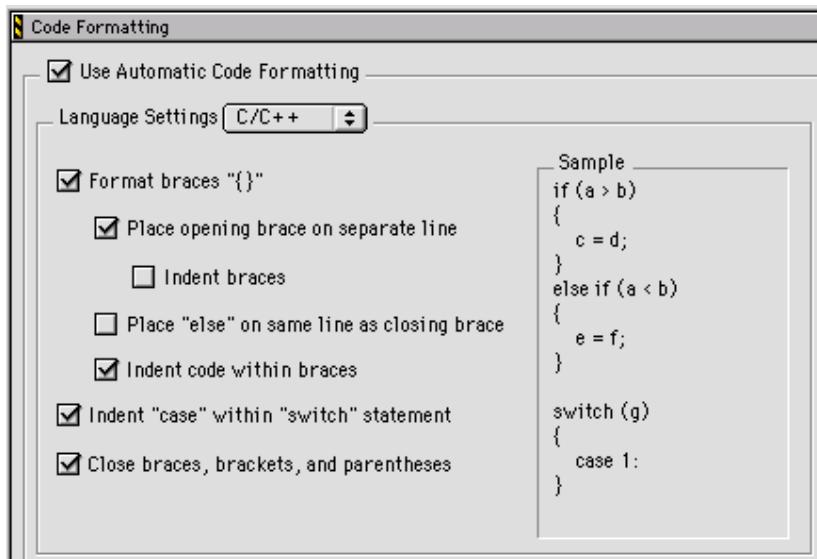


Table 27.11 Code Formatting preference panel

Item	Explanation
Use Automatic Code Formatting	Check to have the editor automatically format your source code according to settings in this panel. Clear to prevent the editor from automatically formatting your code.
Language Settings	Use to specify the language type that you want to format. Your selection changes the other options in this panel to their default states for the selected language.
Format braces	Check to have the editor automatically insert a closing brace when you type an opening brace. The editor places the cursor between the opening brace that you typed and the closing brace that it inserts. Clear to prevent the editor from automatically inserting a closing brace when you type an opening brace.

Table 27.11 Code Formatting preference panel (*continued*)

Item	Explanation
Place opening brace on separate line	Check to have the editor place on the next line an opening brace that you type. Clear to prevent the editor from placing on the next line an opening brace that you type.
Indent braces	Check to have the editor indent braces by one tab stop from the previous line. Clear to prevent the editor from indenting braces by one tab stop from the previous line.
Place “else” on same line as closing brace	Check to have the editor place <code>else</code> and <code>else if</code> text on the same line as the closing brace of the <code>if</code> or <code>else if</code> statement. Clear to prevent the editor from placing <code>else</code> and <code>else if</code> text on the same line as the closing brace of the <code>if</code> or <code>else if</code> statement.
Indent code within braces	Check to have the editor indent code by one tab stop from the braces. Clear to prevent the editor from indenting code by one tab stop from the braces.
Indent “case” within “switch” statement	Check to have the editor indent <code>case</code> statements by one tab stop inside a <code>switch</code> statement. Clear to prevent the editor from indenting <code>case</code> statements by one tab stop inside a <code>switch</code> statement.
Close braces, brackets, and parentheses	Check to have the editor automatically insert the corresponding closing character when you type an opening brace, bracket, or parenthesis. The editor places the cursor between the opening character and the closing character. Clear to prevent the editor from automatically inserting the corresponding closing character when you type an opening brace, bracket, or parenthesis.

Editor Settings

The **Editor Settings** preference panel provides options for customizing the editor, including:

- fonts, window locations, and insertion-point positions

Working with IDE Preferences

Editor Panels

- contextual menus
- additional editor-window features

Figure 27.11 Editor Settings preference panel

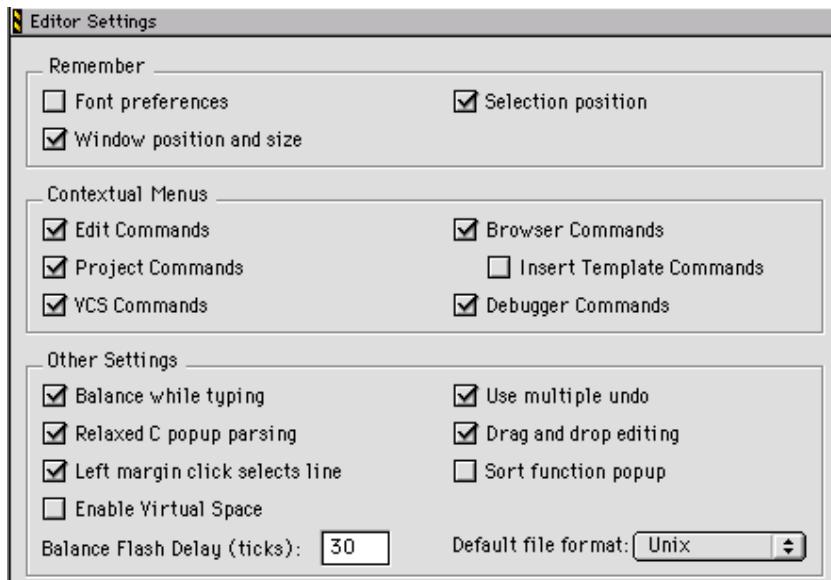


Table 27.12 Editor Settings preference panel

Item	Explanation
Font preferences	Select to retain font settings for each source file. Clear to apply default font settings each time the IDE displays the source file.
Selection position	Select to retain the text-insertion position in each source file.
Window position and size	Select to retain the location and dimensions of each editor window.
Edit Commands	Select to add Edit menu commands to contextual menus.

Table 27.12 Editor Settings preference panel (*continued*)

Item	Explanation
Browser Commands	Select to add Browser menu commands to contextual menus. Also select in order to use the Insert Template Commands option.
Insert Template Commands (Macintosh)	Select to add the Insert Template submenu to contextual menus. The submenu displays source-defined function templates.
Project Commands	Select to add Project menu commands to contextual menus.
VCS Commands	Select to add VCS (Version Control System) menu commands to contextual menus.
Debugger Commands	Select to add Debug menu commands to contextual menus.
Balance while typing	Select to flash the matching (, [, or { after typing),], or } in an editor window.
Use multiple undo	Select to allow multiple undo and redo operations while editing text.
Relaxed C popup parsing	Select to allow the C parser to recognize some non-standard function formats and avoid skipping or misinterpreting some definition styles.
Drag and drop editing	Select to allow drag-and-drop text editing.
Left margin click selects line	Select to allow selection of an entire line of text by clicking in the left margin of the editor window.
Sort function popup	Select to sort function names by alphabetical order in menus. Clear to sort function names by order of appearance in the source file.
Enable Virtual Space (Windows and Macintosh)	Select to allow moving the text-insertion point beyond the end of a source-code line. Entering new text automatically inserts spaces between the former end of the line and the newly entered text.
Balance Flash Delay	Enter the number of ticks to flash a balancing punctuation character. A tick is 1/60 of a second.
Default file format	Choose the default end-of-line format used to save files.

Working with IDE Preferences

Editor Panels

Font & Tabs

The **Font & Tabs** preference panel provides options for customizing settings used by the editor, including:

- font and font size used in editor windows
- auto indentation and tab size
- tabs on selections and replacing tabs with spaces

Figure 27.12 Font & Tabs preference panel

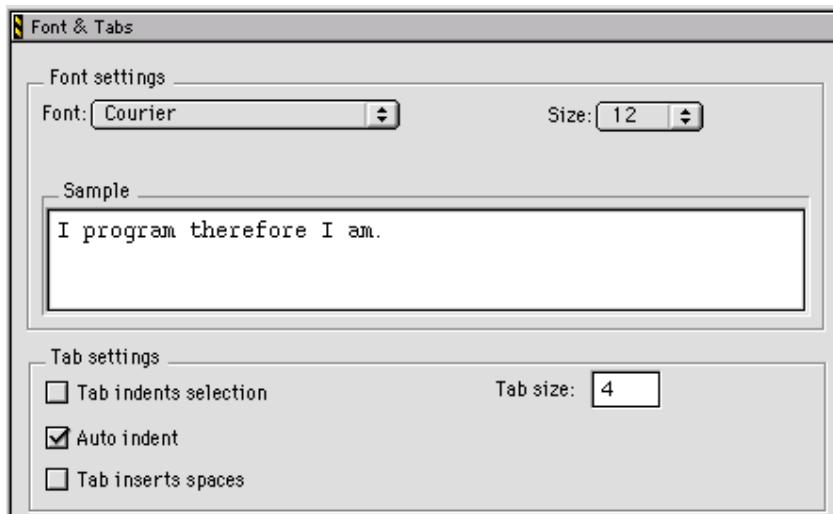


Table 27.13 Font & Tabs preference panel

Item	Explanation
Font	Choose the typeface displayed in editor windows.
Size	Choose the font size displayed in editor windows.
Script (Windows)	Choose the IDE script system. The script system maps keyboard keys to characters of an alphabet.
Tab indents selection	Select to indent each line of selected text after pressing Tab. Clear to replace selected text with a tab character after pressing Tab.

Table 27.13 Font & Tabs preference panel (*continued*)

Item	Explanation
Tab Size	Enter the number of spaces to substitute in place of a tab character. This number applies to the Tab Inserts Spaces option.
Auto Indent	Select to automatically apply the indentation level from the previous line of text to each new line created by pressing Enter or Return.
Tab Inserts Spaces	Select to insert spaces instead of a tab character after pressing Tab. The Tab Size option determines the number of inserted spaces.

Setting the Text Font

To set the text font, follow these steps:

1. Choose **Edit > Preferences**.
2. Select the **Font & Tabs** panel in the **Editor** group in the **IDE Preference Panels** list.
3. In the **Font Settings** area of the **IDE Preferences** window, select a font type in the drop-down menu in the **Font** field.
4. Save your font in the **IDE Preferences** window.
 - Windows: Click **OK**.
 - Macintosh/Linux/Solaris: Click **Save**.

The foreground text changes to the new font.

Setting the Text Size

To set the text size, follow these steps:

1. Choose **Edit > Preferences**.
2. Select the **Font & Tabs** panel in the **Editor** group in the **IDE Preference Panels** list.

Working with IDE Preferences

Editor Panels

3. In the **Font Settings** area of the IDE Preferences window, select the **Size** drop-down menu and choose a text point size (from 2 to 24 points).
4. Save your text size in the **IDE Preferences** window.
 - Windows: Click **OK**.
 - Macintosh/Linux/Solaris: Click **Save**.

The foreground text changes to the new size.

Text Colors

The **Text Colors** preference panel customizes colors applied to elements of source code displayed in editor windows, such as:

- default foreground and background in editor windows
- standard comments, keywords, and strings in source code
- custom-defined keywords
- browser symbols

Default settings provide a simple scheme of at least four source-code colors. If four colors do not provide sufficient detail, modify this preference panel to create more sophisticated color schemes.

Figure 27.13 Text Colors preference panel

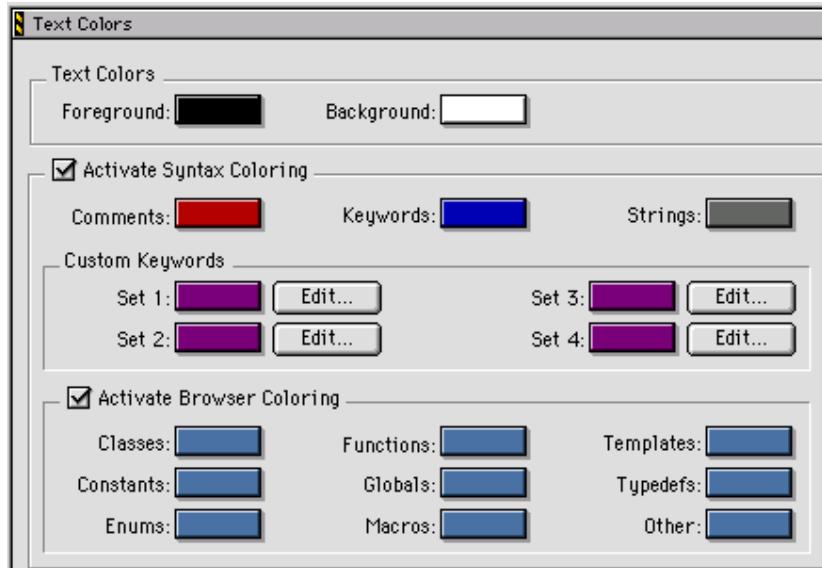


Table 27.14 Text Colors preference panel

Item	Explanation
<u>Foreground</u>	Click the color swatch to display a dialog box. Use the dialog box to set the foreground color used in editor windows for text.
<u>Background</u>	Click the color swatch to set the background color used in editor windows.
<u>Activate Syntax Coloring</u>	Select to apply custom colors to comments, keywords, strings, and custom keywords in text. Clear to use the Foreground color for all text.
<u>Comments</u>	Click the color swatch to set the color used for source-code comments.
<u>Keywords</u>	Click the color swatch to set the color used for source-code language keywords.

Working with IDE Preferences

Editor Panels

Table 27.14 Text Colors preference panel (*continued*)

Item	Explanation
Strings	Click the color swatch to set the color used for source-code string literals.
Set 1, Set 2, Set 3, Set 4	Click a color swatch to set the color used for the corresponding custom-keyword set.
Edit	Click to add, modify, or remove keywords from the corresponding custom-keyword set.
Activate Browser Coloring	Select to apply custom colors to browser symbols in text. Clear to use the Foreground color for all text.
Classes	Click the color swatch to set the color used for source-code classes.
Constants	Click the color swatch to set the color used for source-code constants.
Enums	Click the color swatch to set the color used for source-code enumerations.
Functions	Click the color swatch to set the color used for source-code functions.
Globals	Click the color swatch to set the color used for source-code global variables.
Macros	Click the color swatch to set the color used for source-code macros.
Templates	Click the color swatch to set the color used for source-code templates.
TypeDefs	Click the color swatch to set the color used for source-code type definitions.
Other	Click the color swatch to set the color used for other symbols not specified in the Activate Browser Coloring section.

Setting the Foreground Text Color

Use the **Foreground Color** option to configure the foreground text color displayed in editor windows.

1. Choose **Edit > Preferences**.
2. Select the **Text Colors** panel in the **Editor** group in the **IDE Preference Panels** list.
3. Click the **Foreground** color box to set the editor's foreground color.
4. Pick color.
5. Click **OK** in the **Color Picker** window.
6. Click **OK** or **Save**

The foreground text color changes to the new color.

Setting the Background Text Color

Use the **Background Color** option to configure the background color displayed by all editor windows.

1. Choose **Edit > Preferences**.
2. Select the **Text Colors** panel in the **Editor** group in the **IDE Preference Panels** list.
3. Click the **Background** color box to set the editor's background color.
4. Pick color.
5. Click **OK** in the **Color Picker** window.
6. Click **OK** or **Save**

The background text color changes to the new color.

Activate Syntax and Browser Coloring

Use the **Activate Syntax Coloring** and **Activate Browser Coloring** options to configure the syntax and browser colors that all editor windows display.

1. Choose **Edit > Preferences**.

Working with IDE Preferences

Debugger Panels

2. Select the **Text Colors** panel in the **Editor** group in the **IDE Preference Panels** list.
3. Select the checkbox next to the **Activate Syntax Coloring** or the **Activate Browser Coloring** option.
4. Click on the colored box next to the option.
5. Pick color.
6. Click **OK** in the **Color Picker** window.
7. Click **OK** or **Save**

Debugger Panels

The **Debugger** section of the IDE Preference Panels defines the basic debugger settings assigned to a new project.

The Debugger preference panels available on most IDE hosts include:

- [“Display Settings” on page 412](#)
- [“Window Settings” on page 414](#)
- [“Global Settings” on page 416](#)
- [“Remote Connections” on page 418](#)

Display Settings

The **Display Settings** preference panel provides options for customizing various aspects of the IDE Debugger, including:

- assignment of colors to changed variables and breakpoints
- viewing variable types
- displaying local variables
- using decimal values
- sorting functions
- using dynamic objects

[Figure 27.14 on page 413](#) shows the Display Settings preference panel. [Table 27.15 on page 413](#) explains the items in the preference panel.

Figure 27.14 Display Settings preference panel

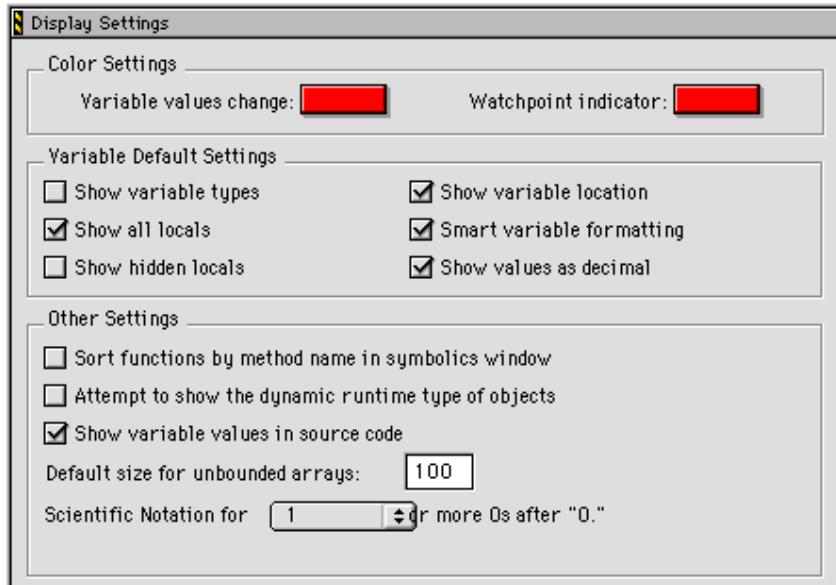


Table 27.15 Display Settings preference panel

Item	Explanation
Variable values change	Click the color swatch to set the color that indicates a changed variable value.
Watchpoint indicator	Click the color swatch to set the color that indicates a changed watchpoint value.
Show variable types	Select to always show the type associated with each variable.
Show variable location	Select to display the Location column in the Variables pane of the Thread window.
Show all locals	Select to show all local variables. Clear to have the debugger show only variables near the program counter.

Working with IDE Preferences

Debugger Panels

Table 27.15 Display Settings preference panel (*continued*)

Item	Explanation
Smart Variable Formatting	Controls whether variables in variable windows, panes and expression displays are formatted using entries in XML files located in the VariableFormats support folder; such as the Windows_Formats.xml file.
Show hidden locals	Select to show hidden local variables. A hidden local variable is a variable that is in scope, but is hidden by a variable of the same name in a deeper scope.
Show values as decimal	Select to always show decimal values instead of hexadecimal values.
Sort functions by method name in symbolics window	Select to sort functions of the form className : :methodName in the Symbolics window by methodName. Clear to sort by className.
Attempt to show the dynamic runtime type of objects	Select to attempt to display the runtime type of the specified language objects. Clear to display the static type.
Show variable values in source code	Select to show variable values in contextual menus in the source code.
Default size for unbounded arrays	Enter the default number of unbounded array elements to display in a View Array window.
Scientific Notation for (1-9) or more 0s after “0.”	Maximum number of zeros after a decimal point in a float value such as 0.034. For example, a value of 2 means 0.00045 will be displayed in scientific notation as 4.5e-4; a value of 3 will be displayed as 0.00045. Does not change value precision, only value display.

Window Settings

The **Window Settings** preference panel provides options for customizing how the debugger displays windows during debugging sessions, including non-debugging and project windows.

Figure 27.15 Window Settings preference panel

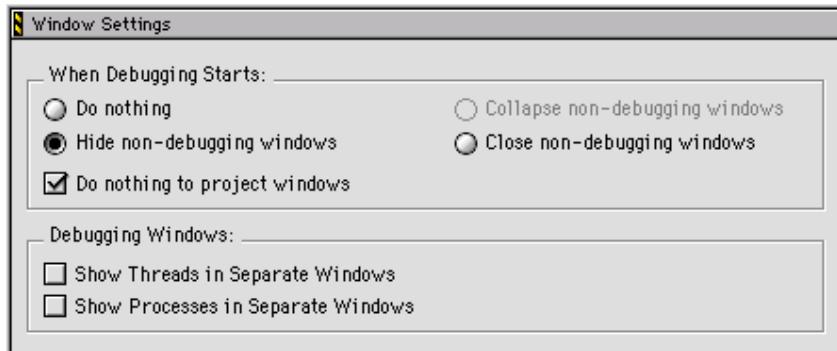


Table 27.16 Window Settings preference panel

Item	Explanation
Do nothing	Select to leave all windows in place when starting a debugging session.
Minimize non-debugging windows (Windows)	Select to minimize all non-debugging windows when starting a debugging session.
Collapse non-debugging windows (Macintosh, Solaris, and Linux)	Select to collapse all non-debugging windows when starting a debugging session.
Hide non-debugging windows	Select to hide, but not close, all non-debugging windows when starting a debugging session.
Close non-debugging windows	Select to close all non-debugging windows, except for the active project window, when starting a debugging session.
Do nothing to project windows	Select to prevent the IDE from hiding project windows when starting a debugging session.
Use Debugging Monitor (Classic Macintosh)	Select to use a second monitor during debugging sessions.
Monitor for debugging (Classic Macintosh)	Choose the monitor to display debugging windows. The coordinates in parentheses identify the selected monitor in the QuickDraw® coordinate space.

Working with IDE Preferences

Debugger Panels

Table 27.16 Window Settings preference panel (*continued*)

Item	Explanation
Move open windows to debugging monitor when debugging starts (Classic Macintosh)	Select to move all open windows to the selected debugging monitor when a debugging session starts.
Open windows on debugging monitor during debugging (Classic Macintosh)	Select to display on the debugging monitor any window opened during a debugging session.
Show threads in separate windows	Select to display threads in separate Thread windows. Clear to show all threads in one window. Restart active debugging sessions in order for changes to take effect.
Show processes in separate windows	Select to display processes in separate windows. Clear to show all processes in one window.

Global Settings

The **Global Settings** preference panel provides options for customizing various global options for the debugger, including:

- file caching to accelerate debugger sessions
- automatic launch of applications and libraries
- confirmation of attempts to close or quit debugging sessions

Figure 27.16 Global Settings preference panel

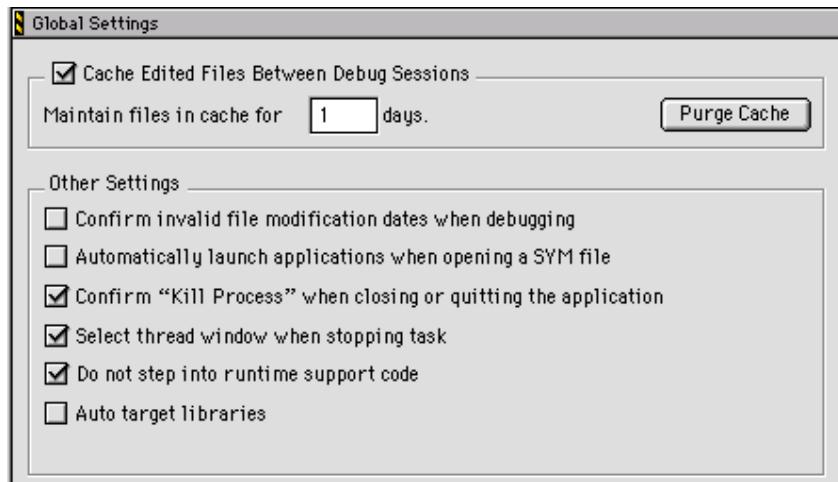


Table 27.17 Global Settings preference panel

Item	Explanation
Cache Edited Files Between Debug Sessions	Select to maintain a cache of modified files between debugging sessions. Use this option to debug through the original source code for files modified since the last build.
Maintain files in cache	Enter the number of days that the IDE maintains its file cache.
Purge Cache	Click to delete the file cache maintained by the IDE, freeing memory and disk space.
Confirm invalid file modification dates when debugging	Select to have the IDE display a warning message when debugging a project with mismatched file modification dates.
Automatically launch applications when SYM file opens	Select to automatically launch the application program associated with an open symbolics file.
Confirm ‘Kill Process’ when closing or quitting	Select to prompt for confirmation before killing processes upon quitting a debugging session.

Working with IDE Preferences

Debugger Panels

Table 27.17 Global Settings preference panel (*continued*)

Item	Explanation
Select stack crawl window when task is stopped	Select to bring forward the Stack Crawl window (also known as the Thread window) after the debugger stops tasks.
Don't step into runtime support code	Select to have the IDE not step into Metrowerks Standard Library (MSL) runtime support code and instead directly step into your own code.
Auto Target Libraries	Select to have the IDE attempt to debug dynamically linked libraries (DLLs) loaded by the target application.

Remote Connections

The **Remote Connections** preference panel configures general network settings for remote-debugging connections between the host computer and other computers. Use these general settings as the basis for defining more specific connections for individual projects in conjunction with the **Remote Debugging** settings panel. The Target Settings window contains the **Remote Debugging** settings panel.

Figure 27.17 Remote Connections preference panel

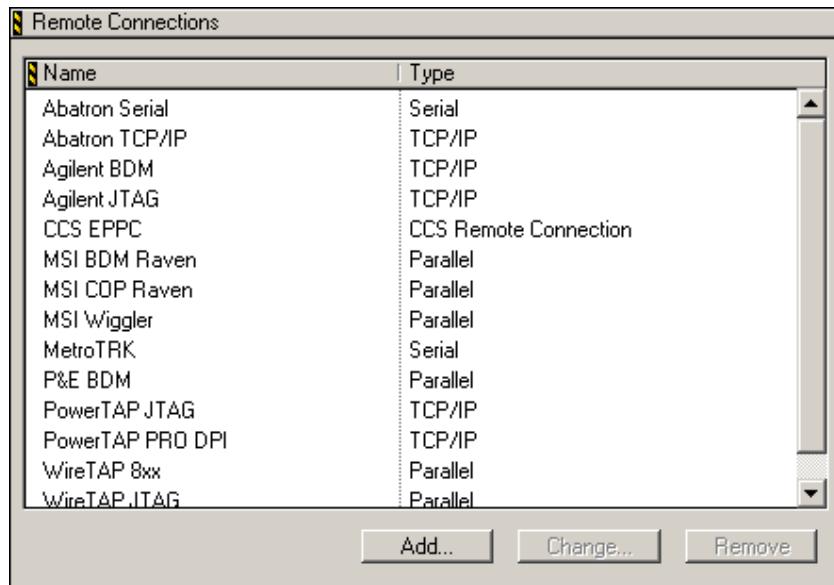


Table 27.18 Remote Connections preference panel

Item	Explanation
Remote Connection list	Displays the name and connection type of all remote connections currently defined.
Add	Click to add a new remote connection to the Remote Connection list.
Change	Click to change the settings of the selected remote connection.
Remove	Click to remove the selected remote connection from the Remote Connection list.

Adding Remote Connections

Add a remote connection that defines a general network connection between the host computer and a remote computer.

1. Choose **Edit > Preferences**.
2. Select the **Remote Connections** panel from the **IDE Preference Panels** list.
3. Click **Add**.

The **New Connection** dialog box appears.

4. Enter the name for the general remote connection.
5. Choose from the **Debugger** pop-up menu the desired debugger for use with the remote connection.
6. Configure the **Browse in processes window** option as desired:
 - selected—the IDE filters the **Processes** window list and the list of available debuggers for an opened symbolics file. The filter prevents an unavailable remote connection from appearing in either list.
 - cleared—the IDE does not filter the **Processes** window list or the list of available debuggers for an opened symbolics file. Both available and unavailable remote connections appear in the lists.

7. Choose from the **Connection Type** pop-up menu the desired network protocol for the remote connection.
8. Enter the Internet Protocol address of the remote computer in the **IP Address** field.
9. Click **OK**.

The IDE adds the new remote connection to the **Remote Connections** list.

10. Click **OK**, **Apply**, or **Save**.

Changing Remote Connections

Change a remote connection to update network-connection information between the host and remote computer.

1. Choose **Edit > Preferences**.

2. Select the **Remote Connections** panel from the **IDE Preference Panels** list.
3. Select from the **Remote Connections** list the remote connection that requires modification.
4. Click **Change**.
A dialog box appears with the current network settings for the selected remote connection.
5. If needed, enter a new name for the general remote connection.
6. If needed, choose from the **Debugger** pop-up menu a new debugger for use with the remote connection.
7. If needed, toggle the **Browse in processes window** option.
8. If needed, choose from the **Connection Type** pop-up menu a new network protocol for the remote connection.
9. If needed, enter a new Internet Protocol address for the remote computer.
10. Click **OK**.

The IDE updates the remote connection and displays changes in the **Remote Connections** list.

11. Click **OK**, **Apply**, or **Save**.

Removing Remote Connections

Remove a remote connection that the project no longer uses.

1. Choose **Edit > Preferences**.
2. Select the **Remote Connections** panel from the **IDE Preference Panels** list.
3. Select from the **Remote Connections** list the obsolete remote connection.
4. Click **Remove**.

The IDE updates the **Remote Connections** list.

5. Click **OK**, **Apply**, or **Save**.

Working with IDE Preferences

Debugger Panels

Working with Target Settings

This chapter explains core CodeWarrior™ IDE target settings panels and provides basic information on target settings options for the current project’s build targets. Consult the *Targeting* documentation for information on platform-specific target settings panels.

This chapter contains these sections:

- [Target Settings Window](#)
- [Target Panels](#)
- [Code Generation Panels](#)
- [Editor Panels](#)
- [Debugger Panels](#)

Abbreviated descriptions appear in this chapter. See [“Preference and Target Settings Options” on page 449](#) for more information on target settings panel options.

Target Settings Window

The **Target Settings** window lists settings for the current project’s build targets. These target settings supersede global preferences defined in the **IDE Preferences** window.

The Target Settings window lists settings by group:

- **Target**—configures overall build target settings, such as names, browser caching, file mappings, and access paths
- **Language Settings**—configures programming language settings. Consult the *Targeting* documentation for more information about these settings panels

Working with Target Settings

Target Settings Window

- **Code Generation (Windows)**—configures processor, disassembler, and optimization settings for generating code
- **Linker**—configure linker settings for transforming object code into a final executable file. Consult the *Targeting* documentation for more information about these settings panels.
- **Editor**—configure custom keyword sets and colors
- **Debugger**—configure settings for executable files, program suspension, and remote debugging
- **Command-Line Extras (Linux/Solaris)**—configure environmental variables for user applications and define custom tool commands (if necessary)

Figure 28.1 Target Settings window

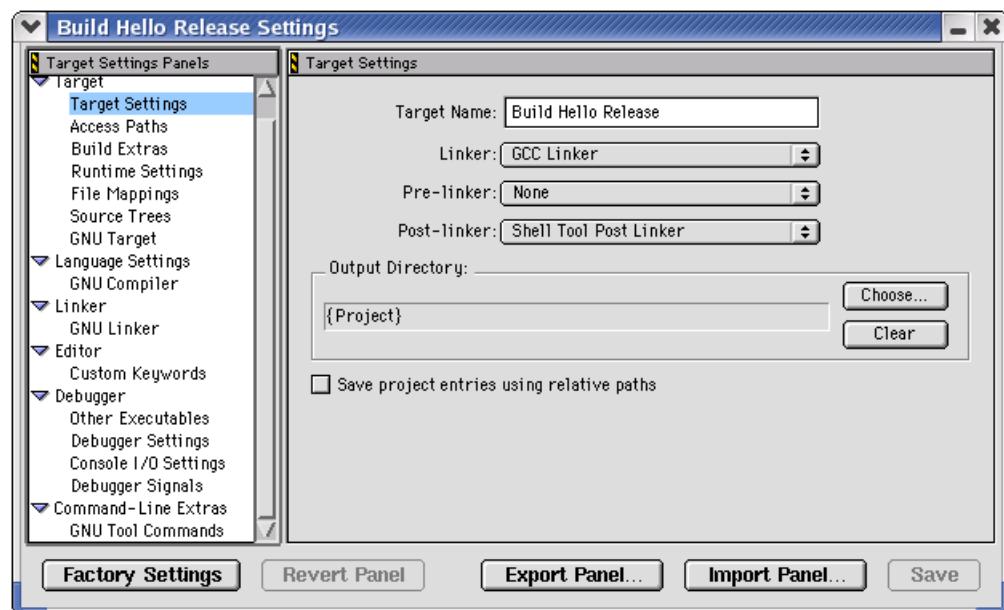


Table 28.1 Target Settings window

Item	Explanation
Target Settings Panels list	Lists settings panels, organized by group. Click the hierarchical control next to a group name to show or hide a list of individual settings panels.
Settings panel	Shows options for the selected item in the Target Settings Panels list.
Factory Settings	Click to restore the default options for the current settings panel.
Revert Panel	Click to restore the most recently saved options for the current settings panel.
Export Panel	Click to save an XML file that contains set options for the current panel.
Import Panel	Click to open an XML file that contains settings for the current panel.
OK (Windows)	Click to save modifications to all settings panels and close the window.
Cancel (Windows)	Click to discard modifications to all settings panels and close the window.
Apply (Windows)	Click to confirm modifications to all settings panels.
Save (Macintosh, Solaris, and Linux)	Click to save modifications to all settings panels.

Opening the Target Settings Window

Use the **Target Settings** window to modify build target options for the current project.

Choose **Edit > targetname Settings** to display the **Target Settings** window.

Working with Target Settings

Target Panels

Target Panels

The **Target** group of the Target Settings Panels defines general target settings assigned to a new project.

The panels available on most IDE hosts include:

- [Target Settings](#)
- [Access Paths](#)
- [Build Extras](#)
- [Runtime Settings](#)
- [File Mappings](#)
- [Source Trees](#)

Target Settings

The **Target Settings** panel provides options for:

- setting the name of the current build target
- setting the linker, pre-linker, and post-linker for the build target
- specifying the project output directory for the final output file

Figure 28.2 Target Settings panel

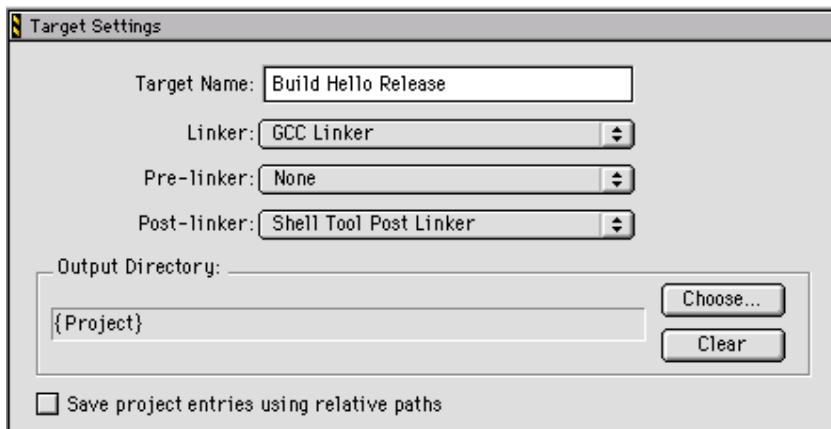


Table 28.2 Target Settings panel—items

Item	Explanation
Target Name	Enter a name (26 or fewer characters) for the selected build target as it will appear in the project window.
Linker	Select the linker to use on the current build target.
Pre-linker	Select the pre-linker to use on the current build target.
Post-linker	Select the post-linker to use on the current build target.
Output Directory	Shows the location where the IDE creates the output binary file. Click Choose to change this location.
Choose	Click to select the directory in which the IDE saves the output binary file.
Clear	Click to delete the current Output Directory path.
Save project entries using relative paths	Select to save project file entries using a relative path from a defined access path. This option is helpful if the project has multiple files with the same name.

Access Paths

The **Access Paths** settings panel defines the search paths for locating and accessing a build target's system files and header files.

NOTE The Windows version of the Access Paths settings panel displays either User Paths or System Paths, depending on the selected radio button. The Macintosh, Solaris, and Linux versions of the Access Paths settings panel display both User Paths and System Paths.

Working with Target Settings

Target Panels

Figure 28.3 Access Paths settings panel

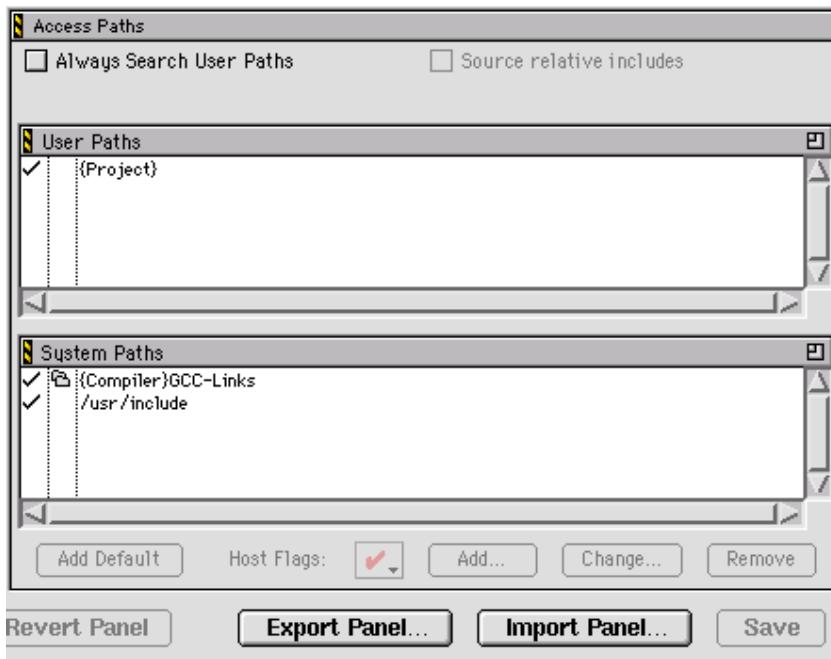


Table 28.3 Access Paths settings panel

Item	Explanation
Always Search User Paths	Select to treat <code>#include <...></code> statements the same as <code>#include "..."</code> statements.
Source relative includes	Select to search for dependent files in the same location as the source file. If the dependent file is not found in this location, specified User and System paths are searched. If this option is enabled, the Always Search User Paths should also be enabled.
User Paths	The User Paths list shows currently defined user-level access paths searched by <code>#include "..."</code> statements.

Table 28.3 Access Paths settings panel (*continued*)

Item	Explanation
System Paths	The System Paths list shows currently defined system-level access paths searched by <code>#include <...></code> statements.
Interpret DOS and Unix Paths (Macintosh)	Select to treat / and \ as subfolder separator characters. Clear to treat / and \ as ordinary text.
Require Framework Style Includes (Mac OS X)	Select to require <code>#include</code> statements of the form LibraryName/HeaderFile.h. Clear to allow statements of the form HeaderFile.h.
Add Default	Click to restore the default user- and system-level access paths.
Host Flags list pop-up	Choose the host platforms that can use the selected access path.
Add	Click to add a user- or system-level access path.
Change	Click to modify the selected user- or system-level access path.
Remove	Click to remove the selected user- or system-level access path.

The **User Paths** and **System Paths** lists display columns with status icons for each access path. There are different types of access paths. [Table 28.4](#) explains these items.

Table 28.4 User Paths and System Paths list columns

Name	Icon	Explanation
Search status	✓	A checkmark icon indicates an active access path that the IDE searches.
		No checkmark icon indicates an inactive access path that the IDE does not search.

Working with Target Settings

Target Panels

Table 28.4 User Paths and System Paths list columns (*continued*)

Name	Icon	Explanation
Recursive search		A folder icon indicates that the IDE recursively searches subdirectories of the access path.
		No folder icon indicates that the IDE does not recursively search the access path.
Framework (Mac OS X development)		An <i>f</i> icon indicates that the access path points to a framework. Framework paths are implicitly recursive.
		No <i>f</i> icon indicates that the access path does not point to a framework.
Access path		<p>Shows the full access path to the selected directory. Access paths have these types:</p> <ul style="list-style-type: none">• Absolute—the complete path, from the root level of the hard drive to the directory, including all intermediate directories• Project—the path from the project file relative to the designated directory• CodeWarrior—the path from the CodeWarrior IDE relative to the designated directory• System—the path from the operating system's base directory relative to the designated directory• Source tree—the path from a user-defined source tree relative to the designated directory

Build Extras

The **Build Extras** settings panel contains options that define how the CodeWarrior IDE builds a project.

Figure 28.4 Build Extras settings panel

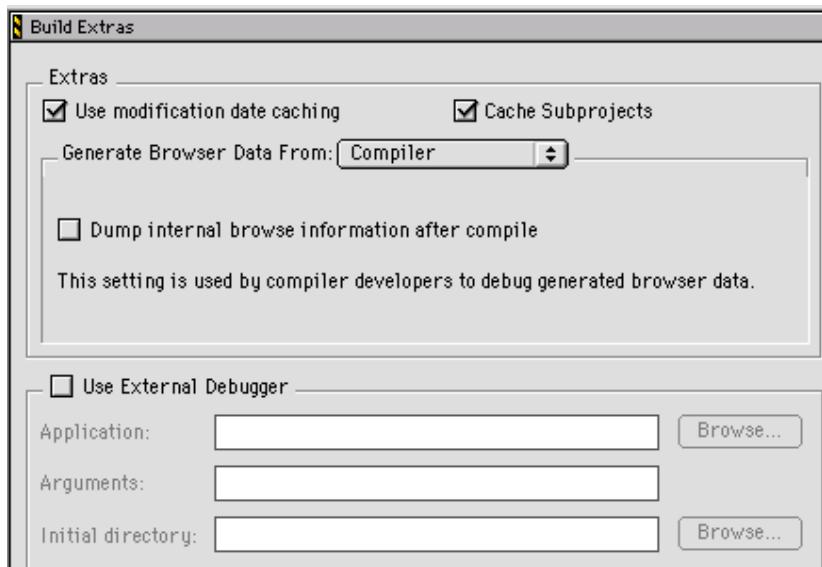


Table 28.5 Build Extras settings panel

Item	Explanation
Use modification date caching	Select to have the IDE cache modification date information and use that information each time it builds a target. Builds are faster if file modification dates are cached. Note that it is recommended to uncheck this option if you are using an external editor or using mounted directories. For one-time changes to files (for example, those updated by a VCS tool outside of the IDE or editing a file with an external editor), you should check the modification date by clicking the "Synchronize Modification Dates" button in the project window toolbar.
Cache Subprojects	Select to improve multi-project updating and linking speed.

Working with Target Settings

Target Panels

Table 28.5 Build Extras settings panel (*continued*)

Item	Explanation
Generate Browser Data From	Choose whether the IDE generates browser data for the project, and the method by which the IDE generates that data.
Dump internal browser information after compile	Select to have the IDE dump raw browser information for viewing. This option appears after selecting Compiler from the Generate Browser Data From pop-up menu.
Prefix file	Enter the path to your project's prefix file. This options appears after selecting Language Parser from the Generate Browser Data From pop-up menu.
Macro file	Enter the path to your project's macro file. This options appears after selecting Language Parser from the Generate Browser Data From pop-up menu.
Use External Debugger	Select to use an external debugger instead of the CodeWarrior debugger.
Application	Click Browse to select the external debugger application. Alternatively, enter the path to the external debugger.
Arguments	Enter any program arguments to pass to the external debugger when the IDE transfers control.
Initial directory	Click Browse to select an initial directory for the external debugger. Alternatively, enter the path to the initial directory.

Runtime Settings

The **Runtime Settings** panel specifies a debugging application for non-executable files. Dynamic linked libraries (DLLs), shared libraries, and code resources are sample non-executable files.

Figure 28.5 Runtime Settings panel

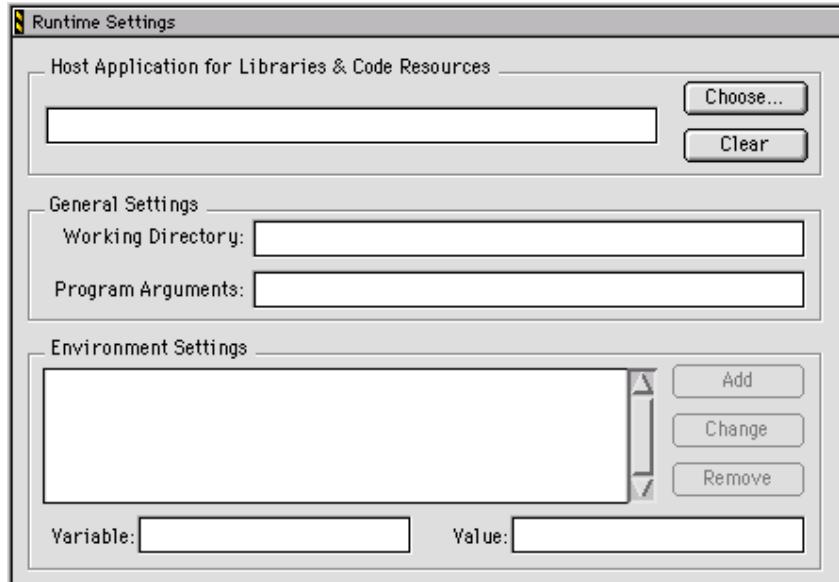


Table 28.6 Runtime Settings panel

Item	Explanation
<u>Host Application for Libraries & Code Resources</u>	Click Choose to select the program for debugging non-executable files. Alternatively, enter the path to the application program. Click Clear to delete the current field entry.
<u>Working Directory</u>	Enter the path to a directory used for debugging the non-executable files. Leave this field blank to use the same directory that contains the non-executable files.
<u>Program Arguments</u>	Enter a command line of program arguments to pass to the host application when the IDE transfers control.
<u>Environment Settings</u>	Lists the environment variables that have been added to the build target.
Add	Click to add the current Variable and Value pair to the Environment Settings list.

Working with Target Settings

Target Panels

Table 28.6 Runtime Settings panel (*continued*)

Item	Explanation
Change	Click to replace the selected entry in the Environment Settings list with the current Variable and Value pair.
Remove	Click to delete the selected environment variable from the Environment Settings list.
<u>Variable</u>	Enter a name for the environment variable. This name pairs with the information in the Value field.
<u>Value</u>	Enter a value for the environment variable. This value pairs with the information in the Variable field.

File Mappings

The **File Mappings** settings panel associates filename extensions with a CodeWarrior plug-in compiler. These associations determine whether the IDE recognizes a source file by its filename extension or file type. Use the settings panel to add, change, and remove file mappings.

Figure 28.6 File Mappings settings panel

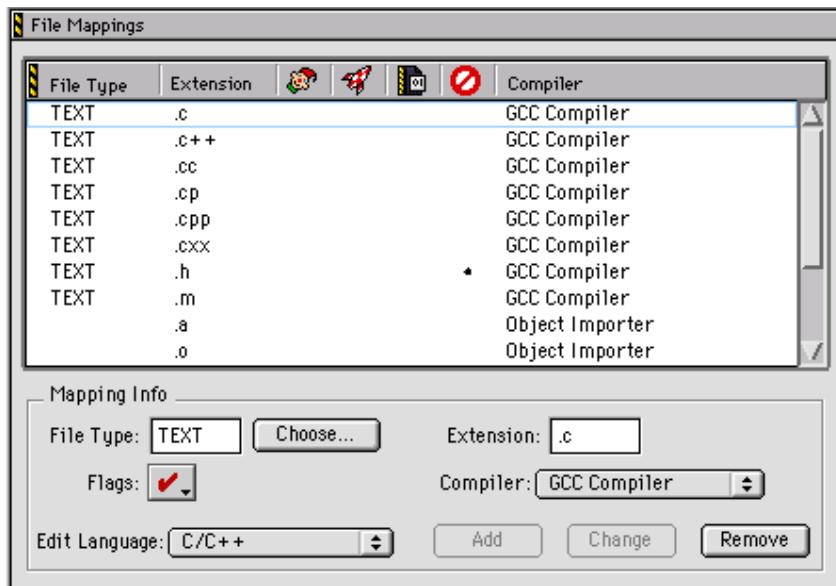


Table 28.7 File Mappings settings panel

Item	Icon	Explanation
File Mappings list		Displays a list of currently defined mappings between filename extensions and plug-in compilers.
File Type		Enter a file type (such as TEXT) for the file mapping. Alternatively, click Choose to set the file type by selecting an example file. This file type also appears in the corresponding column of the File Mappings list.
Extension		Enter the filename extension (such as .cpp) for the file mapping. This filename extension also appears in the corresponding column of the File Mappings list.
Resource File flag		A bullet in this column denotes a resource file. The IDE includes these resource files when building the final output file. Use the Flags pop-up menu to toggle this flag.

Working with Target Settings

Target Panels

Table 28.7 File Mappings settings panel (*continued*)

Item	Icon	Explanation
Launchable flag		A bullet in this column denotes a launchable file. The IDE opens launchable files with the application that created them. Double-click launchable files from the Project window. Use the Flags pop-up menu to toggle this flag.
Precompiled File flag		A bullet in this column denotes a precompiled file. The IDE builds precompiled files before building other files. Use the Flags pop-up menu to toggle this flag.
Ignored By Make flag		A bullet in this column denotes a file ignored by the compiler during builds. For example, use this option to ignore text (.txt) files or document (.doc) files. Use the Flags pop-up menu to toggle this flag.
Compiler		Choose from this list the plug-in compiler to associate with the selected file mapping. This compiler selection also appears in the corresponding column of the File Mappings list.
Flags		Choose from this pop-up menu the desired flags for the selected file mapping. A checkmark indicates an active flag. Bullets appear in the corresponding columns of the File Mappings list to reflect flag states.
Edit Language		Choose from this list the desired language to associate with the selected file mapping. The IDE applies the appropriate syntax coloring for the selected language.
Add		Click to add the current File Type, Extension, Flags, Compiler, and Edit Language entries to the File Mappings list.
Change		Click to change the selected item in the File Mappings list to reflect the current File Type, Extension, Flags, Compiler, and Edit Language entries.
Remove		Click to remove the selected item in the File Mappings list.

Source Trees

The **Source Trees** settings panel in the Target Settings window defines project-specific root paths. These project-specific paths override the global root paths defined in the **Source Trees** preference panel of the IDE Preferences window. Refer to “[Source Trees” on page 396](#) for information on adding, changing, or removing paths.

Code Generation Panels

The **Code Generation** group of the Target Settings Panels provides a single core panel for configuring optimization routines. Consult the *Targeting* documentation for more information about platform-specific settings panels.

Global Optimizations

The **Global Optimizations** settings panel configures how the compiler optimizes object code. All optimization routines rearrange object code without affecting its logical execution sequence.

NOTE Always debug programs with optimization routines disabled. The IDE does not provide source views of optimized code.

The Global Optimizations panel is specific to Metrowerks compilers. This panel is not used on the Linux-hosted IDE, since it uses gcc.

Working with Target Settings

Code Generation Panels

Figure 28.7 Global Optimizations settings panel

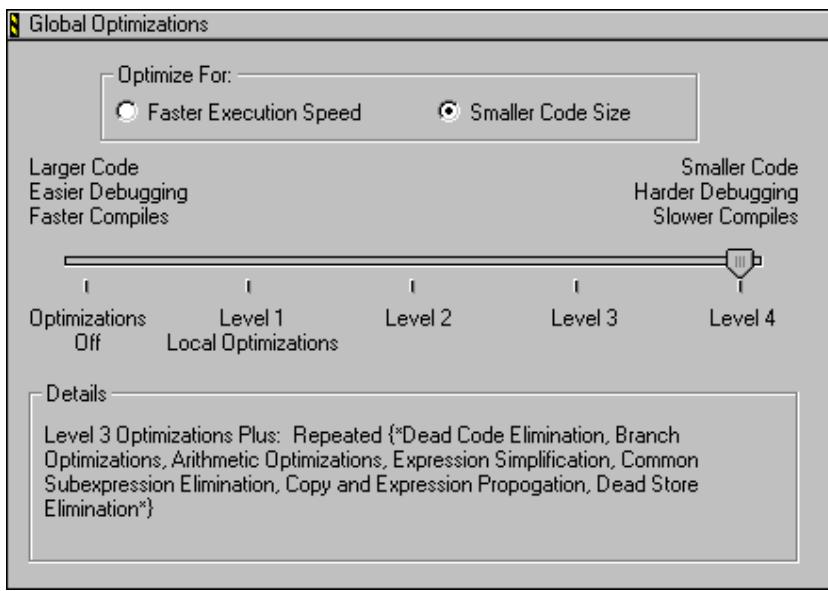


Table 28.8 Global Optimizations settings panel—items

Item	Explanation
Faster Execution Speed	Select to favor optimization routines that increase the execution speed of the final object code, at the expense of larger code size.
Smaller Code Size	Select to favor optimization routines that reduce the size of the final object code, at the expense of slower execution speed.
Optimization Level slider	Move to the desired optimization level. The IDE applies more optimization routines at higher optimization levels. The Details field lists the active optimization routines.

The **Details** field lists individual optimization routines applied at the selected optimization level. [Table 28.9 on page 439](#) explains these optimizations and their availability at certain optimization levels.

Table 28.9 Optimization routines

Optimization Routine	Explanation	Optimization Level
Global Register Allocation or Global Register Allocation Only for Temporary Values	Stores working values of heavily used variables in registers instead of memory.	1, 2, 3, 4
Dead Code Elimination	Removes statements never logically executed or referred to by other statements.	1, 2, 3, 4
Branch Optimizations	Merges and restructures portions of the intermediate code translation in order to reduce branch instructions.	1, 2, 3, 4
Arithmetic Operations	Replaces intensive computational instructions with faster equivalent instructions that produce the same result.	1, 2, 3, 4
Expression Simplification	Replaces complex arithmetic expressions with simplified equivalent expressions.	1, 2, 3, 4
Common Subexpression Elimination	Replaces redundant expressions with a single expression.	2, 3, 4
Copy Propagation or Copy and Expression Propagation	Replaces multiple occurrences of one variable with a single occurrence.	2, 3, 4
Peephole Optimization	Applies local optimization routines to small sections of code.	2, 3, 4
Dead Store Elimination	Removes assignments to a variable that goes unused before being reassigned again.	3, 4
Live Range Splitting	Reduces variable lifetimes to achieve optimal allocation. Shorter variable lifetimes reduce register spilling.	3, 4

Working with Target Settings

Editor Panels

Table 28.9 Optimization routines (*continued*)

Optimization Routine	Explanation	Optimization Level
Loop-Invariant Code Motion	Moves static computations outside of a loop	3, 4
Strength Reduction	Inside loops, replaces multiplication instructions with addition instructions.	3, 4
Loop Transformations	Reorganizes loop object code in order to reduce setup and completion-test overhead.	3, 4
Loop Unrolling or Loop Unrolling (Opt for Speed Only)	Duplicates code inside a loop in order to spread branch and completion-test overhead over more operations.	3, 4
Vectorization	For processors that support vector optimizations, translates computations with code-loop arrays into equivalent vector instructions.	3, 4
Lifetime Based Register Allocation or Register Coloring	In a particular routine, uses the same processor register to store different variables, as long as no statement uses those variables simultaneously.	3, 4
Instruction Scheduling	Rearranges the instruction sequence to reduce conflicts among registers and processor resources.	3, 4
Repeated	Iterates the optimization routines listed between { * and * }.	4

Editor Panels

The **Editor** group of the Target Settings Panels provides a single core panel for configuring custom keywords within a project.

Custom Keywords

The **Custom Keywords** panel configures as many as four keyword sets, each with a list of keywords and syntax coloring for a project. These project-specific settings supersede the global settings defined in the **Text Colors** preference panel of the IDE Preferences window.

Figure 28.8 Custom Keywords settings panel

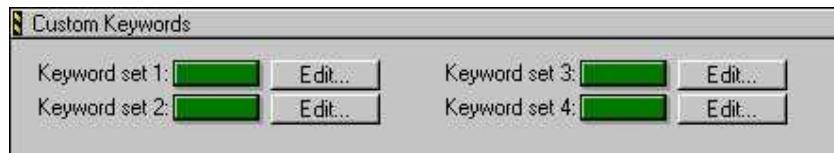


Table 28.10 Custom Keywords settings panel—items

Item	Explanation
Keyword set 1, Keyword set 2, Keyword set 3, Keyword set 4	Click a color swatch to set the color used for the corresponding custom-keyword set.
Edit	Click to add, modify, or remove keywords from the corresponding custom-keyword set.

Adding a Keyword to a Keyword Set

To add a keyword to a keyword set, follow these steps:

1. Click **Edit** next to the desired keyword set.

A dialog box appears. This dialog box lists the current collection of keywords in the keyword set.

2. Enter the new keyword into the field at the top of the dialog box.
3. Click **Add**.

The new keyword appears in the keyword list.

Working with Target Settings

Debugger Panels

4. Select **Case Sensitive** as desired.

When selected, the IDE treats the case of each keyword in the keyword set as significant. When cleared, the IDE ignores the case of each keyword in the keyword set.

5. Click **Done**.

The IDE saves the modified keyword set.

Removing a Keyword from a Keyword Set

To remove a keyword from a keyword set, follow these steps:

1. Click **Edit** next to the desired keyword set.

A dialog box appears. This dialog box lists the current collection of keywords in the keyword set.

2. Select the obsolete keyword in the Custom Keywords list.

3. Press the delete key for your platform.

- Windows, Solaris, and Linux: Backspace
- Macintosh: Delete

4. Click **Done**.

The IDE saves the modified keyword set.

Debugger Panels

The **Debugger** group of the Target Settings Panels defines general debugger settings for the project. Consult the *Targeting* documentation for more information about platform-specific settings panels.

The Debugger panels available on most IDE hosts include:

- [Other Executables](#)
- [Debugger Settings](#)
- [Remote Debugging](#)

Other Executables

The **Other Executables** settings panel configures additional executable files for the IDE to debug together with the current build target.

Figure 28.9 Other Executables settings panel

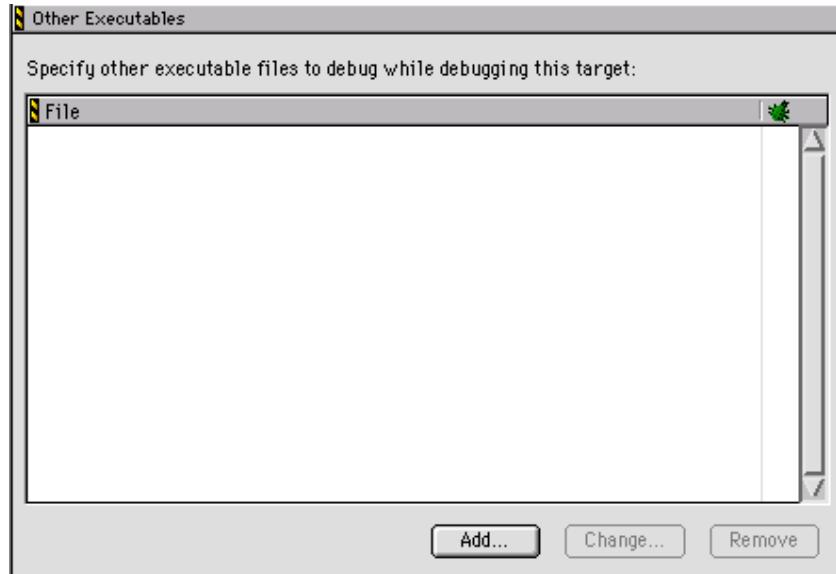


Table 28.11 Other Executables settings panel

Item	Icon	Explanation
File list		Lists executable files that the IDE can debug together with the current build target.
Debug column	debug icon	Click in this column to toggle debugging of the corresponding executable file.
Add		Click to select an executable file to add to the File list.
Change		Click to change the selected entry in the File list.
Remove		Click to remove the selected entry in the File list.

Working with Target Settings

Debugger Panels

Adding an Executable File to the File List

To add an executable file to the File list, follow these steps:

1. Click **Add**.

The **Debug Additional Executable** dialog box appears.

2. Enter in the **File Location** field the path to the executable file.

Alternatively, click **Choose** to display a dialog box. Use the dialog box to select the executable file. The path to the selected executable file appears in the **File Location** field.

3. Select **Download file during remote debugging** as desired.

When selected, the IDE downloads the executable file from a remote computer during the debugging session. Enter the path to the remote file.

Alternatively, click **Choose** to select the file. Click **Clear** to delete the current entry.

4. Select **Debug merged executable** as desired.

When selected, the IDE debugs an executable file that merged with the project output. Enter the path to the original executable file (prior to merging).

Alternatively, click **Choose** to select the file. Click **Clear** to delete the current entry.

5. Click **Done**.

The IDE adds the executable file to the File list.

Changing an Executable File in the File List

To change an executable file in the File list, follow these steps:

1. Select the desired path.

2. Click **Change**.

The **Debug Additional Executable** dialog box appears.

3. Modify the **File Location** field as desired.

4. Modify the **Download file during remote debugging** option as desired.

5. Modify the **Debug merged executable** option as desired.

6. Click **Done**.

The IDE modifies the executable file.

Removing an Executable File from the File List

To remove an executable file from the File list, follow these steps:

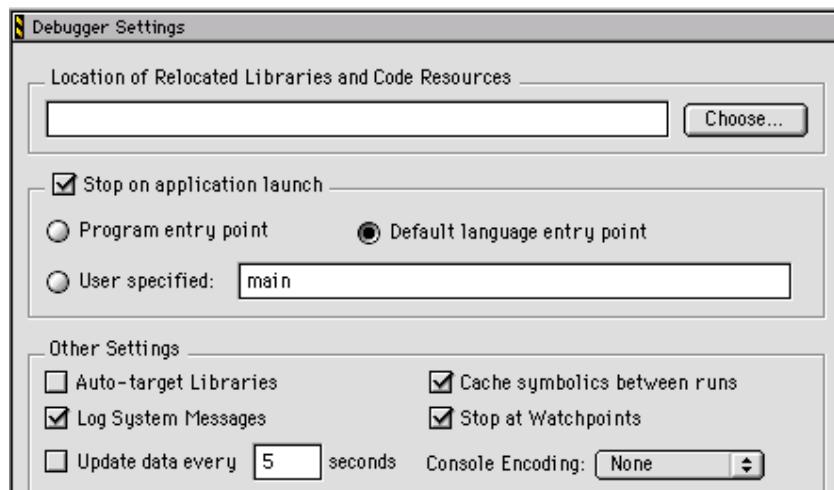
1. Select the obsolete path.
2. Click **Remove**.

The IDE removes the executable file from the File list.

Debugger Settings

The **Debugger Settings** panel configures activity logs, data-update intervals, and other debugger-related options.

Figure 28.10 Debugger Settings panel



Working with Target Settings

Debugger Panels

Table 28.12 Debugger Settings panel

Item	Explanation
Location of Relocated Libraries and Code Resources	Enter the path to code resources or relocated libraries required for debugging the project. Alternatively, click Choose to select the required files.
Stop on application launch	Select to halt program execution at the beginning of a debugging session. Select the desired stop point: Program entry point , Default language entry point , or User specified .
Program entry point	Select to halt program execution upon entering the program.
Default language entry point	Select to halt program execution upon entering a default point defined by the programming language.
User specified	Select to halt program execution at a specified function or address. Enter the desired function name or address in the corresponding field. If you enter an address, ensure that it is correct and within your program.
Auto-target Libraries	Select to debug dynamically linked libraries (DLLs) loaded by the target application, at the expense of slower performance.
Cache symbolics between runs	Select to have the IDE cache the symbolics information it generates for a project. Clear to have the IDE discard the information after each debugging session ends.
Log System Messages	Select to log all system messages to a Log window.
Stop at Watchpoints	Select to halt program execution at every watchpoint. Clear to halt program execution at watchpoints with changed values.
Update data every n seconds	Enter the number of seconds <i>n</i> to wait before updating the data displayed in debugging-session windows.

Remote Debugging

The **Remote Debugging** settings panel configures target-specific network settings for remote-debugging connections between the host computer and other computers. Use this target-specific panel to build on the general connections defined in the **Remote Connections** panel of the IDE Preferences window.

Figure 28.11 Remote Debugging settings panel

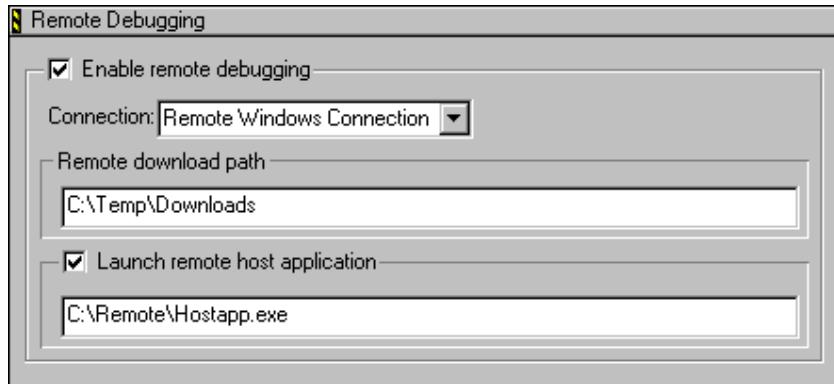


Table 28.13 Remote Debugging settings panel

Item	Explanation
Enable remote debugging	Select to define (for the current build target) a remote-debugging connection in terms of a general connection. Choose from the Connection pop-up menu the desired general connection.
Remote download path	Enter the path to the directory in which to store downloaded files.
Launch remote host application	Select to launch an application on the remote computer to serve as a host application. Enter the path to the remote application.

Working with Target Settings

Debugger Panels

Preference and Target Settings Options

Use this chapter to look up CodeWarrior™ IDE preference panel or target setting options and learn more about their capabilities. Option names are arranged in alphabetical order.

NOTE This chapter covers options for the core IDE preference or target setting panels described in this manual.

A

Activate Browser Coloring

Select this option to activate coloring of browser symbols in editor windows. Clear the option to apply the default text color to all symbols. Click the color swatch next to a symbol to modify its color.

Activate Syntax Coloring

Select this option to activate coloring of Comments, Keywords, Strings, and Custom Keyword Sets symbols in editor windows. Clear the option to apply the default text color to all symbols. Click the color swatch next to a symbol to modify its color.

Add Default

Click this button to restore the default user path or system path to the Access Paths panel.

Always Search User Paths

This option controls the search criteria the IDE uses to find system and user files.

- selected—the IDE treats searching for system files (such as `#include < . . . >`) the same as user files (`#include " . . . "`).
- disabled—the IDE treats system paths differently from user paths.

Application

In this field enter the path to the external debugger that the IDE uses in place of the CodeWarrior debugger. Alternatively, click **Browse** to open a dialog box. Use the dialog box to select the external debugger.

Arguments

In this field enter command-line arguments to pass to the external debugger at the beginning of a debugging session.

Attempt to show the dynamic runtime type of objects

Select this option to display runtime types for C++, Object Pascal, and SOM objects. Clear the option to display static types.

Auto Indent

Select this option to apply automatically the same indentation as the previous line for each new line of text created by pressing Enter or Return. Clear the option to always return to the left margin for each new line of text.

Auto Target Libraries

Select this option to have the IDE attempt to debug dynamically linked libraries (DLLs) loaded by the target application. The IDE debugs the DLLs that have symbolics information.

This option applies to non-project debugging sessions, such as debugging an attached process.

NOTE Selecting this option may slow IDE performance. Clear the option to improve speed.

Automatic Invocation

Select this option to have the Code Completion window automatically open after typing specific programming-language characters in the active editor window. Clear the option to manually open the Code Completion window.

The specific characters that trigger opening of the Code Completion window depend on the programming language that you use. For example, typing a period after a Java class opens the Code Completion window, allowing you to complete the class invocation.

You can change the time it takes for the Code Completion window to appear after you type a trigger character. If you perform any activity during this delay time, the Code Completion window is canceled.

See also:

- [“Code Completion Delay” on page 454](#)

Automatically launch applications when SYM file opens

Select this option to launch an application associated with an open symbolics file. The IDE sets an implicit breakpoint at the main entry point of the application. Clear the option to open the symbolics file without launching the associated application.

[Table 29.1](#) explains how to skip launching the target program

Table 29.1 Bypass launching the target program

On this host...	Do this...
Windows	Press Alt while the IDE opens the symbolics file.
Macintosh	Press Option while the IDE opens the symbolics file.
Solaris	Press Alt while the IDE opens the symbolics file.
Linux	Press Alt while the IDE opens the symbolics file.

B

Background

Click this color swatch to configure the background color of editor windows.

Balance Flash Delay

In this field enter the time, in ticks, to highlight a matching punctuation character during a **Balance while typing** check. Each *tick* represents 1/60th of a second (16.67 milliseconds).

Sample tick values include:

- 0 (zero)—disables balance flashing
- 30—the default flash value (1/2 of a second)
- 999—the maximum-flash delay value

Balance while typing

Select this option to have the editor check for balanced parentheses, brackets, and braces in editor windows. For each closing parenthesis, bracket, or brace, the editor attempts to find the opening counterpart.

The IDE behaves differently, depending on whether it finds the counterpart:

- Found—the editor window scrolls to display the matching character, then returns to the insertion point. The **Balance Flash Delay** option determines how long the editor displays the matching character.
- Not found—the IDE beeps.

Browser Commands

Select this option to add **Browser** menu commands to contextual menus. Clear the option to remove commands from the contextual menus.

Browser Path

In this field enter a path to the browser to use for viewing IDE online help. The Netscape Navigator® browser is the default application. The PATH environment variable specifies the path to the browser.

To change the default setting, or if the IDE cannot find Netscape Navigator, in the **Browser Path** field enter a path to an alternate browser. Alternatively, click **Set** to select the path.

Build before running

Choose from this pop-up menu the way in which the IDE handles project builds before running the compiled application:

- Always—always build projects before running them.
- Never—never build projects before running them.
- Ask—ask each time how to proceed.

C

Cache Edited Files Between Debug Sessions

Select this option to maintain a cache of edited files between debugging sessions. Use this option to debug through the original source code for files modified since the last build.

In the **Maintain files in cache** field enter the number of days to keep the cached files. Click **Purge Cache** to delete the current cache.

See also:

- [“Maintain files in cache” on page 470](#)
- [“Purge Cache” on page 473](#)

Cache Subprojects

Use this option to improve multi-project updating and linking.

Preference and Target Settings Options

- selected—the IDE increases its memory requirements in order to generate symbolics information for both the build targets and the subprojects within each build target.
- cleared—the IDE does not increase its memory requirements and does not generate symbolics information.

Cache symbolics between runs

Select this option to have the IDE maintain a cache of symbolics information generated for the project. The IDE refers to this cached symbolics information during subsequent debugging sessions. The cache improves IDE performance. Clear the option to force the IDE to discard the symbolics information at the end of each debugging session.

Case sensitive

Select this option to have the IDE consider case when completing code. Clear the option to have the IDE ignore case.

The IDE can determine possible symbol matches according to case. For example, if you clear the **Case sensitive** option and type `str` in the active editor window, the IDE displays both `string` and `String` as possible matches. Selecting the option causes the IDE to display only `string` as a possible match.

Close non-debugging windows

Select this option to close non-debugging windows, except for the active project window, when starting a debugging session. At the end of the debugging session, the IDE automatically re-opens the closed windows.

Code Completion Delay

In this field enter the number of ticks to have the IDE wait from the time you type a trigger character to the time the Code Completion window opens. A tick is 1/60 of a second.

Performing any activity during this delay time cancels opening of the Code Completion window.

See also:

- [“Automatic Invocation” on page 451](#)

Collapse non-debugging windows

Select this option to collapse non-debugging windows when starting a debugging session. At the end of the debugging session, the IDE automatically restores the collapsed windows.

Comments

Select the **Activate Syntax Coloring** option in order to configure this option. Use this option to configure the color of C, C++, and Java comments displayed in editor windows. The IDE then uses the chosen color for comments placed between /* and */ or from // to the end of a line.

Click the color swatch next to Comments to set the color.

Compiler

Choose from this list pop-up the desired compiler for the selected **File Type** in the **File Mappings** list. Select **None** to not associate the selected file type with any compiler.

Compiler thread stack

In this field enter the maximum kilobytes of stack size for the IDE to allocate to compiling and linking thread support.

The IDE threads all build processes, with compiling and linking occurring on a thread separate from the main application thread. This setting controls the compiler-thread stack size.

To avoid frequent compiler crashes, such as when building very large or complex projects, increase the default compiler-thread-stack size.

Confirm invalid file modification dates when debugging

Select this option to keep track of source-file modification dates in a project. The IDE displays a warning message if the modification dates do not match. The message warns of possible discrepancies between object code and source code. Clear the option to prevent the IDE from displaying the warning message.

Confirm “Kill Process” when closing or quitting

Select the **Confirm “Kill Process” when closing or quitting** option to have the IDE prompt for confirmation before killing processes upon closing the Thread window or quitting the IDE. Clear the option to kill processes without prompting.

Context popup delay

In this field enter the minimum time, in ticks, to hold down the mouse button before IDE contextual menus appear. Each *tick* represents 1/60 of a second (16.67 milliseconds).

Sample tick values include:

- 0 (zero)—disables appearance of contextual menus
- 40—default popup delay value (2/3 of a second)
- 240—maximum popup delay value

D

Debugger Commands

Select this option to add **Debug** menu commands to IDE contextual menus. Clear the option to remove commands from the contextual menus.

Default file format

Choose from this list pop-up the default end-of-line (EOL) conventions used by the IDE to save files:

- Macintosh: <CR>
- DOS: <LF><CR>
- UNIX: <LF>

Default language entry point

Select this option to halt program execution upon entering a default point defined by the programming language. For example, C++ defines the `main()` function as the default point.

Default size for unbounded arrays

Enter in this field the default number of elements to display in **View Array** windows for unbounded arrays.

Disable third party COM plugins

Select this option to prevent the IDE from loading third-party Component Object Model (COM) plugins. Clear the option to have the IDE load the plugins at start-up time.

Use this option to help troubleshoot problems with the IDE. If the problem goes away after disabling the plug-ins, then a conflict exists between the third-party plugins and the IDE plugins.

Display deprecated items

Select this option to have the Code Completion window display obsolete programming-language items. Clear the option to have the window hide the obsolete items.

Deprecated items appear in gray text in the Code Completion window.

Do nothing

Select this option to leave all windows in place during a debugging session.

Do nothing to project windows

Select this option to prevent the IDE from manipulating project windows when starting a debugging session. Use this option to help debug multiple build targets or multiple projects.

Documents

In this field enter the number of recent documents to display in the **Open Recent** submenu.

Preference and Target Settings Options

Don't step into runtime support code

Select this option to have the IDE bypass stepping into the Metrowerks Standard Library (MSL) runtime support code and instead directly step into your own code. Clear the option to have the IDE step into the MSL runtime setup code, then step into your own code.

Drag and drop editing

Select this option to allow dragging and dropping of text in editor windows. Clear the option to disable drag-and-drop text editing.

Dump internal browse information after compile

Select this option to view the raw browser information that a plug-in compiler or linker provides for the IDE. Use this option to help develop plug-ins for use with the IDE.

NOTE After enabling the **Dump internal browse information after compile** option, compile only single files or small files. Compiling an entire project can create huge internal browser information for the IDE to display.

E

Edit Commands

Select this option to add **Edit** menu commands to IDE contextual menus. Clear the option to remove the commands from the contextual menus.

Edit Language

Choose from this pop-up menu the programming language to associate with the selected file mapping. The selected language determines the syntax-color scheme. For example, choose **C/C++** to apply the appropriate syntax-color scheme for C or C++ programming-language components.

Enable automatic Toolbar help

Select this option to display Balloon Help after resting the cursor over a toolbar button. Clear the option to prevent Balloon Help from appearing.

Enable remote debugging

Select this option to define a remote-debugging connection specific to the current build target. Choose from the **Connection** pop-up menu the general connection to use as the basis for the target-specific connection.

Enable Virtual Space

Use this option to configure the editor for handling spaces in different ways.

- selected—the editor allows moving the text-insertion point past the end of a line of text, using either the arrow keys or the mouse. After moving to the desired position, begin entering text. The editor automatically inserts spaces between the former end of the line and the newly entered text.
- cleared—the editor requires manual insertion of spaces to move past the end of a line of text.

Environment Settings

Use this section to specify environment variables to pass to your program as part of the environment parameter in your program's `main()` function, or as part of environment calls. These environment variables are only available to the target program. When your program terminates, the settings are no longer available.

NOTE The **Environment Settings** section appears only when you develop code for a Windows build target. The section does not appear for any other build target.

Export Panel

Click this button to save to an Extensible Markup Language (XML) file the current state of the active preference or settings panel.

Preference and Target Settings Options

Extension

In this field enter a filename extension, such as the `.c` or `.h`, for a selected File Type in the File Mappings list. [Table 29.2 on page 460](#) lists default filename extensions.

Table 29.2 Default filename extensions

Type	Extension	Explanation
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
	.dbg	CodeWarrior Debug Preferences
	.exp	Exported Symbol File
	.iMAP	CodeWarrior Link Map
	.MAP	CodeWarrior Link Map
Assembly	.a	Assembly Source File (Windows and Macintosh)
	.asm	Assembly Source File
	.dump	CodeWarrior Disassembled File

Table 29.2 Default filename extensions (*continued*)

Type	Extension	Explanation
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
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
Java	.JMAP	Java Import Mapping Dump
	.jpo	Java Constructor File
	.mf	Java Manifest File

Preference and Target Settings Options

Table 29.2 Default filename extensions (*continued*)

Type	Extension	Explanation
Library	.a	(Static) Archive Library (Solaris and Linux)
	.lib	Library File
	.o	Object File (Windows and Macintosh)
	.o	Object (Relocatable) Library or Kernel Module (Solaris and Linux)
	.obj	Object File
	.pch	Precompiled Header Source File
	.pch++	Precompiled Header Source File
	.so	Shared Library (Linux)
Script	.sh	Shell Script (Linux)
	.psh	Precompile Shell Script (Linux)
	.pl	Perl Script (Linux)
Mac OS X	.dylib	Mach-O Dynamic Library
	.a	Mach-O Static Library
	.o	Mach-O Object File
	.plist	Property List

F

Factory Settings

Click this button to change all modified options to their default values in the current preference or settings panel.

Failure

Choose from this pop-up menu a sound to play after a **Bring Up To Date** or **Make** operation fails.

File Type

Enter in this field the four-character file type for the selected file mapping in the **File Mappings** list.

Find and compare operations



A bullet in the **Find and compare operations** column, whose label appears at left, indicates that the IDE ignores matching folders for find-and-compare operations. Such operations include dragging a folder into fields in the **Find** window, or comparing folder contents.

Find Reference using

Choose from the **Find Reference using** options, an online browser application to look up references and definitions.

For example, use this option to look up documentation for language keywords:

1. Select an online browser application, such as THINK Reference, with the **Find Reference using** option.
2. Select a language keyword, such as `boolean`, in the source code.
3. Choose the **Find Reference** menu command. The IDE looks up reference information for the `boolean` keyword in the THINK Reference documentation.

Although they are not included with the CodeWarrior product, the IDE supports these online browser formats:

- Apple Help Viewer (CW manuals)
- Apple Help Viewer (Mac OS X API Ref)
- PalmQuest Reference (Palm Pilot)
- QuickView—such as Macintosh Programmer's Toolbox Assistant (MPTA)
- THINK Reference

Font

Choose from the **Font** options the typeface to use for displaying text in editor windows. This setting behaves in two different ways, depending on the current IDE state:

- No editor windows open—the setting modifies the default font. All editor windows take on the default font.
- Editor windows open—the setting modifies the font displayed in the frontmost editor window only. Other editor windows remain unaffected. The default font remains unchanged.

Font preferences

Select the **Font preferences** option to remember font settings for each file in a project. Clear the option to use the default font settings every time the IDE opens each file. The **Font & Tabs** preference panel defines the default settings.

Foreground

Use the **Foreground** option to configure the color of any text not affected by the **Activate Syntax Coloring** or **Activate Browser Coloring** options.

Click the color swatch to change the current color.

Generate Browser Data From

Choose from this pop-up menu whether the IDE generates browser data, and from what source it generates that data.

Choose from these possibilities:

- **None**—Disable browser-data generation. Certain IDE features that use browser data will be unable to work with the project, but the project's size will be smaller.
- **Compiler**—Have the IDE use the compiler to generate browser data. If you choose this option, you must Make the project in order to generate the

browser data. The IDE uses the compiler assigned to the project to generate browser data during the build process.

- **Language Parser**—Have the IDE use the language parser to generate the browser data. Certain IDE features, such as C/C++ Code Completion, function more effectively if you choose this option. The IDE uses the language parser assigned to the project to generate browser data.

NOTE If you choose the **Language Parser** option, you can also have the IDE take into account your custom macro definitions. To do so, enter the path to your prefix file in the **Prefix file** field and the path to your macro file in the **Macro file** field.

Grid Size X

In the **Grid Size X** field enter the number of pixels to space between markings on the x-axis of the Layout Editor grid.

Grid Size Y

In the **Grid Size Y** field enter the number of pixels to space between markings on the y-axis of the Layout Editor grid.

Hide non-debugging windows

Select the **Hide non-debugging windows** option to hide, but not close, non-debugging windows when starting a debugging session.

To reveal the hidden windows, do one of these tasks:

- Use the **Window** menu, or
- Double-click the names of the hidden files in the Project window, or
- Perform lookups for symbols within the hidden windows.

At the end of the debugging session, the IDE automatically reveals the hidden windows.

Host Application for Libraries & Code Resources

The **Host Application for Libraries & Code Resources** 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 that you

Preference and Target Settings Options

specify in this field is not the debugger application, but rather the application with which the non-executable file interacts.

Host Flags

The **Host Flags** list pop-up defines the host platforms which can use the selected access path. The settings include:

- **None**—no host can use this access path.
- **All**—all hosts can use this access path.
- **Windows**—only use this path for Windows build targets.
- **Mac OS**—only use this path for Mac OS build targets.

NOTE Multiple hosts can be selected.

Import Panel

Click **Import Panel** to load the contents of a previously saved Extensible Markup Language (XML) file into the active preference or settings panel.

Include file cache

Use the **Include file cache** option to specify the upper limit of kilobytes of memory used by the IDE for caching `#include` files and precompiled headers. The larger the value entered, the more memory the IDE uses to accelerate builds.

Initial directory

In this field enter the initial directory for use with the external debugger. Alternatively, click **Browse** to open a dialog box. Use the dialog box to select the initial directory.

Insert Template Commands

Select the **Insert Template Commands** option to display the **Insert Template** submenu in contextual menus. The submenu displays source-defined function templates. Clear to remove the submenu from the contextual menus.

NOTE Select the **Browser Commands** option in order to select the **Insert Template Commands** option. Otherwise, the **Insert Template Commands** state has no effect.

Interpret DOS and Unix Paths

This option determines how the IDE treats filenames for interface files:

- Selected—the IDE treats the backslash (\) and the forward slash (/) characters as subfolder separator characters. In the example

```
#include "sys/socks.h"
```

the IDE searches for a subfolder called sys that contains a socks.h file.
- Cleared—the IDE treats both the backslash and forward slash characters as part of the filename. Using the same example, the IDE now searches for a sys/socks.h filename.

K-L

Keywords

Use the **Keywords** option to configure the color of C, C++, and Java programming language's keywords displayed in editor windows when the **Activate Syntax Coloring** option is enabled. Coloring does not include macros, types, variables defined by system interface files, or variables defined in source code. Click the color swatch next to Keywords to set the color.

Launch Editor

Enter in the **Launch Editor** field a command-line expression that specifies the third-party text editor that the CodeWarrior IDE runs to edit text files.

The IDE expands the %file variable of the command-line expression into the full file path. For example, to run the Emacs text editor to edit text files, enter this command-line expression:

```
runemacs %file
```

Preference and Target Settings Options

Consult the documentation provided with the third-party text editor for more information about using command lines.

Launch Editor w/ Line

Enter in the **Launch Editor w/ Line #** field a command-line expression that specifies the third-party text editor that the IDE runs to edit text files, and an initial line of text that the third-party editor displays upon running.

The IDE expands the `%line` variable of the command-line expression into an initial line of text for the third-party text editor to display. For example, to run the Emacs text editor to edit a text file, and to have the Emacs editor display the line provided to it by the IDE, enter this command-line expression:

```
emacs %file %line
```

Consult the documentation provided with the third-party text editor for more information about using command lines.

Launch remote host application

Select this option to launch an application on the remote computer to serve as a host application. Enter the path to the remote host application.

Left margin click selects line



Select the **Left margin click selects line** option to use a right-pointing cursor, shown at left, to select entire lines of text from the left margin. Clear the option to disable use of the right-pointing cursor.

With the right-pointing cursor active, click in the left margin to select the current line, or click and drag along the left margin to select multiple lines.

Level

Choose from the **Level** options the amount of information reported for IDE plug-ins in development. This information is useful for diagnosing plug-in behavior or for viewing information about the properties of installed plug-ins.

Choose one of these levels of plug-in diagnostic information:

- **None** (default)—The IDE does not activate plug-in diagnostics or produce output.

- **Errors Only**—The IDE reports problems encountered while loading plug-ins. These problems appear in a new text file after the IDE starts up
- **All Info**—The IDE reports information for each installed plug-in, such as problems with plug-in loading, optional plug-in information, and plug-in properties. This information appears in a new text file after the IDE starts up. The text file also contains a complete list of installed plug-ins and their associated preference panels, compilers, and linkers.

The IDE allows saving and printing the text file. Use the file as an error reference for troubleshooting plug-ins. The text file also provides suggestions for correcting general plug-in errors.

Linker

Use the **Linker** option menu to select the linker to use with the project. The choices available are always dependent on the plug-in linkers that are available to the CodeWarrior IDE.

To learn more about the linkers, see the appropriate *Targeting* manual.

Location of Relocated Libraries and Code Resources

Enter in this field the path to the relocated libraries and code-resource files required for debugging the project. Alternatively, click **Choose** to display a dialog box. Use the dialog box to select the required files.

Log System Messages

Select this option to have the IDE maintain a log of all system messages generated during the debugging session. The Log window displays this platform-specific information. Clear the option to disable the log.

M

Maintain files in cache

Enter in the **Maintain files in cache** text box the number of days that the IDE maintains files in the file cache.

Menu bar layout

Choose from the **Menu bar layout** options the desired configuration of menus listed in the IDE:

- **Windows**—organizes the menu bar according to a typical Microsoft® Windows® arrangement
- **Macintosh**—organizes the menu bar according to a typical Apple® Mac® OS arrangement

Minimize non-debugging windows

Select the **Minimize non-debugging windows** option to minimize non-debugging windows to a reduced size when a debugging session starts. At the end of the debugging session, the IDE automatically restores the minimized windows.

NOTE The **Minimize non-debugging windows** option is only available in MDI mode.

See also:

- [“Use Multiple Document Interface” on page 487](#)

Monitor for debugging

Choose from the **Monitor for debugging** options the specific monitor to use during debugging sessions. The IDE displays debugging windows in the selected monitor. The coordinates in parentheses identify the selected monitor in QuickDraw space.

Move open windows to debugging monitor when debugging starts

Select the **Move open windows to debugging monitor when debugging starts** option to move all open windows to the selected debugging monitor after a debugging session starts. At the end of the debugging session, the IDE restores the moved windows to their original positions.

O

Open windows on debugging monitor during debugging

Select the **Open windows on debugging monitor during debugging** option to display on the debugging monitor any window that opens during the debugging session.

The IDE does not save the positions of windows closed on the debugging monitor during the debugging session. This behavior prevents window positions from gravitating to the debugging monitor.

Output Directory

Use the **Output Directory** caption to show the location the IDE places a final linked output file. The default location is the directory that contains your project file. Select **Choose** to specify the location path.

P

Play sound after ‘Bring Up To Date’ & ‘Make’

Select the **Play sound after ‘Bring Up To Date’ & ‘Make’** option to play a sound after a build operation completes. Choose different sounds for successful

Preference and Target Settings Options

and unsuccessful builds using the **Success** and **Failure** pop-up options, respectively.

See also:

- [“Failure” on page 463](#)
- [“Success” on page 483](#)

Post-linker

Use the **Post-linker** option to select a post-linker that performs additional work (such as format conversion) on the final executable file.

For more information see the appropriate *Targeting* manual.

Pre-linker

Use the **Pre-linker** option to select a pre-linker that performs additional work on the object code in a project. This work takes place before the IDE links the object code into the final executable file.

For more information about the pre-linkers available, see the build targets *Targeting* manual.

Program Arguments

Use the **Program Arguments** field to enter command-line arguments to pass to the project at the beginning of a debugging session. Your program receives these arguments after you choose **Project > Run**.

Program entry point

Select this option to halt program execution upon entering the program.

Projects

Enter the number of recent projects to display in the **Open Recent** submenu.

Project Commands

Select the **Project Commands** option to add **Project** menu commands to contextual menus. Clear the option to remove the commands from the contextual menus.

Project operations



A bullet in the **Project operations** column, whose label appears at left, indicates that the IDE ignores matching folders for project operations. Such operations include dragging a folder into the Project window, building a project, or searching access paths after choosing **File > Open**.

Purge Cache

Click **Purge Cache** to delete the contents of the current file cache.

R

Recommended

Select the **Recommended** option to allow the number of concurrent compiles suggested by the IDE. This suggestion takes into account the number of active Central Processing Units (CPUs) on the host computer.

Regular Expression

Enter in the **Regular Expression** field a text pattern to match against folder names. The IDE excludes matching folders and their contents from selected project operations or find-and-compare operations.

Relaxed C popup parsing

Use the **Relaxed C popup parsing** option to control the strictness of C coding conventions:

- Select the option to have the IDE recognize some non-standard functions that interfere with Kernighan-and-Ritchie conventions. The IDE displays the non-standard functions in the **Routine** list pop-up.
- Clear the option to have the IDE recognize only functions that conform to Kernighan-and-Ritchie conventions. The IDE displays only the standard functions in the **Routine** list pop-up.

Preference and Target Settings Options

For more information, refer to “Reference Manual,” of *The C Programming Language, Second Edition*, by Kernighan and Ritchie, published by Prentice Hall.

NOTE Toggle the **Relaxed C popup parsing** option to maximize recognition of functions, macros, and routine names in the source code.

Remote download path

Enter the path to the directory in which to store files downloaded from the remote host application.

Require Framework Style Includes

This option determines the strictness with which the IDE treats `#include` statements for frameworks:

- selected—the IDE requires the framework in addition to the referenced header file. In the example

```
#include <Cocoa/CocoaHeaders.h>
```

the IDE requires the presence of `Cocoa/` in order to find the `CocoaHeaders.h` file.
- cleared—the IDE requires only the referenced header file. Using the same example, `Cocoa/` becomes optional.

Revert Panel

Click **Revert Panel** to revert all modified options in the current preference or settings panel to the values present when the panel was originally opened.

S

Save open files before build

Select the **Save open files before build** option to automatically save files during project operations:

- Preprocess
- Precompile
- Compile
- Disassemble
- Bring Up To Date
- Make
- Run

Save project entries using relative paths

Use the **Save project entries using relative paths** option to store the location of a file using a relative path from one of the access paths. The settings include:

- **enabled**—the IDE stores extra location information to distinctly identify different source files with the same name. The IDE remembers the location information even if it needs to re-search for files in the access paths.
- **disabled**—the IDE remembers project entries only by name. This setting can cause unexpected results if two or more files share the same name. In this case, re-searching for files could cause the IDE to find the project entry in a different access path.

Script

Choose from the **Scripts** options the script system (language) used to display text in editor windows. This setting behaves in two different ways, depending on the current IDE state:

- No editor windows open—the setting modifies the default script system. All editor windows take on the default script system.
- Editor windows open—the setting modifies the script system displayed in the frontmost editor window only. Other editor windows remain unaffected. The default script system remains unchanged.

Select stack crawl window when task is stopped

Select the **Select stack crawl window when task is stopped** option to automatically bring the Thread window to the foreground after the debugger stops a task. Clear the option to leave the Thread window in its previous position.

Preference and Target Settings Options

This option is useful for watching variable values change in multiple Variable windows as the debugger steps through code.

Selection position

Select the **Selection position** option to remember these items for each editor window:

- visible text
- insertion-point location
- selected text

Clear the option to open each editor window according to default settings and place the insertion point at the first line of text.

NOTE The IDE must be able to write to the file in order to remember selection position.

Show all locals

Select the **Show all locals** option to display all local variables in Variable windows. Clear the option to show only variables near the program counter.

The Variables pane uses these display settings:

- **Variables: All**—shows all local variables in the code.
- **Variables: Auto**—only shows the local variables of the routine to which the current-statement arrow currently points.
- **Variables: None**—does not show variables. Use this setting to improve stepping performance for slow remote connections.

Show Code and Data Sizes

Enable this option in the IDE Extras panel of the IDE preferences panels to display the Code and Data columns in the project manager window.

Show hidden locals

In previous versions of the CodeWarrior debugger, all local variables were displayed at all times in the local variables view. This meant that if there were multiple sub-scopes within a function that contained variables of the same name,

all copies of these variables would be shown at all times. This made it difficult at times to determine which copy of a variable was the "current" one.

The CodeWarrior debugger now optionally filters out out-of-scope local variables in the local variables view. It is now possible to make the CodeWarrior debugger only display those variables that are actually "live" for the current location.

A new filter option (live) has been added to the existing options (all, auto, and none). The new option will attempt to filter out variables that are not currently in scope.

A hidden local variable is a variable that is in scope, but is hidden by a variable of the same name in a deeper scope. If the Show hidden locals option is checked, hidden locals are shown dimmed (greyed out).

Consider the following function:

```
int main( void )
{
    char varA = 'a';
    int varB = 111;

    for (int loop1 = 0; loop1 < 10; loop1++)
    {
        /* loop-scoped variables that should hide all others */
        char varA = 'b';
        float varB = 2.22;
        .
        .
    }
    .
}
```

Previously, when debugging this function, the CodeWarrior debugger would show five variables at all times, like this:

```
loop1 : int
varA : char
varA : char
varB : int
```

Preference and Target Settings Options

```
varB : float
```

This could be confusing, since the different varA's were not apparent in the local variables pane. With the new "live" filter, only those variables that are actually active at a given point in the function are displayed. Using the "live" filter, this display will look like this at the beginning of the function:

```
varA : char
```

```
varB : int
```

and the display will look like this when inside the for loop:

```
loop1 : int
```

```
varA : char
```

```
varB : float
```

Note in the second case that the "varA" and "varB" variables declared at the beginning of the function are still "live", but they are hidden by the same-named variables declared within the for loop. If you would like these hidden variables to be displayed, the "Show hidden locals" option should be checked. If that option is selected, the variable list when inside the for loop will look like this:

```
loop1 : int
```

```
varA : char // displayed as dimmed
```

```
varA : char
```

```
varB : int // displayed as dimmed
```

```
varB : float
```

NOTE This feature ONLY works correctly if there is compiler and symbolics plugin support for sub-scopes within functions. At present, there are very few compilers that actually generate sub function-level scope information, so it is entirely possible that you will see no difference between the "live" and "all" settings -- you will continue to have the "classic" CodeWarrior variable display with all variables shown.

Show message after building up-to-date project

Select the **Show message after building up-to-date project** option to have the IDE display a message after building an up-to-date project.

Show threads in separate windows

Select the **Show threads in separate windows** option to open a separate Thread window for each task. Clear the option to use one Thread window to display multiple tasks.

Show processes in separate windows

Select the **Show processes in separate windows** option to open a separate window for each process. Clear the option to use one window to display multiple tasks processes.

Show the component palette when opening a form

Select the **Show the component palette when opening a form** option to automatically display the Component Palette after opening a form in the Layout Editor. Clear the option to require manual opening of the Component Palette.

Show the object inspector when opening a form

Select the **Show the object inspector when opening a form** option to automatically open an Object Inspector window when opening a layout in the Layout Editor. Clear the option to require manual opening of the Object Inspector.

Show values as decimal

Select the **Show values as decimal instead of hex** option to display variable values in decimal form. Clear the option to display the values in hexadecimal form.

Show variable location

Select the **Show variable location** option to display the **Location** column in the Variables pane of the Thread window. Clear the option to hide the **Location** column.

Show variable types

Select the **Show variable types** option to display the type associated with each variable in Variable windows. Clear the option to hide the variable types.

Show variable values in source code

Select the **Show variable values in source code** option to show current values for variable names displayed in contextual menus. Clear the option to show variable names only.

Size

Choose from the **Size** options the font size used to display text in editor windows. This setting behaves in two different ways, depending on the current IDE state:

- No editor windows open—the setting modifies the default font size. All editor windows take on the default font size.
- Editor windows open—the setting modifies the font size displayed in the frontmost editor window only. Other editor windows remain unaffected. The default font size remains unchanged.

Sort functions by method name in symbolics window

Select the **Sort functions by method name in symbolics window** option to alphabetically sort functions by method name. Clear the option to alphabetically sort by class name. The sorting affects functions of the form `className::methodName` that appear in the Symbolics window.

Since most C++ and Java source files contain methods that belong to the same class, select the option to simplify selection of functions by typing method names.

Stop at Watchpoints

Select this option to halt program execution at each watchpoint, regardless of whether the watchpoint value changed. Clear the option to halt execution at watchpoints with changed values.

Stop on application launch

Select this option to halt program execution at a specified point each time a debugging session begins.

Strings

Use the **Strings** option to configure the color of anything that is not a comment, keyword, or custom keyword and displayed in editor windows when the **Activate Syntax Coloring** option is enabled. Sample strings include literal values, variable names, routine names, and type names.

Click the color swatch next to Strings to set the color.

Smart Variable Formatting

The Variable Formatter is an IDE plugin that customizes the display of Variables based on format data it reads from an XML format file. For specific types of variables, the Variable Formatter will replace the text shown next to the variable name to the results of an expression. For example, if you have a struct:

```
struct Rect {  
    short          top;  
    short          left;  
    short          bottom;  
    short          right;  
};
```

then normally a variable of that type would look like this in the debugger:

```
myRect 0x000DCEA8
```

If the Variable Formatter is given a format that looks like this:

Preference and Target Settings Options

```
<variableformat>
<typename>Rect</typename>
<expression>
" {T: " + ^var.top +
" L: " + ^var.left +
" B: " + ^var.bottom +
" R: " + ^var.right +
"}{H: " + (^var.bottom - ^var.top) +
" W: " + (^var.right - ^var.left) + "}"
</expression>
</variableformat>
```

then the variable will be displayed with the result of the expression:

```
myRect {T: 30 L: 30 B: 120 R: 120}{H: 90 W: 90}
```

When the IDE starts, the variable formatter plugin looks in the plugin's support folder for a "VariableFormats" folder. It scans this folder for XML files and reads the variable formats for each one.

Variable Format Tags:

variableformat - Identifies the start of a variable format record.

osName - Restricts format use to the indicated operating system. OS names are "osMac" and "osWin32".

runtimename - Restricts format use to the indicated runtime model. Runtime names are "runtimePPCCFM", "runtimePPCMacho" and "runtimeWin32".

cpuname - Restricts format use to the indicated CPU model. CPU names are "cpuPowerPCBig", "cpuJava" and "cpux86".

typename - Identifies the name of the Type this record will format.

condition - Specifies a condition that must be met for the format to be used. This can be used to test for one element of data before attempting to format another element.

typenamematch - Specifys how to match type names to variable types. Possible values are: "ExactMatch", "BeginsWith", "EndsWith", and "Contains".

expression - Specifies an expression string. The expression will be evaluated and the result displayed next to the variable. Before evaluation, all instances of "^var" in the format string will be replaced with the name of the variable.

expressionformat - Specifies the data format to use when formatting an expression. The format names match the menu item names in the "Data" menu: "Pascal String", "C String", "Character", "Unicode" etc.

Sort function popup

Select the **Sort function popup** option to sort function names by alphabetical order in list pop-ups. Clear the option to sort function names by order of appearance in the source file.

Source relative includes

Select to search for dependent files in the same location as the source file. If the dependent file is not found in this location, specified User and System paths are searched. If this option is enabled, the Always Search User Paths should also be enabled. For example, if the compiler is currently scanning the main source file and discovers an include header file statement, the header file is searched for in the same location as the main file. If not found, the specified access paths will be searched. If the header file declared in the main file also contains an include statement for another header file, it too will be searched for in the same sequence.

Success

Choose from the **Success** options a sound to play after a **Bring Up To Date** or **Make** operation succeeds.

Symbolics

Enter the number of recent symbolics files to display in the **Open Recent** submenu.

System Paths

Click the **System Paths** radio button to display the System Paths pane in the Access Paths preference panel.

Supported hosts:

- Windows: available.
- Macintosh: not available.

T

Tab indents selection

Use the **Tab indents selection** option to control how the editor inserts tabs into the currently selected lines of text:

- Select the option so that pressing Tab causes the editor to insert tab characters in front of each selected line of text. The editor thereby indents the selected text.
- Clear the option so that pressing Tab causes the editor to replace selected text with a tab character. The editor thereby overwrites the selected text.

Tab Inserts Spaces

Select the **Tab Inserts Spaces** option to have the editor insert spaces instead of tab characters into text. Clear the option to have the editor use tab characters.

The **Tab Size** option determines the number of spaces inserted by the editor.

Tab Size

Enter in the **Tab Size** field the number of spaces to substitute in place of a tab character in text. This number applies to the **Tab Inserts Spaces** option.

Target Name

Use the **Target Name** text box to set or modify the name of the current build target. This name appears in the Targets view in the Project window. This name is not the name assigned to the final output file, that is set in the Linker panel for the build target.

Type

Choose from the **Type** options the desired source-tree path type:

- **Absolute Path**—This source-tree type is based on a file path.
- **Environment Variable**—This source-tree type is based on an existing environment-variable definition. The Macintosh-hosted IDE cannot create or modify this source-tree type.

- **Registry Key**—This source-tree type is based on an existing Windows registry key entry.
-

U

Update data every *n* seconds

Select this option to update the information displayed in debugging-session windows after a specified time interval. Enter the number of seconds *n* to elapse before the next update. Clear this option to prevent data updates and keep the same window information throughout the debugging session.

Use Concurrent Compiles

Select the **Use Concurrent Compiles** option to run more than one compiling process at a time. Concurrent compiling makes better use of available processor capacity by allowing the operating system to optimize resource utilization, such as taking advantage of over-lapped input/output.

Both single- and multi-processor systems benefit from enabling concurrent compiles. On multiprocessor systems, the speed-up is significant.

Use Debugging Monitor

Select the **Use Debugging Monitor** option to view debugging windows on a second monitor after a debugging session starts. This option only appears when the second monitor is connected to the computer.

Use default workspace

Select this option to have the IDE use the default workspace. The IDE uses the default workspace to save and restore window and debugging states from one session to the next.

For example, if you select this option and close the IDE with a project window visible onscreen, that project window reappears the next time you start the IDE.

Clear this option to have the IDE start with the same default state for each new session: no windows visible onscreen.

Preference and Target Settings Options

For example, if you clear this option and close the IDE with a project window visible onscreen, that project window does not appear the next time you start the IDE. Instead, the IDE always starts without opening any windows.

Use External Debugger

Select this option to have the IDE use an external debugger application in place of the CodeWarrior debugger.

Use External Editor

Select the **Use External Editor** option to use an external text editor to modify text files in the current project. Clear the option to use the text editor included with the IDE.

Use Local Project Data Storage

Select the **Use Local Project Data Storage** option to store (on the host computer) data associated with a project file on a read-only volume. Clear the option to store project data inside the same folder as the project file itself.

After loading a project file, the IDE creates or updates an associated project data folder. The IDE stores intermediate project data in this folder. When building or closing a project, the IDE uses the information in the project data folder to update the project file.

By default, the IDE places the project data folder within the same folder as the project file. However, the IDE cannot create or update a project data folder in a location that grants read-only privileges.

If you are creating one project to be accessed by multiple users that are running CodeWarrior on separate machines, then each user should select this option to create a local data storage folder for the shared project. The folder containing the project file should be set to read-only. This will cause the target information to be stored locally on each user's machine, instead of inside a folder next to the project file.

Use modification date caching

Use the **Use modification date caching** option to determine whether the IDE checks the modification date of each project file prior to making the project. The settings include:

- **enabled**—the IDE caches 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. This can shorten compilation time significantly for large projects.
- **disabled**—the IDE checks every file at each recompile of the project. Use this setting if using third-party editors to ensure that the IDE checks every file at compilation time.

Use Multiple Document Interface

Toggle this option to change the IDE interface:

- Selected—The IDE uses **MDI** (Multiple Document Interface). In this interface, the IDE uses a main application window with a gray background. IDE windows appear inside the main application window. The gray background obscures your view of the desktop.
- Cleared—The IDE uses **FDI** (Floating Document Interface). In this interface, the IDE does not use a main application window. You can see through the IDE user interface to your desktop. IDE windows appear above the desktop.

Use multiple undo

Select the **Use multiple undo** option to remember several undo and redo operations in editor windows. Clear the option to remember only the most recent undo or redo action.

The IDE stores undo and redo actions on a stack in first-in last-out (FILO) order, however, the stack size and capability are limited. For example, assume there are five undo actions on the stack (ABCDE). If the IDE redoes two actions (ABC), then performs a new action (ABCF), the undo events (DE) are no longer available.

Use Script menu

Select the **Use Script menu** option to display the **Scripts** menu in the IDE menu bar. Clear the option to remove the Scripts menu from the menu bar. The Scripts menu provides convenient access to IDE scripts.

For more information about scripting the IDE, refer to the *CodeWarrior Scripting Reference*.

Use Third Party Editor

Select the **Use Third Party Editor** option to use a third-party text editor to modify text files. Clear the option to use the text editor included with the IDE.

Enter in the **Launch Editor** and **Launch Editor w/ Line #** fields command-line expressions that specify information that the IDE passes to the third-party editor.

Consult the documentation provided with the third-party text editor for more information about using command lines.

See also:

- [“Launch Editor” on page 467](#)
- [“Launch Editor w/ Line #” on page 468](#)

Use ToolServer menu

Select the **Use ToolServer menu** option to display the **ToolServer** menu in the IDE menu bar. Clear the option to remove the ToolServer menu from the menu bar.

User Paths

Click this radio button to display the **User Paths** pane in the **Access Paths** preference panel.

User Specified

Select the **User Specified** option to stipulate the number of concurrent compiles to allow in the IDE. Enter the desired number in the text box beside the option.

NOTE The IDE accommodates a maximum of 1024 concurrent compiles. However, there is a point where the host system becomes compute-bound, and allowing more processes only adds overhead. For a single-processor system, the practical limit is approximately 12 concurrent compiles.

V

Value

The **Value** text box defines the value of the variable defined in the **Variable** text box that will be passed to a host application when control is transferred to it by the IDE.

Variable

The **Variable** text box defines the name of a variable to be passed to a host application when control is transferred to it by the IDE.

Variable values change

Use the **Variable values change** option to configure the color of changed variables that appear in debugger windows. Click the color swatch to change the current color.

VCS Commands

Select the **VCS Commands** option to add **VCS** menu commands to contextual menus. Clear the option to remove the commands from the contextual menus.

Refer to the documentation that came with the version control system to learn about using it with the CodeWarrior IDE.

W-Z

Watchpoint indicator

Use the **Watchpoint indicator** option to configure the color of watchpoints that appear in debugger windows. Click the color swatch to change the current color.

Window follows insertion point

Select this option to have the Code Completion window follow the insertion point as you edit text in the active editor window. Clear the option to leave the Code Completion window in place.

Window position and size

Select the **Window position and size** option to remember the location and dimensions of each editor window. Clear the option to open each editor window according to default settings.

NOTE The IDE must be able to write to the file in order to remember window position and size.

Working Directory

Enter the path to the default directory to which the current project has access. Debugging occurs in this location. If this field is blank, debugging occurs in the same directory as the executable file.

Workspaces

Enter the number of recent workspace files to display in the **Open Recent** submenu.

Zoom windows to full screen

Use the **Zoom windows to full screen** option to configure the behavior of the zoom box in the upper right-hand corner of all editor windows:

- Select the option to have the IDE resize a zoomed window to fill the entire screen.
- Clear the option to have the IDE resize a zoomed window to its default size.

Menus

This section contains these chapters:

- [IDE Menus](#)
- [Menu Commands](#)

IDE Menus

This chapter provides an overview of CodeWarrior™ IDE menus and their commands. The IDE provides two different arrangements of IDE menus, configurable in the **IDE Extras** preference panel:

- **Windows** menu layout
- **Macintosh** menu layout

This chapter lists the IDE menus under each menu layout. For each menu, a table shows this information:

- **Menu command**—the name of each command in the menu.
- **Description**—a short description of each command.

This chapter has these sections:

- [“Windows Menu Layout” on page 493](#)
- [“Macintosh Menu Layout” on page 507](#)

Windows Menu Layout

This section provides an overview of the menus and menu commands available in the **Windows** menu layout.

File Menu

The **File** menu contains commands for opening, creating, saving, closing, and printing source files and projects. The File menu also provides different methods for saving edited files.

IDE Menus

Windows Menu Layout

Table 30.1 File menu commands

Menu command	Explanation
New	Creates new projects using the New Project wizard or project stationery files.
Open	Opens source and project files for editing and project modification operations.
Find and Open File	Opens the file specified in the Find and Open File dialog or from the selected text in the active window. When using the Windows menu layout on a Macintosh host, hold down the Option key to change this command to Find .
Close	Closes the active window.
Save	Saves the active file using the editor window's filename. When using the Windows menu layout on a Macintosh host, hold down the Option key to change this command to Save All .
Save All	Saves all open editor windows. When using the Windows menu layout on a Macintosh host, hold down the Option key to substitute this command for the Save command.
Save As	Saves a copy of the active file under a new name and closes the original file.
Save A Copy As	Saves a copy of the active file without closing the file.
Revert	Discards all changes made to the active file since the last save operation.
Open Workspace	Opens a workspace that you previously saved.
Close Workspace	Closes the current workspace. (You cannot close the default workspace.)
Save Workspace	Saves the current state of onscreen windows, recent items, and debugging.
Save Workspace As	Saves an existing workspace under a different name.
Import Components	Imports the components from another catalog into the current catalog.

Table 30.1 File menu commands (*continued*)

Menu command	Explanation
Close Catalog	Closes the current catalog and its associated Catalog Components window and Component Palette.
Import Project	Imports a project file previously saved in extensible markup language format (XML) and converts it into project file format.
Export Project	Exports the active project file to disk in extensible markup language (XML) format.
Page Setup	Displays the Page Setup dialog for setting paper size, orientation, and other printer options.
Print	Displays the Print dialog for printing active files, and the contents of Project, Message, and Errors & Warning window contents.
Open Recent	Displays a submenu of recently opened files and projects that can be opened in the IDE.
Exit	Quits the CodeWarrior IDE. When using the Windows menu layout on a Macintosh host, this command does not appear. Instead, use the Quit or Quit CodeWarrior command in the File menu or the Quit or Quit CodeWarrior command in the CodeWarrior menu.

Edit Menu

The **Edit** menu contains all customary editing commands, along with some CodeWarrior additions. This menu also includes commands that open the Preferences and Target Settings windows.

IDE Menus

Windows Menu Layout

Table 30.2 Edit menu commands

Menu command	Explanation
Undo	Undoes the last cut, paste, clear, or typing operation. If you cannot undo the action, this command changes to Can't Undo .
Redo	Redoes the action of the last Undo operation. If you cannot redo the action, this command changes to Can't Redo .
Cut	Removes the selected text and places a copy of it on the Clipboard.
Copy	Copies the selected text and places a copy of it on the Clipboard.
Paste	Places the contents of the Clipboard at current insertion point or replaces the selected text.
Delete	Removes the selected text without placing a copy on the Clipboard. When using the Windows menu layout on a Macintosh host, this command does not appear. Instead, use the Clear command.
Select All	Selects all text in current editor window or text box for cut, copy, paste, clear, or typing operations.
Balance	Selects text between the nearest set of parenthesis, braces, or brackets.
Shift Left	Moves selected text one tab stop to the left.
Shift Right	Moves selected text one tab stop to the right.
Get Previous Completion	Shortcut for selecting the previous item that appears in the Code Completion window.
Get Next Completion	Shortcut for selecting the next item that appears in the Code Completion window.
Complete Code	Opens the Code Completion window.
Preferences	Opens the IDE Preferences window where you can set general IDE, editor, debugger, and layout options.

Table 30.2 Edit menu commands (*continued*)

Menu command	Explanation
Target Settings (the name changes, based on the name of the active build target)	Opens the project's Target Settings window where you can set target, language, code generation, linker, editor, and debugger options.
Version Control Settings	Opens the VCS Settings window to enable activation of a version control system and its relevant settings.
Commands & Key Bindings	Opens the Customize IDE Commands window where you can create, modify, remove menus, menu commands, and key bindings.

View Menu

The **View** menu contains commands for viewing toolbars, the class browser, the Message window, and debugging windows.

Table 30.3 View menu commands

Menu command	Explanation
Toolbars	Use the Toolbars menu to show, hide, reset, and clear window and main toolbars.
Project Inspector	Opens or brings to the front a Project Inspector window.
Browser Contents	Opens or brings to the front a Browser Contents window.
Class Browser	Opens or brings to the front a New Class Browser window.
Class Hierarchy or Class Hierarchy Window	Opens or brings to the front a Class Hierarchy window.
Build Progress or Build Progress Window	Opens the Build Progress window.
Errors & Warnings or Errors & Warnings Window	Opens or brings to the front an Errors & Warnings window.
Symbolics or Symbolics Window	Opens the Symbolics window.

IDE Menus

Windows Menu Layout

Table 30.3 View menu commands (*continued*)

Menu command	Explanation
Processes or Processes Window	Opens or brings to the front a Processes window.
Breakpoints or Breakpoints Window	Opens or brings to the front the Breakpoints window. Use this window to view, create, modify, and remove breakpoints.
Registers or Register Window	Opens or brings to the front a Register window.
Expressions or Expressions Window	Opens or brings to the front an Expressions window. Use to view, create, modify, and remove expressions.
Global Variables or Global Variables Window	Opens or brings to the front a Global Variables window.

Search Menu

The **Search** menu contains commands for finding text, replacing text, comparing files, and navigating code.

Table 30.4 Search menu commands

Menu command	Explanation
Find	Opens the Find and Replace window for performing searches in the active editor window.
Replace	Opens the Find and Replace window for replacing text in the active editor window.
Find in Files	Opens the Find in Files window for performing searches in the active editor window.
Find Next	Finds the next occurrence of the find string in the active editor window. When using the Windows menu layout on a Macintosh host, hold down the Shift key to change this command to Find Previous .

Table 30.4 Search menu commands (*continued*)

Menu command	Explanation
Find In Next File	Finds the next occurrence of the find string in the next file listed in the Find window's File Set. When using the Windows menu layout on a Macintosh host, hold down the Shift key to change this command to Find In Previous File .
Enter Find String	Replaces the Find text box string with the selected text. When using the Windows menu layout on a Macintosh host, hold down the Shift key to change this command to Enter Replace String .
Find Selection	Finds the next occurrence of the selected text in the active editor window. When using the Windows menu layout on a Macintosh host, hold down the Shift key to change this command to Find Previous Selection .
Replace Selection	Replaces the replace string in the Replace text box with the selected text.
Replace and Find Next	Replaces the selected text with the Replace text box string, then performs a Find Next operation. When using the Windows menu layout on a Macintosh host, hold down the Shift key to change this command to Replace and Find Previous .
Replace All	Finds all matches of the Find text box string and replaces them with the Replace text box string.
Find Definition	Searches for definition of the routine name selected in the active editor window using the project's source files. When using the Windows menu layout on a Macintosh host, hold down the Option key to change this command to Find Reference .
Go Back	Returns to the previous CodeWarrior browser view.
Go Forward	Moves to the next CodeWarrior browser view.
Go to Line	Opens the Go To Line dialog where you can specify by line number where to position the text insertion point.
Compare Files	Opens the Compare Files Setup window where you can choose to compare folders or files and merge their contents.

IDE Menus

Windows Menu Layout

Table 30.4 Search menu commands (*continued*)

Menu command	Explanation
Apply Difference	Adds, removes, or changes the selected text in the destination file to match the selected text in the source file.
Unapply Difference	Reverses the modifications made to the destination file by the Apply Difference command.

Project Menu

The **Project** menu contains commands for manipulating files, handling libraries, compiling projects, building projects, and linking projects.

Table 30.5 Project menu commands

Menu command	Explanation
Add Window	Adds the active window to the project.
Add Files	Opens a dialog box that you can use to add multiple files to the active project.
Create Group	Opens the Create Group dialog box that you can use to add a new file group to the active project. The new file group appears below the selected file or group.
Create Target	Opens the Create Target dialog box that you can use to add a new build target to the active project. The new build target appears below the selected build target.
Create Overlay or Create Segment or Create Overlay or Create Segment	Opens the Create Segment/Overlay dialog box that you can use to add a new segment or overlay to the active project. The new segment or overlay appears below the selected one.
Create Design	Opens the Create New Design dialog box that you can use to add a design to the active project. The new design appears in the Design tab of the project window.
Export Project as GNU Makefile	Exports the current project to a GNU makefile. When using the Windows menu layout on a Macintosh host, this command does not appear.

Table 30.5 Project menu commands (*continued*)

Menu command	Explanation
Check Syntax	Checks the active editor window or selected files in the project window for compilation errors.
Preprocess	Preprocesses the active editor window or selected files in the project window and displays results in a new editor window.
Precompile	Precompiles the active editor window or selected files in the project window and stores results in a new header file.
Compile	Compiles the active editor window or selected files in the project window.
Disassemble	Disassembles the active editor window or selected files in the project window and displays results in a new editor window.
Bring Up To Date	Compiles all marked or modified files in the current build target of the active project.
Make	Compiles and links all marked or modified files in the current build target of the active project, saving the executable file.
Stop Build	Stops the current compile and linking operation and cancels the remainder of the build process.
Remove Object Code	Removes object code from one or more build targets in the project. When using the Windows menu layout on a Macintosh host, hold down the Shift key to change this command to Remove Object Code & Compact .
Re-search for Files	Resets the cached locations of source files using the project access paths, and stores them for faster builds and project operations.
Reset Project Entry Paths	Resets the location of all source files in the active project using the project access paths.
Synchronize Modification Dates	Updates the modification dates of all source files in the active project.

IDE Menus

Windows Menu Layout

Table 30.5 Project menu commands (*continued*)

Menu command	Explanation
Debug or Resume	Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.
Run	Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.
Set Default Project	Uses the Set Default Project menu to choose the default project when more than one project is open in the IDE.
Set Default Target	Uses the Set Default Target menu to choose the default build target when more than one build target is present in the project file.

Debug Menu

The **Debug** menu contains commands for managing program execution.

Table 30.6 Debug menu commands

Menu command	Explanation
Break	Pauses execution of the program in a debugging session to enable examination of register and variable contents
Kill	Terminates the current debugging session returning control to the IDE.
Restart	Terminates the current debugging session, then restarts the program from the beginning.
Step Over	Executes each source line in the program, treating routine calls as a single statement and stopping the program at the next line of code.
Step Into	Executes each source line in the program, following any subroutine calls.

Table 30.6 Debug menu commands (*continued*)

Menu command	Explanation
Step Out	Executes each source line in the subroutine and stops the program when the routine returns to its caller.
Run to Cursor	Sets a temporary breakpoint on the source line containing the insertion point.
Change Program Counter	Opens the Change Program Counter dialog box that you can use to move the current statement arrow to an address or symbol.
Set Breakpoint or Clear Breakpoint	Sets a breakpoint on the source line containing the insertion point. Clears the breakpoint on the source line containing the insertion point.
Set Eventpoint	Sets an eventpoint on the source line containing the insertion point.
Clear Eventpoint	Clears the breakpoint on the source line containing the insertion point.
Set/Clear Breakpoint	Opens the Set/Clear Breakpoint dialog box that you can use for setting or clearing breakpoints by address or symbol.
Enable Breakpoint or Disable Breakpoint	Activates the disabled breakpoint on the source line containing the insertion point. De-activates the breakpoint on the source line containing the insertion point.
Clear All Breakpoints	Clears all breakpoints currently set in the default build target of the active project.
Show Breakpoints or Hide Breakpoints	Adds a Breakpoint Column to all project editor windows where you can set, view, or clear breakpoints. Removes the Breakpoint Column from all project editor windows.
Set Watchpoint or Clear Watchpoint	Sets a watchpoint on the source line containing the insertion point. Clears the watchpoint on the source line containing the insertion point.

IDE Menus

Windows Menu Layout

Table 30.6 Debug menu commands (*continued*)

Menu command	Explanation
Enable Watchpoint or Disable Watchpoint	Activates the disabled watchpoint on the source line containing the insertion point. De-activates the watchpoint on the source line containing the insertion point.
Clear All Watchpoints	Clears all watchpoints currently set in the default build target of the active project.
Break on C++ Exception	Configures the debugger to break at <code>__throw()</code> each time a C++ exception occurs.
Break on Java Exceptions	Use this menu to select the Java exceptions on which the debugger breaks.
Connect	Establishes communication with an embedded device to start a debugging session. When using the Windows menu layout on a Macintosh host, this command does not appear.

Data Menu

The **Data** menu contains commands that control how the CodeWarrior debugger displays data values. This menu appears only during a debugging session.

Table 30.7 Data menu commands

Menu command	Explanation
Show Types	Toggles the appearance of the data type on local and global variables displayed in Variable panes and Variable windows.
Refresh All Data	Updates data displays.
New Expression	Creates a new expression entry in the Expressions window.
Copy to Expression	Copies the selected variable to the Expressions window.
View As	Displays the View As dialog where the data type of the selected variable can be specified.
View Variable	Displays the selected variable in a new Variables window.

Table 30.7 Data menu commands (*continued*)

Menu command	Explanation
View Array	Displays the selected array variable in a new Arrays window.
View Memory	Displays the selected variable in a new Memory window.
View Memory As	Displays the View As dialog where the data type of the selected variable can be specified, then shown in a new Memory window.
Cycle View	Toggles the data view among View Source , View Disassembly , View Mixed , and View Raw Data .
View Source	View data as source code.
View Disassembly	View data as language disassembly.
View Mixed	View data as source code and its disassembly.
View Raw Data	View data without applied formatting.
View As Default	Views the selected variable in the default value format.
View As Binary	Views the selected variable as a binary value.
View As Signed Decimal	Views the selected variable as a signed decimal value.
View As Unsigned Decimal	Views the selected variable as an unsigned decimal value.
View As Hexadecimal	Views the selected variable as a hexadecimal value.
View As Character	Views the selected variable as a character value.
View As C String	Views the selected variable as a C string.
View As Pascal String	Views the selected variable as a Pascal string.
View As Unicode String	Views the selected variable as a Unicode string.
View As Floating Point	Views the selected variable as a floating point value.
View As Enumeration	Views the selected variable as an enumerated value.
View As Fixed	Views the selected variable as a 32-bit fixed value.

IDE Menus

Windows Menu Layout

Window Menu

The **Window** menu contains commands that manipulate IDE windows.

The menu lists the names of all open file and project windows. A checkmark appears beside the active window, and an underline indicates a modified and unsaved file.

Table 30.8 Window menu commands

Menu command	Explanation
Close	Closes the active window. When using the Windows menu layout on a Macintosh host, hold down the Option key to change this command to Close All .
Close All	Closes all non-project windows. When using the Windows menu layout on a Macintosh host, hold down the Option key to substitute this command for the Close command.
Cascade	Arranges all editor windows so that only the title bar is visible.
Tile Horizontally	Tiles all editor windows horizontally on the screen so none overlap.
Tile Vertically	Tiles all editor windows vertically on the screen so none overlap.
Save Default Window	Saves the active browser windows settings and applies it to other browser windows as they are opened.

Help Menu

The **Help** menu contains commands for accessing the IDE's online help.

Table 30.9 Help menu commands

Menu command	Explanation
CodeWarrior Help	Launches a help viewer to display the online help. Click on a link to view a specific IDE topic.
Index	Launches a help viewer to display a glossary of common terms used in the CodeWarrior help and manuals.
Search	Launches a help viewer to a page for searching the CodeWarrior help and manuals.
Metrowerks Website	Launches a browser and automatically points you to the Metrowerks web site.
About Metrowerks CodeWarrior	Displays the CodeWarrior IDE version and build number information.

Macintosh Menu Layout

This section provides an overview of the menus and menu commands available in the **Macintosh** menu layout.

Apple Menu

The **Apple** menu (Mac OS 9.x.x and earlier) provides access to the CodeWarrior About box, shows system applications, and lists additional items.

Select [About Metrowerks CodeWarrior](#) to display the IDE version and build-number information.

When using the Macintosh menu layout on a Windows host, this menu does not appear.

CodeWarrior Menu

The **CodeWarrior Menu** (visible in Mac OS X only) provides access to the CodeWarrior About box, IDE preferences, and the command that quits the IDE.

When using the Macintosh menu layout on a Windows host, this menu does not appear.

IDE Menus

Macintosh Menu Layout

Table 30.10 Apple menu commands

Menu command	Explanation
About Metrowerks CodeWarrior	Displays the CodeWarrior IDE version and build number information.
Preferences	Opens the IDE Preferences window where you can set general IDE, editor, debugger, and layout options.
Quit or Quit CodeWarrior	Quits the CodeWarrior IDE.

File Menu

The **File** menu contains commands for opening, creating, saving, closing, and printing source files and projects. The File menu also provides different methods for saving edited files.

Table 30.11 File menu commands

Menu command	Explanation
New Text File	Creates a new text file and displays it in a new editor window.
New	Creates new projects using the New Project wizard or project stationery files.
Open	Opens source and project files for editing and project modification operations.
Open Recent	Displays a submenu of recently opened files and projects that can be chosen to open in the IDE.
Find and Open File	Opens the file specified in the Find and Open File dialog or from the selected text in the active window.
Close	Closes the active window. When using the Macintosh menu layout on a Macintosh host, hold down the Option key to change this command to Close All .

Table 30.11 File menu commands (*continued*)

Menu command	Explanation
Save	Saves the active file using the editor window's filename. When using the Macintosh menu layout on a Macintosh host, hold down the Option key to change this command to Save All .
Save As	Saves a copy of the active file under a new name and closes the original file.
Save A Copy As	Saves a copy of the active file without closing the file.
Revert	Discards all changes made to the active file since the last save operation.
Open Workspace	Opens a workspace that you previously saved.
Close Workspace	Closes the current workspace. (You cannot close the default workspace.)
Save Workspace	Saves the current state of onscreen windows, recent items, and debugging.
Save Workspace As	Saves an existing workspace under a different name.
Import Components	Imports the components from another catalog into the current catalog.
Close Catalog	Closes the current catalog and its associated Catalog Components window and Component Palette.
Import Project	Imports a project file previously saved in extensible markup language format (XML) and converts it into project file format.
Export Project	Exports the active project file to disk in XML format.
Page Setup	Displays the Page Setup dialog for setting paper size, orientation, and other printer options.
Print	Displays the Print dialog for printing active files, and the contents of Project, Message, and Errors & Warning window contents.
Quit or Quit CodeWarrior (Classic Mac OS)	Quits the CodeWarrior IDE.

IDE Menus

Macintosh Menu Layout

Edit Menu

The **Edit** menu contains all customary editing commands, along with some CodeWarrior additions. This menu also includes the commands that open the Preferences and Target Settings windows.

Table 30.12 Edit menu commands

Menu command	Explanation
Undo	Undoes the action of the last cut, paste, clear or typing operation. If you cannot undo the action, this command changes to Can't Undo .
Redo	Redoes the action of the last Undo operation. If you cannot redo the action, this command changes to Can't Redo .
Cut	Removes the selected text and places a copy of it on the Clipboard.
Copy	Copies the selected text and places a copy of it on the Clipboard.
Paste	Places the contents of the Clipboard at current insertion point or replaces the selected text.
Clear	Removes the selected text without placing a copy on the Clipboard. When using the Macintosh menu layout on a Windows host, this command does not appear. Instead, use the Delete command.
Select All	Selects all text in the current editor window or text box for cut, copy, paste, clear, or typing operations.
Balance	Selects text between the nearest set of parenthesis, braces, or brackets.
Shift Left	Moves selected text one tab stop to the left.
Shift Right	Moves selected text one tab stop to the right.
Get Previous Completion	Shortcut for selecting the previous item that appears in the Code Completion window.

Table 30.12 Edit menu commands (*continued*)

Menu command	Explanation
Get Next Completion	Shortcut for selecting the next item that appears in the Code Completion window.
Complete Code	Opens the Code Completion window.
Insert Reference Template	Inserts a routine template corresponding to the selected Mac OS Toolbox call in the active window. When using the Macintosh menu layout on a Windows host, this command does not appear.
Preferences	Opens the IDE Preferences window where you can set general IDE, editor, debugger, and layout options.
Target Settings (name changes, based on name of active build target)	Opens the project's Target Settings window where you can set target, language, linker, editor, and debugger options.
Version Control Settings	Opens the VCS Settings window to enable activation of a version control system and its relevant settings.
Commands & Key Bindings	Opens the Customize IDE Commands window where you can create, modify, remove menus, menu commands, and key bindings.

Search Menu

The **Search** menu contains commands for finding text, replacing text, comparing files, navigating code, and finding routine definitions.

Table 30.13 Search menu commands

Menu command	Explanation
Find and Replace	Opens the Find and Replace window for performing find and replace operations on the active editor window.
Find in Files	Opens the Find in Files window for performing searches in the active editor window.

IDE Menus

Macintosh Menu Layout

Table 30.13 Search menu commands (*continued*)

Menu command	Explanation
Find Next	Finds the next occurrence of the find string in the active editor window. When using the Macintosh menu layout on a Macintosh host, hold down the Shift key to change this command to Find Previous .
Find In Next File	Finds the next occurrence of the find string in the next file listed in the Find window's File Set. When using the Macintosh menu layout on a Macintosh host, hold down the Shift key to change this command to Find In Previous File .
Enter Find String	Replaces the Find text box string with the selected text. When using the Macintosh menu layout on a Macintosh host, hold down the Shift key to change this command to Enter Replace String .
Find Selection	Finds the next occurrence of the selected text in the active editor window. When using the Macintosh menu layout on a Macintosh host, hold down the Shift key to change this command to Find Previous Selection .
Replace Selection	Replaces the replace string in the Replace text box with the selected text.
Replace and Find Next	Replaces the selected text with the Replace text box string, then performs a Find Next operation. When using the Macintosh menu layout on a Macintosh host, hold down the Shift key to change this command to Replace and Find Previous .
Replace All	Finds all matches of the Find text box string and replaces them with the Replace text box string.
Find Definition	Searches for the definition of the routine name selected in the active editor window using the project's source files. When using the Macintosh menu layout on a Macintosh host, hold down the Shift key to change this command to Find Definition & Reference .

Table 30.13 Search menu commands (*continued*)

Menu command	Explanation
Find Reference	Searches for the definition of the routine name selected in the active editor window using the specified online help system. When using the Macintosh menu layout on a Windows host, this command does not appear.
Go Back	Returns to the previous CodeWarrior browser view.
Go Forward	Moves to the next CodeWarrior browser view.
Go to Line	Opens the Go To Line dialog where you can specify by line number where to position the text insertion point.
Compare Files	Opens the Compare Files Setup window where you can choose to compare folders or files and merge their contents.
Apply Difference	Adds, removes, or changes the selected text in the destination file to match the selected text in the source file.
Unapply Difference	Reverses the modifications made to the destination file by the Apply Difference command.

Project Menu

The **Project** menu contains commands for manipulating files, handling libraries, compiling projects, building projects, and linking projects.

Table 30.14 Project menu commands

Menu command	Explanation
Add Window (name changes, based on name of selected item)	Adds the active window to the project.
Add Files	Opens a dialog that you can use to add multiple files to the active project.

IDE Menus

Macintosh Menu Layout

Table 30.14 Project menu commands (*continued*)

Menu command	Explanation
Create Group or Create Target or Create Overlay or Create Segment or Create Overlay or Create Segment	Displays the Create Group dialog where you can add a new file group to the active project immediately after selected file or group. Displays the Create Target dialog where you can add a new build target to the active project immediately after selected build target. Displays the Create Overlay dialog where you can add a new memory overlay to the active project immediately after the selected overlay. Displays the Create Segment dialog where you can add a new segment to the active project immediately after the selected segment.
Create Design	Opens the Create New Design dialog box that you can use to add a design to the active project. The new design appears in the Design tab of the project window.
Check Syntax	Checks the active editor window or selected files in the project window for compilation errors.
Preprocess	Preprocesses the active editor window or selected files in the project window and displays results in a new editor window.
Precompile	Precompiles active editor window or selected files in project window and stores results in a new header file.
Compile	Compiles the active editor window or selected files in project window.
Disassemble	Disassembles the active editor window or selected files in the project window and displays the results in a new editor window.
Bring Up To Date	Compiles all marked or modified files in the current build target of the active project.
Make	Compiles and links all marked or modified files in the current build target of the active project; saving the executable file.
Stop Build	Stops the current compile and link operation and cancels the remainder of the build process.

Table 30.14 Project menu commands (*continued*)

Menu command	Explanation
Remove Object Code	Removes the object code from one or more build targets in the project. When using the Macintosh menu layout on a Macintosh host, hold down the Option key to change this command to Remove Object Code & Compact .
Re-search for Files	Resets the cached locations of source files using the project access paths, and storing them for faster builds and project operations.
Reset Project Entry Paths	Resets the location of all source files in the active project using the project access paths.
Synchronize Modification Dates	Updates the modification dates of all source files in the active project.
Debug or Resume	Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file. Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.
Run	Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.
Set Default Project	Uses the Set Default Project menu to choose the default project when more than one project is open in the IDE.
Set Default Target	Uses the Set Default Target menu to choose the default build target when more than one build target is present in the project file.

Debug Menu

The **Debug** menu contains commands for managing program execution.

IDE Menus

Macintosh Menu Layout

Table 30.15 Debug menu commands

Menu command	Explanation
Kill	Terminates the current debugging session returning control to the IDE.
Restart	Terminates the current debugging session, then restarts the program from the beginning.
Step Over	Executes each source line in the program, treating routine calls as a single statement and stopping the program at the next line of code.
Step Into	Executes each source line in the program, following any subroutine calls.
Step Out	Executes each source line in the subroutine and stops the program when the routine returns to its caller.
Stop	Pauses execution of the program in a debugging session to enable examination of register and variable contents.
Set Breakpoint or Clear Breakpoint	Sets a breakpoint on the source line containing the insertion point. Clears the breakpoint on the source line containing the insertion point.
Set Eventpoint	Sets an eventpoint on the source line containing the insertion point.
Clear Eventpoint	Clears the breakpoint on the source line containing the insertion point.
Set/Clear Breakpoint	Displays the Set/Clear Breakpoint dialog for setting or clearing breakpoints by address or symbol.
Enable Breakpoint or Disable Breakpoint	Activates the disabled breakpoint on the source line containing the insertion point. De-activates the breakpoint on the source line containing the insertion point.
Clear All Breakpoints	Clears all breakpoints currently set in the default build target of the active project.

Table 30.15 Debug menu commands (*continued*)

Menu command	Explanation
Show Breakpoints or Hide Breakpoints	Adds a Breakpoint Column to all project editor windows where breakpoints can be set, viewed, and cleared. Removes the Breakpoint Column from all project editor windows.
Set Watchpoint or Clear Watchpoint	Sets a watchpoint on the source line containing the insertion point. Clears the watchpoint on the source line containing the insertion point.
Enable Watchpoint or Disable Watchpoint	Activates the disabled watchpoint on the source line containing the insertion point. De-activates the watchpoint on the source line containing the insertion point.
Clear All Watchpoints	Clears all watchpoints currently set in the default build target of the active project.
Run to Cursor	Sets a temporary breakpoint on the source line containing the insertion point.
Change Program Counter	Displays the Change Program Counter dialog where you can move the current statement arrow to an address or symbol.
Break on C++ Exception	Configures the debugger to break at <code>__throw()</code> each time a C++ exception occurs.
Break on Java Exceptions	Use this menu to select which Java exceptions the debugger should break on.
Switch to Monitor	Configures the IDE to use an external debugger instead of the CodeWarrior debugger. When using the Macintosh menu layout on a Windows host, this command does not appear.

Data Menu

The **Data** menu contains commands that control how the CodeWarrior debugger displays data values. This menu only appears during a debugging session.

IDE Menus

Macintosh Menu Layout

Table 30.16 Data menu commands

Menu command	Description
Show Types	Toggles the appearance of the data type on local and global variables displayed in Variable panes and Variable windows.
Refresh All Data	Updates data displays.
New Expression	Creates a new expression entry in the Expressions window.
Copy to Expression	Copies the selected variable to the Expressions window.
View As	Displays the View As dialog where the data type of the selected variable can be specified.
View Variable	Displays the selected variable in a new Variables window.
View Array	Displays the selected array variable in a new Arrays window.
View Memory	Displays the selected variable in a new Memory window.
View Memory As	Displays the View As dialog where the data type of the selected variable can be specified, then shown in a new Memory window.
Cycle View	Toggles the data view among View Source , View Disassembly , View Mixed , and View Raw Data .
View Source	View data as source code.
View Disassembly	View data as disassembled.
View Mixed	View data as source code and disassembled.
View Raw Data	View data without applied formatting.
View As Default	Views selected variable in default value format.
View As Binary	Views selected variable as a binary value.
View As Signed Decimal	Views selected variable as a signed decimal value.
View As Unsigned Decimal	Views selected variable as an unsigned decimal value.

Table 30.16 Data menu commands (*continued*)

Menu command	Description
<u>View As Hexadecimal</u>	Views selected variable as a hexadecimal value.
<u>View As Character</u>	Views selected variable as a character value.
<u>View As C String</u>	Views selected variable as a C string.
<u>View As Pascal String</u>	Views selected variable as a Pascal string.
<u>View As Unicode String</u>	Views selected variable as a Unicode string.
<u>View As Floating Point</u>	Views selected variable as a floating point value.
<u>View As Enumeration</u>	Views selected variable as an enumerated value.
<u>View As Fixed</u>	Views selected variable as a 32-bit fixed value.
<u>View As Fract</u>	Views selected variable as a fract value. When using the Macintosh menu layout on a Windows host, this command does not appear.

Window Menu

The **Window** menu contains commands that manipulate IDE windows. The **Window** menu is divided into several sections:

- window commands to stack, tile, zoom, collapse, and save window positions.
- toolbar submenu for showing, hiding, resetting, and clearing window and floating toolbars.
- commands to open specific browser, IDE, and debugger windows.
- names of all open file and project windows.

A check mark appears beside the active window, and an underline denotes a modified and unsaved file.

IDE Menus

Macintosh Menu Layout

Table 30.17 Window menu commands

Menu command	Description
Stack Editor Windows	Arranges all editor windows so that only the title bar is visible.
Tile Editor Windows	Tiles all editor windows horizontally on the screen so none overlap.
Tile Editor Windows Vertically	Tiles all editor windows vertically on the screen so none overlap.
Zoom Window	Restores the active editor window to its previous size and position.
Collapse Window (Minimize Window) or Expand Window (Maximize Window)	Collapses the active editor window so that only its title bar is visible. Expands the collapsed editor window to its previous size and position.
Save Default Window	Saves the current browser-window settings for later re-use.
Toolbars	Use the Toolbars submenu to show, hide, reset, and clear window, main, and floating toolbars.
Browser Contents	Opens or brings to the front a Browser Contents window.
Class Hierarchy or Class Hierarchy Window	Opens or brings to the front a Class Hierarchy window.
New Class Browser	Opens or brings to the front a New Class Browser window.
Build Progress or Build Progress Window	Opens or brings to the front a Build Progress window.
Errors & Warnings or Errors & Warnings Window	Opens or brings to the front an Errors & Warnings window.
Project Inspector	Opens or brings to the front a Project Inspector window.
ToolServer Worksheet	Opens or brings to the front a ToolServer Worksheet window. When using the Macintosh menu layout on a Windows host, this command does not appear.

Table 30.17 Window menu commands (*continued*)

Menu command	Description
Symbolics or Symbolics Window	Opens or brings to the front a Symbolics window.
Processes or Processes Window	Opens or brings to the front a Processes window.
Expressions or Expressions Window	Opens or brings to the front an Expressions window. Use to view, create, modify, and remove expressions.
Global Variables or Global Variables Window	Opens or brings to the front a Global Variables window.
Breakpoints or Breakpoints Window	Opens or brings to the front a Breakpoints window. Use to view, create, modify, and remove breakpoints.
Registers or Register Window	Opens or brings to the front a Register window.

VCS Menu

The **VCS** (Version Control System) menu appears in the IDE's menu bar when the **Use Version Control** option is enabled. The CodeWarrior IDE can operate with many difference version control systems including CVS, Visual SourceSafe, and others.



This icon represents the VCS menu in the Macintosh-hosted IDE menu bar.

Refer to the documentation that came with the version control system to learn about using it with the CodeWarrior IDE.

Tools Menu

On the Mac OS hosted IDE, the **Tools** menu appears in the IDE's menu bar after you enable the **Use ToolServer menu** checkbox in the [IDE Extras](#) preference panel. The Tools menu contains commands for controlling Apple® ToolServer™ and Macintosh Programmer's Workbench (MPW).



Macintosh: This icon represents the Tools menu in the Macintosh-hosted IDE menu bar.

IDE Menus

Macintosh Menu Layout

Refer to *Targeting Mac OS* to learn about using ToolServer and MPW with projects.

Scripts Menu

The **Scripts** menu appears in the IDE's menu bar after you enable the **Use script menu** checkbox in the [IDE Extras](#) preference panel and creates a (Scripts) folder in the Metrowerks CodeWarrior folder of your IDE installation.

The Scripts menu uses the directory structure of the (Scripts) folder to create a hierarchical menu. This hierarchical menu lists all scripts in the (Scripts) folder. Open a script-editing utility or a text editor to learn more about the scripts that might already exist in the (Scripts) folder.

Refer to the *CodeWarrior Scripting Reference* to learn about scripting the IDE.

Help Menu

The **Help** menu contains commands for accessing the IDE's online help.

Table 30.18 Help menu commands

Menu command	Description
CodeWarrior Help	Launches a help viewer to display the online help. Click on a link to view a specific IDE topic.
Index (Windows)	Opens the online help to the Index tab.
Search (Windows)	Opens the online help to the Search tab.
Metrowerks Website	Launches a browser and automatically points you to the Metrowerks website.
About Metrowerks CodeWarrior (Windows)	Displays the CodeWarrior IDE version and build number information.

NOTE Classic Macintosh: The Help menu contains additional menu commands for Balloon Help and accessing Apple's online help.

Menu Commands

This section presents an alphabetical listing of all available menu commands in the CodeWarrior™ IDE. Menu commands that appear only on certain host platforms are documented. A menu command that has no host information is available on all hosts.

Use this listing as a reference to find information about a specific menu command.

A

About Metrowerks CodeWarrior

This command displays the CodeWarrior IDE version and build number information.

TIP Click the **Installed Products** button in this window to view and save information about installed products and plug-ins for the CodeWarrior IDE. You can also use this window to enable or disable plug-in diagnostics.

Add Files

The **Add Files** command opens a dialog which allows one or more files to be added to the project.

Add Window

The **Add Window** command adds the file in the active Editor window to the open project. The name of the menu command changes, based on the name of the active window. For example, if the name of the active window is `MyFile`, the name of the menu command changes to **Add MyFile to Project**.

Align

Reveals the **Align** submenu with component alignment commands like Right Edges, Vertical Centers, and others.

See also:

- [“Bottom Edges” on page 525](#)
- [“Horizontal Center” on page 538](#)
- [“Left Edges” on page 539](#)
- [“Right Edges” on page 547](#)
- [“To Grid” on page 553](#)
- [“Top Edges” on page 554](#)
- [“Vertical Center” on page 556](#)

All Exceptions

The **All Exceptions** command of the **Java** submenu tells the debugger to break every time an exception occurs. This behavior includes exceptions thrown by the virtual machine, your own classes, the debugger, classes in `classes.zip`, and so on. Java programs throw many exceptions in the normal course of execution, so catching all exceptions causes the debugger to break often.

Anchor Floating Toolbar

The **Anchor Floating Toolbar** command attaches the floating toolbar beneath the menu bar. Once attached, the anchored toolbar can not be moved again until it is unanchored.

See also: [“Unanchor Floating Toolbar” on page 555](#)

Apply Difference

The **Apply Difference** command applies the selected difference from the source file into the destination file.

B**Balance**

The **Balance** command selects all text starting at the current insertion point and enclosed in parentheses (), brackets [], or braces { },

Bottom Edges

The **Bottom Edges** command of the **Align** submenu aligns the bottom edges of the selected components.

Break

The **Break** command temporarily suspends execution of the target program and returns control to the debugger.

See also: [“Stop” on page 551](#).

Break on C++ Exception

The **Break on C++ Exception** command tells the debugger to break at __throw() each time a C++ exception occurs.

Break on Java Exceptions

The **Break on Java Exceptions** command reveals the Java Exceptions submenu.

See also:

- [“Exceptions in Targeted Classes” on page 533](#)
- [“Uncaught Exceptions Only” on page 555](#).

Breakpoints or Breakpoints Window

These commands open the Breakpoints window.

Bring To Front

The **Bring To Front** command moves the selected objects so that they are displayed in front of all other objects.

Bring Up To Date

The **Bring Up To Date** command updates the current build target in the active project by compiling all of the build target's modified and touched files.

Browser Contents

The **Browser Contents** command opens the Browser Contents window. This command is not available if the Enable Browser option is not activated.

Build Progress or Build Progress Window

These commands open the Build Progress window. Use it to monitor the IDE's status as it compiles a project.

C

Cascade

The **Cascade** command arranges open editor windows one on top of another, with their window titles visible.

Change Program Counter

The **Change Program Counter** command opens a window that lets you move the current-statement arrow to a particular address or symbol.

Check Syntax

The **Check Syntax** command checks the syntax of the source file in the active Editor window or the selected files in the open project window. If the IDE detects one or more errors, a Message window appears and shows information about the errors.

The **Check Syntax** command is not available if the active Editor window is empty or no project file is open.

Check Syntax does not generate object code.

[Table 31.1](#) explains how to abort the syntax-checking process:

Table 31.1 Aborting the syntax-checking process

On this host...	Do this...
Windows	Press Esc
Macintosh	Press Command-. (Command-Period)
Solaris	Press Esc
Linux	Press Esc

Class Browser

The **Class Browser** command opens a Class Browser window. This command is unavailable if the **Enable Browser** option is not enabled.

Class Hierarchy or Class Hierarchy Window

These commands open a Multi-Class Browser window. This command is unavailable if the **Enable Browser** option is not enabled.

Clear

The **Clear** command removes the selected text. This menu command is equivalent to pressing the Backspace or Delete key.

Clear All Breakpoints

The **Clear All Breakpoints** command clears all breakpoints in all source files belonging to the target program.

Clear All Watchpoints

The **Clear All Watchpoints** command clears all watchpoints in the current program.

Clear Breakpoint

The **Clear Breakpoint** command clears the breakpoint at the currently selected line. If the **Show Breakpoints** option is enabled, the marker in the Breakpoints column of the Editor window disappears.

Clear Eventpoint

This command opens a submenu that lets you remove an eventpoint from the currently selected line. If the **Show Breakpoints** option is active, the Breakpoints column in the editor windows shows a marker next to each line with an eventpoint. The marker represents the eventpoint type.

Clear Floating Toolbar

The **Clear Floating Toolbar** command removes all shortcut icons from the floating toolbar. Once the toolbar is cleared, drag shortcut icons from the Commands and Key Bindings window to the toolbar to create a custom floating toolbar.

Clear Main Toolbar

The **Clear Main Toolbar** command removes all shortcut icons from the main toolbar. Once the toolbar is cleared, drag shortcut icons from the Commands and Key Bindings window to the toolbar to create a custom main toolbar.

Clear Watchpoint

The **Clear Watchpoint** command removes a watchpoint from the selected variable or memory range.

Clear Window Toolbar

The **Clear Window Toolbar** command removes all shortcut icons from the window toolbar. Once the toolbar is cleared, drag shortcut icons from the Commands and Key Bindings window to the toolbar to create a custom window toolbar.

Close

The **Close** command closes the active window.

Close All

The **Close All** command closes all open windows of a certain type. The name of this menu command changes, based on the type of item selected. For example, select one of several open editor windows, the menu command changes its name to **Close All Editor Documents**.

NOTE Macintosh: Press the Option key to change Close to Close All.

Close Catalog

The **Close Catalog** command closes the current catalog and removes the catalog from the Component Catalog window and the Component Palette.

Close Workspace

This command closes the current workspace.

You cannot close the default workspace, but you can choose whether to use it by toggling the [Use default workspace](#) option in the [IDE Extras](#) preference panel.

Commands & Key Bindings

The **Commands and Key Bindings** command opens the Customize IDE Commands window.

Complete Code

The **Complete Code** command opens the Code Completion window. Use this window to help you automatically complete programming-language symbols as you type them in the active editor window.

CodeWarrior Glossary

The **CodeWarrior Glossary** command opens and displays a list of vocabulary terms used by the CodeWarrior manuals and online help.

CodeWarrior Help

This command opens the online help for the CodeWarrior IDE.

Collapse Window

The **Collapse Window** command collapses the active window so that only its title is visible.

Compare Files

The **Compare Files** command opens the Compare Files Setup window. Use it to choose two files or folders for comparison and merging. After choosing the items, a comparison window appears that shows differences between the items.

Compile

The **Compile** command compiles selected source files into binary files. The IDE compiles source files that are:

- part of the current project and open in the active Editor window, or
- selected files, segments, or groups in a project window.

Connect

The **Connect** command establishes communication between the IDE and embedded hardware to begin a debugging session.

Copy

The **Copy** command copies selected text to the system Clipboard. If the Message Window is active, the Copy command copies all text in the Message Window to the Clipboard.

Copy to Expression

The **Copy to Expression** command copies the variable selected in the active pane to the Expressions window.

Create Design

This command creates a new design in the current project. The new design appears in the **Design** tab of the project window. You cannot create a design if each build target in the project already belongs to a design.

Create Group

The **Create Group** command creates a new group in the current project. This command is active when the **Files** view is visible in the project window.

Create Overlay or Create Segment

These commands create a new segment or overlay in the current project. This command is active when the **Segments** view or **Overlays** view is visible in the project window.

Create Target

The **Create Target** command creates a new build target in the current project. This command is active when the **Targets** view is visible in the project window.

Cut

The **Cut** command copies the selected text to the system Clipboard, replacing the previous Clipboard contents, and removes it from the current document or text box.

Cycle View

Toggles view among various data formats.

See also:

- [“View Disassembly” on page 557](#)
- [“View Mixed” on page 558](#)
- [“View Raw Data” on page 558](#)
- [“View Source” on page 558](#)

D

Debug

This command compiles and links a project, then runs the CodeWarrior debugger with the project’s code. If debugging is active, the debugging window appears to examine program information and step through the code as it executes. If debugging is not active, the window appears, but the program executes without stopping in the debugger.

Delete

The **Delete** command removes selected text without placing it on the system clipboard. This menu command is equivalent to pressing the Backspace or Delete key.

Disable Breakpoint

The **Disable Breakpoint** command de-activates the breakpoint at the currently selected line.

Disable Watchpoint

The **Disable Watchpoint** command de-activates a watchpoint for the selected variable or memory range.

Disassemble

The **Disassemble** command disassembles the compiled source files selected in the project window. After disassembling a file, the IDE creates a .dump file that contains the file's object code. The .dump file appears in a new window after the IDE completes the disassembly process.

Display Grid

The **Display Grid** command toggles the visibility of grid lines in the layout window. When checked, the grid lines appear, otherwise, no grid is visible.

E

Enable Breakpoint

The **Enable Breakpoint** command activates a breakpoint at the currently selected line. The breakpoint appears in the left side of the editor window if the Breakpoint column is visible. The states of the breakpoint marker include:

- ● enabled breakpoint.
- ○ disabled breakpoint.
- - no breakpoint in line.

Enable Watchpoint

The **Enable Watchpoint** command activates a watchpoint for the selected variable or memory range.

Enabled watchpoints are indicated by an underline of the selected variable or range of memory. Disabled watchpoints have a grey underline. The underline's color can be configured in the Display Settings preference panel of the IDE Preference window.

Enter Find String

The **Enter Find String** command copies selected text in the active window directly into the target search string. It will then appear in the **Find** text box of both the **Find and Replace** and **Find in Files** windows. Once done, use any of the find commands to search for matches without opening any Find-related windows.

Enter Replace String

The **Enter Replace String** command copies the selected text in the active window directly into the target search string. It will then appear in the **Replace with** text box of both the **Find and Replace** and **Find in Files** windows. Once done, use any of the find commands to search for matches without opening any Find-related windows.

NOTE Macintosh: Press the Shift key to change the Enter Find String command to the Enter Replace String menu command.

Errors & Warnings or Errors & Warnings Window

These commands open the Errors and Warnings window.

Exceptions in Targeted Classes

The **Exceptions in Targeted Classes** command of the **Java** submenu instructs the debugger to break on exceptions thrown by your own classes in the project. Choose this command to break on exceptions thrown by your classes, rather than exceptions that Java programs throw in the normal course of execution.

Exit

The **Exit** command exits the CodeWarrior IDE immediately, provided that:

- all changes to the open editor files are already saved, or
- the open editor files are not changed.

If a Project window is open, the IDE saves all changes to the project file before exiting. If an Editor window is open and changes are not saved, the CodeWarrior IDE asks if you want to save your changes before exiting.

Expand Window

The **Expand Window** command expands a collapsed window (a window with only its title visible). Only available when a collapsed window is currently active.

Export Project

The **Export Project** command exports a CodeWarrior project to a file in XML format. The IDE prompts for a name and location to save the new XML file.

Export Project as GNU Makefile

This command exports a CodeWarrior project to a GNU makefile. The IDE displays a message that tells you the name of the makefile and its location on the hard disk.

Expressions or Expressions Window

These commands open an Expressions window.

F

Find

The **Find** command opens the Find and Replace window to perform find operations within the active file.

Find Definition & Reference

The **Find Definition & Reference** command searches for the definition of the selected routine name in the active Editor window. Searching starts within the source files belonging to the open project. If the IDE does not find a definition, a system beep sounds.

If the IDE does not find the routine definition within the project files, searching continues, using the online help system specified in the **IDE Extras** preference panel.

NOTE Macintosh: Press the Option key to change the Find Definition menu command to the Find Definition & Reference menu command.

Find Definition

The **Find Definition** command searches for the definition of the selected routine name in the active window. Searching occurs in the source files belonging to the open project. If the IDE finds the definition, the source file that contains the definition appears in an Editor window, and the routine name appears highlighted.

If the IDE finds more than one definition, a Message window appears warning of multiple definitions. If the IDE does not find a definition, a system beep sounds.

NOTE Select the **Activate Browser** option in the **Build Extras** target settings panel and re-compile the project in order to use the **Find Definition** command.

Find in Files

The **Find in Files** command opens the Find in Files window. This window allows you to perform find-and-replace operations across multiple files using specified search criteria.

Find In Next File

The **Find in Next File** command searches for the next occurrence of the **Find** text box string in the next file listed in the Find in Files window.

Find In Previous File

This command searches for the next occurrence of the **Find** text box string in the previous file listed in the Find in Files window.

NOTE (Macintosh) Press the Shift key to change the Find In Next File menu command to the Find In Previous File menu command.

Find Next

The **Find Next** command searches for the next occurrence of the Find text box string in the active window.

Find and Open File

The **Find and Open File** command opens the Find and Open File dialog. Enter a filename, click OK, and the IDE searches the current project access paths as specified in the Access Paths panel of the Target Settings window.

Find and Open ‘Filename’

The **Find and Open ‘Filename’** command opens an existing text file, using the currently selected text in the Editor window as the filename.

Find Previous

The **Find Previous** command searches for the previous occurrence of the user defined string in the active window.

NOTE Macintosh: Press the Shift key to change the Find Next menu command to the Find Previous menu command.

Find Previous Selection

The **Find Previous Selection** searches for the previous occurrence of the selected text in the active editor window.

NOTE Macintosh: Press the Shift key to change the Find Selection menu command to the Find Previous Selection menu command.

Find Reference

The **Find Reference** command searches for the definition of the selected routine name in the active Editor window, using the online help system specified in the **IDE Extras** preference panel.

If the IDE does not find a definition, a system beep sounds.

Find and Replace

The **Find and Replace** command opens the Find and Replace window. Use this window to perform find-and-replace operations within the active file.

Find Selection

The **Find Selection** command searches for the next occurrence of the selected text in the active Editor window.

G

Get Next Completion

The **Get Next Completion** command acts as a shortcut that bypasses using the Code Completion window. Instead of scrolling through the Code Completion window to select the next symbol from the one currently selected, use this command to insert that next symbol directly into the active editor window.

Get Previous Completion

The **Get Previous Completion** command acts as a shortcut that bypasses using the Code Completion window. Instead of scrolling through the Code Completion window to select the previous symbol from the one currently selected, use this command to insert that previous symbol directly into the active editor window.

Global Variables or Global Variables Window

These commands open the Global Variables window. Use this window to view global variables for an entire project or for a single file. Click a filename in the **Files** list to display the file's global variables in the **Variables** list.

Go Back

The **Go Back** command returns to the previous view in the CodeWarrior browser.

Go Forward

The **Go Forward** command moves to the next view in the CodeWarrior Browser (after you select **Go Back** command to return to previous view).

Go to Line

The **Go to Line** command opens the **Line Number** dialog box. Enter a specific line number to move the text-insertion point. If the line number specified exceeds

the number of lines in the file, the text-insertion point moves to the last line in the file.

H

Hide Breakpoints

The **Hide Breakpoints** command conceals the Breakpoints column, which appears to the left of the source code shown in editor windows.

Hide Floating Toolbar

The **Hide Floating Toolbar** command conceals the IDE's floating toolbar. After concealing the floating toolbar, the command changes to **Show Floating Toolbar**.

Hide Main Toolbar

The **Hide Main Toolbar** command conceals the IDE's main toolbar. After concealing the main toolbar, the command changes to **Show Main Toolbar**.

Hide Window Toolbar

The **Hide Window Toolbar** command conceals the toolbar in the active window. After concealing the window toolbar, the command changes to **Show Window Toolbar**.

Horizontal Center

The **Horizontal Center** command of the **Align** submenu aligns the horizontal centers of the selected components.

I

Import Components

The **Import Components** command imports components from another catalog for use with the current catalog.

Import Project

The **Import Project** command imports project files previously saved in a XML file with the **Export Project** command.

Insert Reference Template

This command inserts a routine template corresponding to the selected Mac OS Toolbox call in the active window. The IDE uses the online reference database application specified in the **Find Reference Using** pop-up to search for the routine's definition.

K-L

Kill

The **Kill** command terminates the target program and returns control to the debugger.

Left Edges

The **Left Edges** command of the **Align** submenu aligns the left edges of the selected components.

M-N

Make

The **Make** command builds the selected project by compiling and linking its modified and touched files. The results of a successful build depend on the selected project type.

Maximize Window

Windows equivalent of Expand Window.

See also: [“Expand Window” on page 534](#)

Metrowerks Website

The **Metrowerks Website** command launches a web browser and displays the Metrowerks web site.

Minimize Window

Windows equivalent of Collapse Window.

See also: [“Collapse Window” on page 529](#)

New

The **New** command opens the **New** window. Use the **New** window to create new projects, files, components, and objects.

New Class

The **New Class** command opens the New Class wizard. Use this wizard to help create new classes in a project.

New Class Browser

The **New Class Browser** command opens a Browser window. The IDE grays out this menu command if the CodeWarrior browser is not activated. This menu command is equivalent to the **Class Browser** menu command.

New Data Member

The **New Data Member** command opens the New Data Member wizard. Use this wizard to help create new data members for a class.

New Event

The **New Event** command opens the New Event window. Use this window to help create new events for a selected class in a project.

New Event Set

The **New Event Set** command opens the New Event Set window to create a new event set for a selected class in a project.

New Expression

The **New Expression** command creates a new entry in the Expressions window, prompting entry of a new expression.

New Member Function

The **New Member Function** command opens the New Member Function wizard. Use this wizard to help create new member functions for a class.

New Method

The **New Method** command opens the New Method window. Use this window to create a new method for a selected class in a project.

New Property

The **New Property** command opens the New Property window. Use this window to create a new property for a selected class in a project.

New Text File

The **New Text File** command creates a new editable text file and opens an editor window.

No Exceptions

The **No Exceptions** command of the **Java** submenu instructs the debugger to not break when exceptions occur.

O

Open

The **Open** command opens an existing project or source file.

Open Recent

The **Open Recent** menu item reveals a submenu of recently opened projects and files. Choose a file from the submenu to open that item.

If two or more files in the submenu have identical names, the submenu shows the full paths to those files in order to distinguish between them.

Open Scripts Folder

This command opens the (Scripts) folder. This command is only available if the **Use Scripts menu** option is enabled.

Open Workspace

This command opens a workspace file that you previously saved.

P-Q

Page Setup

The **Page Setup** command sets the options used for printing CodeWarrior IDE files.

Paste

The **Paste** command replaces the selected text with contents of the system clipboard into the active Editor window or text box. If no text is selected, the IDE places the clipboard contents at the text-insertion point.

The **Paste** command is unavailable if the Message window is active.

Precompile

The **Precompile** command precompiles the text file in the active Editor window into a precompiled header file.

Preferences

The **Preferences** command opens the IDE Preferences window. Use this window to change the global preferences used by the CodeWarrior IDE.

Preprocess

This command preprocesses selected source files in any language that has a preprocessor, such as C, C++, and Java.

Print

The **Print** command prints CodeWarrior IDE files, as well as Project, Message, and Errors and Warnings window contents.

Processes or Processes Window

These commands open the Processes window for those platforms that support it.

Project Inspector

Opens the Project Inspector window so that you can view information about your project. You can also use this window to manipulate file-specific information.

Quit or Quit CodeWarrior

Mac OS command equivalent of [Exit](#): See [“Exit” on page 533](#).

R

Redo

After undoing an operation, you can redo it. For example, after choosing the **Undo Typing** command to remove some text that you typed, you can choose **Redo Typing** to override the undo and restore the text.

Menu Commands

You can enable the **Use multiple undo** option in the **Editor Settings** preference panel to allow greater flexibility with regard to **Undo** and **Redo** operations.

After enabling this option, you can choose **Undo** multiple times to undo multiple actions, and you can **Redo** multiple times to redo multiple actions.

Refresh All Data

This command updates the data that appears in all windows.

Register Details Window

The **Register Details Window** command opens the Register Details window, which allows you to view descriptions of registers, bit fields, and bit values.

Registers or Register Window

These commands reveal the Registers submenu, which can be used to view general registers or FPU registers.

See also: [“Register Details Window” on page 544](#)

Remove Object Code

The **Remove Object Code** command shows the Remove Object Code dialog box. Use this dialog box to remove binary object code from the active project, or to mark the project's files for re-compilation.

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.

Remove Selected Items

The **Remove Selected Items** command removes the currently selected items from the Project window.

CAUTION You cannot undo this command.

Replace

The **Replace** command opens the Find and Replace dialog box. Use this dialog box to perform find-and-replace operations within the active file.

Replace All

The **Replace All** command finds all 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, the IDE dims this menu command.

Replace and Find Next

This command substitutes selected text with text in the **Replace** text box of the Find window, and then performs a **Find Next** operation. If no text is selected in the active Editor window and there is no text in the Find field of the Find window, the IDE grays out this menu command.

Replace and Find Previous

This command substitutes selected text with the text in the **Replace** text box of the Find window, and then performs a **Find Previous** operation. If no text is selected in the active Editor window and there is no text in the Find field of the Find window, the IDE grays out this menu command.

NOTE (Mac OS) Press the Shift key to change the Replace and Find Next menu command to the Replace and Find Previous menu command.

Replace Selection

The **Replace Selection** command substitutes the selected text in the active window with the text in the **Replace** text box of the Find window. If no text is selected in the active Editor window, the IDE grays out the menu command.

This menu command replaces one instance of a text string without having to open the Find window. Suppose that you replaced all occurrences of the variable `iCount` with `jCount`. While scrolling through your source code, you notice an instance of the variable `iCount` misspelled as `iCont`. To replace this misspelled variable with `jCount`, select `iCont` and the **Replace Selection** menu command.

Re-search for Files

The **Project > Re-search for Files** command speeds up builds and other project operations, the IDE caches the location of project files after finding them in the access paths. **Re-search for Files** forces the IDE to forget the cached locations and re-search for them in the access paths. This command is useful if you moved several files and you want the IDE to find the files in their new locations.

If the **Save project entries using relative paths** option is enabled, the IDE does not reset the relative-path information stored with each project entry, so re-searching for files finds 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 only re-searches for header files. To force the IDE to also re-search for source files, choose the **Project > Reset Project Entry Paths** menu command.

If the **Save project entries using relative paths** option is disabled, the IDE re-searches for both header files and source files.

Reset

The **Reset** command resets the program and returns control to the IDE.

Reset Floating Toolbar

The **Reset Floating Toolbar** command restores the default state of the floating toolbar. Use this command to return the floating toolbar to its original default settings.

Reset Main Toolbar

The **Reset Main Toolbar** command restores the default state of the main toolbar. Use this command to return the main toolbar to its original default settings.

Reset Project Entry Paths

The **Reset Project Entry Paths** command resets the location information stored with each project entry and forces the IDE to re-search for the project entries in the access paths. This command does nothing if the **Save project entries using relative paths** option is disabled.

Reset Window Toolbar

The **Reset Window Toolbar** command restores the default state of the toolbar in the active window. Use this command to return the toolbar to its original default settings.

Resize

The **Resize** command reveals the Resize submenu.

See also:

- [“To Largest Height” on page 553](#)
- [“To Largest Width” on page 553](#)
- [“To Smallest Height” on page 554](#)
- [“To Smallest Width” on page 555](#)

Restart

The **Restart** command terminates the current debugging session, then starts a new debugging session.

Restore Window

The **Restore Window** command restores a minimized window (a window reduced to an item in the task bar).

Resume

The **Resume** command switches from the IDE to the running application. This menu command only appears after the IDE starts a debugging session and the application being debugged is currently running.

Revert

The **Revert** command restores the last saved version of the active Editor window.

Right Edges

The **Right Edges** command of the **Align** submenu aligns the right edges of the selected components.

Run

The **Run** command compiles, links, creates a standalone application, and then runs that application. This command is unavailable if the project creates libraries, shared libraries, code resources, and other non-application binaries.

Run to Cursor

The **Run to Cursor** command sets a temporary breakpoint at the line of source code that has the text-insertion point, then runs the program.

S

Save

The **Save** command saves the contents of the active window to disk.

Save A Copy As

The **Save A Copy As** command saves the active window to a separate file. This command operates in different ways, depending on the active window.

Save All

The **Save All** command saves all currently open editor files.

NOTE Mac OS: Press the Option key to change the Save command to the Save All menu command.

Save As

The **Save As** command saves the contents of the active window to disk under a different name.

Save Default Window

This command saves the window settings, such as position and size, of the active Browser or Search Results window. The IDE applies the saved settings to subsequently opened windows.

Save Workspace

This command saves the current state of onscreen windows, recent items, and debugging. Use the dialog box that appears to name the workspace and navigate to a location in which to store the workspace file.

Save Workspace As

This command saves a copy of an existing workspace. Use this command to save the workspace under a different name.

Select All

The **Select All** command selects all text in the active window or text box. This command is usually used in conjunction with other **Edit** menu commands such as Cut, Copy, and Clear.

Send To Back

The **Send To Back** command moves the selected window behind all other windows.

Set Breakpoint

The **Set Breakpoint** command sets a breakpoint at the currently selected line. If the **Show Breakpoints** option is active, the Breakpoints column in the editor windows will display a marker next to each line with a breakpoint.

Set/Clear Breakpoint

The **Set/Clear Breakpoint** command displays the **Set/Clear Breakpoints** dialog that lets you set or clear a breakpoint at a particular address or symbol.

Set Default Project

The **Set Default Project** command sets a particular project as the default project when more than one project is open. This is the project that all commands are directed.

Set Default Target

The **Set Default Target** command allows you to specify a different build target within the current project. Choose the build target to work with from the submenu. This menu command is useful for switching between multiple build targets in a project and performing a build for each target.

Set Eventpoint

This command opens a submenu that lets you set an eventpoint at the currently selected line. If the **Show Breakpoints** option is enabled, the Breakpoints

Menu Commands

column in the editor windows shows a marker next to each line with an eventpoint. The marker represents the eventpoint type.

Set Watchpoint

The **Set Watchpoint** command sets a watchpoint for the selected variable or memory range. Watchpoint variables are identified using an underline.

Shift Left

The **Shift Left** command shifts the selected source code one tab to the left. The amount of shift is controlled by the **Tab Size** option.

Shift Right

The **Shift Right** command shifts the selected source code one tab to the right. The amount of shift is controlled by the **Tab Size** option.

Show Breakpoints

The **Show Breakpoints** command displays the Breakpoints column in editor windows. When active, the Breakpoints column appears along the left edge of all editor windows.

Show Floating Toolbar

The **Show Floating Toolbar** command displays the IDE's floating toolbar. After displaying the floating toolbar, the command changes to **Hide Floating Toolbar**.

Show Main Toolbar

The **Show Main Toolbar** command displays the IDE's main toolbar. After displaying the main toolbar, the command changes to **Hide Main Toolbar**.

Show Types

The **Show Types** command displays the data types of all local and global variables that appear in the active variable pane or variable window.

Show Window Toolbar

The **Show Window Toolbar** command displays the toolbar in the active window. After displaying the window toolbar, the command changes to **Hide Window Toolbar**.

Stack Editor Windows

The **Stack Editor Windows** command arranges open editor windows one on top of another, with their window titles visible.

Step Into

The **Step Into** command executes a single statement, stepping into function calls.

Step Out

The **Step Out** command executes the remainder of the current function, then exits to that function's caller.

Step Over

The **Step Over** command executes a single statement, stepping over function calls.

Stop

This command temporarily suspends execution of the target program and returns control to the debugger.

Stop Build

The **Stop Build** command halts the build currently in progress.

Switch to Monitor

This command transfers control from the CodeWarrior debugger to an external third-party debugger.

Symbolics or Symbolics Window

These commands open the Symbolics window. Use this window to examine the executable files in a project.

Synchronize Modification Dates

The **Synchronize Modification Dates** command updates the modification dates stored in the project file. The IDE checks the modification date of each file in the project and marks (for recompiling) those files modified since the last successful compile process.

T-U

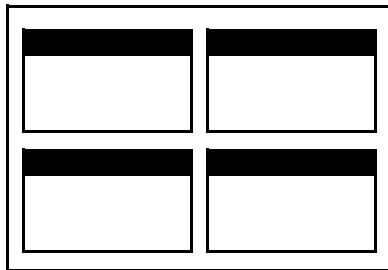
Target Settings

The **Target Settings** command displays the Target Settings window. This window contains settings panels used by the active build target. The name of the menu command changes, based on the name of the current build target. For example, if the name of the current build target is `ReleaseTarget`, the name of the menu command changes to **ReleaseTarget Settings**.

Tile Editor Windows

The **Tile Editor Windows** command arranges and resizes all open editor windows so that none overlap on the monitor.

Figure 31.1 Tile Editor windows—example



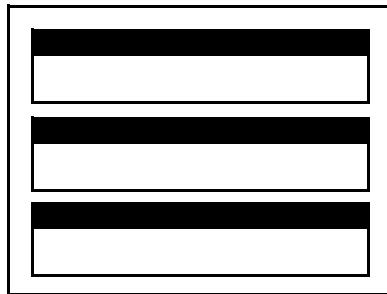
Tile Editor Windows Vertically

The **Tile Editor Windows Vertically** command resizes all open editor windows to be vertically long, and arranged horizontally across the monitor so that all are viewable.

Tile Horizontally

This command arranges open editor windows horizontally so that none overlap.

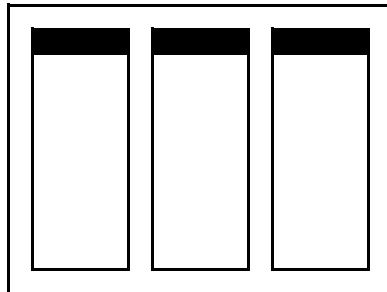
Figure 31.2 Tile horizontally—example



Tile Vertically

This command resizes open editor windows vertically and arranges them so that none overlap.

Figure 31.3 Tile vertically—example



To Grid

The **To Grid** command of the **Align** submenu aligns selected components to a grid in the layout. You can display or hide the on screen grid.

To Largest Height

The **To Largest Height** command of the **Resize** submenu resizes the selected components to match the height of the component with the largest height.

To Largest Width

The **To Largest Width** command of the **Resize** submenu resizes the selected components to match the width of the component with the largest width.

Toolbars

The **Toolbars** command reveals the Toolbars submenu.

See also:

- [“Show Window Toolbar” on page 550](#)
- [“Hide Window Toolbar” on page 538](#)
- [“Reset Window Toolbar” on page 546](#)
- [“Clear Window Toolbar” on page 528](#)
- [“Show Main Toolbar” on page 550](#)
- [“Hide Main Toolbar” on page 538](#)
- [“Reset Main Toolbar” on page 546](#)
- [“Clear Main Toolbar” on page 528](#)
- [“Hide Floating Toolbar” on page 538](#)
- [“Show Floating Toolbar” on page 550](#)
- [“Reset Floating Toolbar” on page 546](#)
- [“Clear Floating Toolbar” on page 528](#)

ToolServer Worksheet

The **ToolServer Worksheet** command opens the ToolServer Worksheet window for use with the Apple® ToolServer™ application program.

The IDE can disable this command for these reasons:

- You did not install ToolServer on your computer.
- You installed ToolServer on your computer, but you did not start it.

Top Edges

The **Top Edges** command of the **Align** submenu aligns the top edges of the selected components.

To Smallest Height

The **To Smallest Height** command of the **Resize** submenu resizes the selected components to match the height of the component with the smallest height.

To Smallest Width

The **To Smallest Width** command of the **Resize** submenu resizes selected components to match the width of the component with the smallest width.

Unanchor Floating Toolbar

The **Unanchor Floating Toolbar** command detaches the floating toolbar from beneath the menu bar.

Unapply Difference

The **Unapply Difference** command reverses the action of the **Apply Difference** command in a file-comparison window.

Uncought Exceptions Only

The **Uncought Exceptions Only** command of the **Java** submenu instructs the debugger to break only on unhandled exceptions.

Undo

The **Undo** command reverses the last action. The name of this menu command changes based upon the editor settings as well as the most recent action. For example, after typing text in an open Editor window, the **Undo** command changes its name to **Undo Typing**. Choose the **Undo Typing** command to remove the typed text.

By default, only one undo or redo action is allowed. If the **Use multiple undo** option is enabled, undo and redo can act upon multiple actions.

Ungroup

The **Ungroup** command separates a selected group so that you can move each component independently.

V-Z

Version Control Settings

The **Version Control Settings** command opens the VCS Settings window.

Vertical Center

The **Vertical Center** command of the **Align** submenu aligns the vertical centers of the selected components.

View Array

The **View Array** command creates a separate window to display a selected array.

View As

The **View As** command displays a selected variable in a specified data type.

View As Binary

The **View As Binary** command displays the selected variable as a binary value.

View As Character

The **View As Character** command displays the selected variable as a character value.

View As C String

The **View As C String** command displays the selected variable as a C character string.

View As Default

The **View As Default** command displays the selected variable in its default format, based on the variable's type.

View As Enumeration

The **View As Enumeration** command displays the selected variable as an enumeration.

View As Fixed

The **View As Fixed** command displays the selected variable as a fixed-type numerical value.

View As Floating Point

The **View As Floating Point** command displays the selected variable as a floating-point value.

View As Fract

This command displays the selected variable as a fractional data type.

NOTE The fractional data type is specific to the Mac OS.

View As Hexadecimal

The **View As Hexadecimal** command displays the selected variable as a hexadecimal value.

View As Pascal String

The **View As Pascal String** command displays the selected variable as a Pascal character string.

View As Signed Decimal

This command displays the selected variable as a signed decimal value.

View As Unicode String

The **View As Unicode String** command displays the selected variable as a Unicode character string.

View As Unsigned Decimal

The **View As Unsigned Decimal** command displays the selected variable as an unsigned decimal value.

View Disassembly

This command changes the data view to show language disassembly.

View Memory

The **View Memory** command displays the contents of memory as a hexadecimal/ASCII character dump.

View Memory As

The **View Memory As** command displays the memory that a selected variable occupies or the memory to which a selected register points.

View Mixed

This command changes the data view to show source code intermixed with assembly code.

View Raw Data

This command changes the data view to show raw data (instead of formatting that data as source code, disassembly, or another format).

View Source

This command changes the data view to show source code.

View Variable

The **View Variable** command creates a separate window to display a selected variable.

Zoom Window

The **Zoom Window** command expands the active window to its previously set size. Choose **Zoom Window** a second time to return the window to its original size.

Register Details Window XML Specification

The CodeWarrior Register Details window provides online documentation about hardware registers up to 32 bits in length. This documentation helps programmers better understand the register and its constituent parts. Furthermore, the Register Details window can dynamically update its information to reflect changes in register state or changes in bitfield values.

The Register Details window documentation can include bitfield descriptions, descriptions that change based on certain conditions, and explanations of bitfield values. The same window handles both system registers and memory-mapped registers.

CodeWarrior parses XML files with specific tags to display the appropriate information in the window. This chapter describes the XML format understood by the Register Details window.

- [Register Details Window XML Specification](#)
- [Accessing the XML Files from CodeWarrior](#)
- [A Sample XML File](#)
- [References](#)

Register Details Window XML Specification

XML is composed of elements, which are similar to tags in Hypertext Markup Language (HTML). Each element contains attributes that give detailed information about the element structure. Some elements are required and some are optional.

As is the case with HTML, XML elements include tags that mark the beginning and end of the element, such as <ELEM> and </ ELEM>. An alternate construct allows <ELEM> and </> to mark the beginning and end of the element, respectively.

Register Details Window XML Specification

Register Details Window XML Specification

Attributes can include numeric values, which you may enter in a variety of formats ([Table A.1](#)). The table shows sample values for each format. The table also describes additional requirements for valid values that the Register Details window can understand. For example, valid character values must be enclosed in single-quote marks, like this: ' c '

Table A.1 Numeric attribute value formats

Value Format	Sample Values	Requirements
Decimal	123 and -334	
Hexadecimal	0x12 and 0X12	Preceded by 0x or 0X
Octal	012	Preceded by 0
Binary	0b11001 and 0B11	Preceded by 0b or 0B
Character	' c ' and ' XYZ '	Enclosed in ''

Every XML file created for use with the Register Details window must conform to the specification shown in [Listing A.1](#). The specification defines the following types of elements:

- [REGISTER](#)
- [BITFIELD](#)
- [BFVALUE](#)

Descriptions of these elements follow the specification.

Listing A.1 Register Details Window XML Specification

```
<!DOCTYPE REGISTER [>
  <!ELEMENT REGISTER      (BITFIELD+)>
  <!ATTLIST REGISTER
    NAME          CDATA      #REQUIRED
    BITRANGE     CDATA      #REQUIRED
    RESETVALUE   CDATA      #IMPLIED
    ADDRESS      CDATA      #IMPLIED
    DESCRIPTION   CDATA      #IMPLIED>

  <!ELEMENT BITFIELD      (BFVALUE*)>
  <!ATTLIST BITFIELD
    NAME          CDATA      #REQUIRED
    BITRANGE     CDATA      #REQUIRED
```

```
FORMAT  
(binary|b|hex|h|decimal|d|unsigned|u|character|c  
          |value|v) "binary"  
ACCESS      (read|r|write|w|readwrite|rw|reserved)  
          "readwrite"  
CONDITION   CDATA      #IMPLIED  
DESCRIPTION  CDATA      #IMPLIED>  
  
<!ELEMENT BFVALUE      EMPTY>  
<!ATTLIST BFVALUE  
      VALUE      CDATA      #REQUIRED  
      DESCRIPTION CDATA      #REQUIRED>  
>
```

REGISTER

The REGISTER element describes the name, bitrange, reset value, address, and general description of the register. The italicized portions of the format are placeholders that indicate where you must supply additional information. The remainder of this section describes each REGISTER attribute.

Element Format

```
<REGISTER  
      NAME      = "RegisterName"  
      BITRANGE  = "MSB:LSB/BitNumber"  
      RESETVALUE = "0x11223344"  
      ADDRESS   = "0x10000+4"  
      DESCRIPTION = "RegisterDescription">  
</REGISTER>
```

NAME

This attribute specifies the register name. If the element does not include an ADDRESS attribute, CodeWarrior matches XML files based on the NAME attribute, and the register is assumed to be registered with the IDE under the NAME attribute. If the element includes an ADDRESS attribute, the register is assumed to be memory mapped, and the debugger evaluates the ADDRESS attribute to find the address of the register, using the information in the NAME attribute for display purposes only.

This attribute is a required part of the REGISTER element.

Register Details Window XML Specification

Register Details Window XML Specification

You must enter a NAME attribute in the form

String

where *String* represents the name of the register.

BITRANGE

This attribute defines the register bitrange. If the bitrange is a single bit, enter the bit number. If the bitrange is longer than a single bit, enter the range separated by a colon. For example, enter 0 : 6 to specify a range from 0 to 6. The bit ordering can be in any order, such as 0 : 31 or 31 : 0. This flexibility lets you accommodate varying product documentation, where bit ordering is sometimes reversed. However, after specifying a particular bit order, each subsequent bitrange attribute in the BITFIELD element must follow the same order.

This attribute is a required part of the REGISTER element.

You must enter a BITRANGE attribute in the form

MSB:LSB

or

BitNumber

where *MSB:LSB* refers to the range between the most-significant bit and the least-significant bit, and *BitNumber* refers to the single bit that represents the bitrange.

RESETVALUE

This attribute allows you to specify the register reset value.

This attribute is an optional part of the REGISTER element.

You must enter a RESETVALUE attribute in the form

Value

where *Value* represents the reset value of the register. Refer to [Table A.1 on page 560](#) for more information about valid values.

ADDRESS

This attribute lets you specify that the register is a memory-mapped register. The CodeWarrior expression evaluator determines the value of the attribute by evaluating the following items:

- mathematical operations

- boolean operations such as AND, OR, NOT, and XOR
- the values of registers whose names begin with a dollar sign (\$)
- variables included in the generated symbolics file for the project

This attribute is an optional part of the REGISTER element.

You must enter a RESETVALUE attribute in the form

String

where *String* represents the name of the register.

DESCRIPTION

This attribute lets you provide a description for the register. This description can be of arbitrary length. The Description field in the Register Details window includes scrollbars, allowing you to view the entire description within the window.

This attribute is an optional part of the REGISTER element.

You must enter a DESCRIPTION attribute in the form

String

where *String* represents the description of the register.

BITFIELD

The BITFIELD element describes the name, bitrange, format, access, condition, and general description of individual bitfields within the register. The italicized portions of the format are placeholders that indicate where you must supply additional information. The remainder of this section describes each BITFIELD attribute in detail.

NOTE If you choose not to describe the register bitfields, or if the register does not have bitfields that require individual descriptions, you can leave the BITFIELD element empty in the XML file.

Element Format

```
<BITFIELD  
    NAME      =      "BitfieldName"
```

Register Details Window XML Specification

Register Details Window XML Specification

```
BITRANGE      =  "(MSB:LSB/Number)"
FORMAT        =  "BitfieldFormat"
ACCESS        =  "AccessFormat"
CONDITION     =  "Expression"
DESCRIPTION   =  "BitfieldDescription">
</BITFIELD>
```

NAME

This attribute specifies the bitfield name.

This attribute is a required part of the BITFIELD element.

You must enter a NAME attribute in the form

String

where *String* represents the name of the bitfield.

BITRANGE

This attribute defines the bitfield range. If the bitrange is a single bit, enter the bit number. If the bitrange is longer than a single bit, enter the range separated by a colon. For example, enter 0 : 6 to specify a range from 0 to 6. The bit ordering must follow the order you specified in the [BITRANGE](#) attribute of the REGISTER element. See [“BITRANGE” on page 562](#) for more information.

This attribute is a required part of the BITFIELD element.

You must enter a BITRANGE attribute in the form

MSB:LSB

or

BitNumber

where *MSB:LSB* refers to the range between the most-significant bit and the least-significant bit, and *BitNumber* refers to the single bit that represents the bitrange.

FORMAT

This attribute determines the default format of the register values displayed in the Register Details window. You can enter one of the following formats for this attribute:

- binary or b
- hex or h

- decimal or d
- unsigned or u
- character or c
- value or v

If you omit this attribute, CodeWarrior assumes a default binary format. If you choose the value format, the bitfield appears as the text description value, or else appears in binary format when no description is provided for the specified bitfield value.

This attribute is an optional part of the BITFIELD element.

You must enter a FORMAT attribute in the form

FullName

or

abbr

where *FullName* represents the full name of the format and *abbr* represents the abbreviation of that format.

ACCESS

This attribute lets you specify the bitfield access permissions. You can enter one of the following permissions for this attribute:

- read or r
- write or w
- readwrite or rw
- reserved or " "

If you omit this attribute, CodeWarrior assumes a default readwrite access permission.

This attribute is an optional part of the BITFIELD element.

You must enter an ACCESS attribute in the form

FullName

or

abbr

where *FullName* represents the full name of the format and *abbr* represents the abbreviation of that format.

Register Details Window XML Specification

Register Details Window XML Specification

CONDITION

This attribute lets you provide a particular description for a bitfield, depending on whether a condition you specify is met. You specify a conditional *Expression* for the bitfield using the CONDITION attribute. CodeWarrior evaluates the expression, and if the expression is true, assumes that the bitfield attribute is valid.

This capability is useful for providing different descriptions for the same bitfield, based on the value of the *Expression*. For example, you can create two bitfield entries for the same register bit. Each bitfield entry has a distinct CONDITION attribute, allowing CodeWarrior to choose the appropriate BITFIELD element to display in the Register Details window.

This attribute is an optional part of the BITFIELD element.

You must enter a CONDITION attribute in the form

Expression

where *Expression* represents the condition that CodeWarrior must evaluate ([Table A.2](#)). The expression can refer to other registers by adding a dollar sign (\$) to the beginning of each register name. The expression can also refer to the current register value by entering two dollar signs (\$\$) in the *Expression*. The Register Details window replaces these dollar signs with the current register value. The CodeWarrior expression evaluator also accepts local and global variables in the *Expression*.

Table A.2 Sample expressions for the CONDITION attribute

Expression	Explanation
CONDITION = "\$\$&0x80"	The current register value ANDed with 0x80
CONDITION = "!(\$\$&0x80)"	The inversion of the current register value ANDed with 0x80
CONDITION = "\$MSR&0x8000"	Another register value (the MSR register) ANDed with 0x8000
CONDITION = "foo&0x10 >= 0"	An expression using a variable named foo

DESCRIPTION

This attribute lets you provide a description for the bitfield. This description can be of arbitrary length. The Description field in the Register Details window includes scrollbars, allowing you to view the entire description within the window.

This attribute is an optional part of the BITFIELD element.

You must enter a DESCRIPTION attribute in the form

String

where *String* represents the bitfield description.

BFVALUE

The BFVALUE element lets you explain the individual values of a bitfield described by the [DESCRIPTION](#) attribute in the BITFIELD element. The italicized portions of the format are placeholders that indicate where you must supply additional information. The remainder of this section describes each BFVALUE attribute.

NOTE If you choose not to describe individual bitfield values, or if the bitfields do not require individual descriptions, you can leave the BFVALUE element empty in the XML file.

Element Format

```
<BFVALUE
    VALUE      =  "BitfieldValue"
    DESCRIPTION =  "BitfieldValueDescription">
</BFVALUE>
```

VALUE

This attribute specifies the value of the bitfield to be described by the [DESCRIPTION](#) attribute.

This attribute is a required part of the BFVALUE element.

You must enter a VALUE attribute in one of the following forms:

- decimal

Register Details Window XML Specification

Accessing the XML Files from CodeWarrior

- unsigned decimal
- hexadecimal (the value must begin with 0x or 0X)
- octal (the value must begin with 0)
- binary (the value must begin with 0b or 0B)
- character (enclosed in single quote marks, like this: ' c ')

DESCRIPTION

This attribute lets you provide a description of the bitfield value specified by the VALUE attribute.

This attribute is a required part of the BFVALUE element.

You must enter a DESCRIPTION attribute in the form

String

where *String* represents the bitfield value description.

Accessing the XML Files from CodeWarrior

The CodeWarrior Register Details window searches a specific folder for relevant files. You must place XML Register Details window files within the Registers folder in your CodeWarrior installation.

Windows

If necessary, create the Registers folder at the following location:

CodeWarrior\Bin\Plugins\Support\Registers

Mac OS

If necessary, create the Registers folder at the following location:

Metrowerks CodeWarrior:CodeWarrior Plugins:Support:Registers

IDE plug-ins can provide access paths to the IDE to better control the placement of XML files within the Registers folder. For example, you can create sub-folders within the Registers folder for specific processors or processor variations and use the plug-in access paths to search those sub-folders. These access paths are relative to the Registers folder.

The IDE looks up system registers by name. Memory-mapped registers can have any name, since that name is used for display purposes only.

A Sample XML File

This section provides examples of creating XML file for use with the Register Details window.

- [Creating the New XML File](#)
- [Adding Multiple BITFIELD Attributes](#)
- [Adding BFVALUE Attributes](#)
- [Completing the New XML File](#)

Creating the New XML File

When you create a new file, you usually follow these high-level steps:

1. Locate a base XML file.

Instead of creating a completely new XML file, you can adapt an existing XML file for use with the register you wish to document. For example, you can locate a simple, generic XML file and modify it to describe more sophisticated registers.

[Listing A.2](#) shows a sample base XML file that you can easily adapt to explain complex registers. Note that this generic base file lacks multiple bitfield attributes, conditional expressions, or individual bitfield value attributes.

Listing A.2 A sample base XML file

```
<REGISTER NAME="BAR"

BITRANGE="0:31"
DESCRIPTION="Breakpoint Address Register">

<BITFIELD BITRANGE="0:31"
DESCRIPTION="The address of the load/store cycle that generates the
breakpoint.">
</BITFIELD>

</REGISTER>
```

Register Details Window XML Specification

A Sample XML File

2. Save the base XML file under a new name.

Use CodeWarrior to save a copy of the base XML file under a new name, and work with this newly named file for the remainder of the process. This step prevents you from accidentally modifying the original XML file.

3. Modify the base XML file to suit your needs.

After opening your copy of the base XML file, you can adapt the file to accurately document complicated register properties. For example, you can add multiple BITFIELD attributes, BFVALUE attributes, and conditional expressions to the base XML file. The resulting file accurately and thoroughly describes the register. Such a file appears in [Listing A.5 on page 574](#).

Adding Multiple BITFIELD Attributes

Multiple BITFIELD attributes divide the register into smaller ranges of bits, or “bitfields.” Such bitfields can have various meanings depending on the register. For example, one bitfield of a register could refer to condition flag information, while another bitfield in the same register could refer to current state information.

Suppose you wish to document a “Memory Controller Base 2” register that contains twelve bitfields. Each bitfield has its own name and description.

Using the sample base XML file of [Listing A.2](#), you could begin adapting the file to your needs by adding eleven additional BITFIELD elements. Following the element formats described in the [“Register Details Window XML Specification” on page 559](#), your first revision of the base file might appear as shown in [Listing A.3 on page 570](#).

Listing A.3 First revision of the base XML file

```
<REGISTER NAME="BR2"

BITRANGE="0:31"
DESCRIPTION="Memory Controller Base Register 2 . ">

<BITFIELD BITRANGE="0:16"
NAME="BA"
DESCRIPTION="Place the BA bitfield description here.">
</BITFIELD>

<BITFIELD BITRANGE="17:19"
NAME="AT"
DESCRIPTION="Place the AT bitfield description here.">
```

Register Details Window XML Specification

A Sample XML File

```
</BITFIELD>

<BITFIELD BITRANGE="20:21"
  NAME="PS"
  DESCRIPTION="Place the PS bitfield description here.">
</BITFIELD>

<BITFIELD BITRANGE="22"
  NAME="--"
  DESCRIPTION="Make a note that this bitfield is reserved.">
</BITFIELD>

<BITFIELD BITRANGE="23"
  NAME="WP"
  DESCRIPTION="Place the AT bitfield description here.">
</BITFIELD>

<BITFIELD BITRANGE="24:25"
  NAME="--"
  DESCRIPTION="Make a note that this bitfield is reserved.">
</BITFIELD>

<BITFIELD BITRANGE="26"
  NAME="WEBS"
  DESCRIPTION="Place the WEBS bitfield description here.">
</BITFIELD>

<BITFIELD BITRANGE="27"
  NAME="TBDIP"
  DESCRIPTION="Place the TBDIP bitfield description here.">
</BITFIELD>

<BITFIELD BITRANGE="28"
  NAME="LBDIP"
  DESCRIPTION="Place the LBDIP bitfield description here.">
</BITFIELD>

<BITFIELD BITRANGE="29"
  NAME="SETA"
  DESCRIPTION="Place the SETA bitfield description here.">
</BITFIELD>

<BITFIELD BITRANGE="30"
  NAME="BI"
  DESCRIPTION="Place the BI bitfield description here.">
</BITFIELD>

<  <BITFIELD BITRANGE="31"
```

Register Details Window XML Specification

A Sample XML File

```
NAME="V"
DESCRIPTION="Place the V bitfield description here.">
</BITFIELD>

/REGISTER>
```

Adding BFVALUE Attributes

The values in an individual bitfield can describe different things about a register. For example, a bitfield value of 00 might indicate that a certain process is used, while a bitfield value of 01 might indicate that the same process is not used.

To cater your descriptions to accurately reflect such changes in behavior, you can use multiple BFVALUE attributes within a BITFIELD element. Each BFVALUE provides a specific description based on the bitfield value. CodeWarrior determines the applicable description to display in the Register Details window.

Using the example discussed in previous sections, assume that six of the bitfields in the “Memory Controller Base Register 2” could benefit from the use of BFVALUE attributes.

Following the element formats described in the [“Register Details Window XML Specification” on page 559](#), your second revision of the base file might resemble [Listing A.3 on page 570](#).

Listing A.4 Second revision of the XML base file

```
<REGISTER NAME="BR2"

    BITRANGE="0:31"
    DESCRIPTION="Memory Controller Base Register 2.">

    <BITFIELD BITRANGE="0:16"
        NAME="BA"
        DESCRIPTION="Place the BA bitfield description here.">
    </BITFIELD>

    <BITFIELD BITRANGE="17:19"
        NAME="AT"
        DESCRIPTION="Place the AT bitfield description here.">
    </BITFIELD>

    <BITFIELD BITRANGE="20:21"
        NAME="PS"
        DESCRIPTION="Place the PS bitfield description here.">
```

Register Details Window XML Specification

A Sample XML File

```
<BFVALUE VALUE="00" DESCRIPTION="Place the description of the  
bitfield value of 00 here." />  
    <BFVALUE VALUE="01" DESCRIPTION="Place the description of the  
bitfield value of 01 here." />  
        <BFVALUE VALUE="10" DESCRIPTION="Place the description of the  
bitfield value of 10 here." />  
            <BFVALUE VALUE="11" DESCRIPTION="Place the description of the  
bitfield value of 11 here." />  
    </BITFIELD>  
  
<BITFIELD BITRANGE="22"  
NAME=" - "  
DESCRIPTION="Make a note that this bitfield is reserved.">  
</BITFIELD>  
  
<BITFIELD BITRANGE="23"  
NAME="WP"  
DESCRIPTION="Place the AT bitfield description here.">  
    <BFVALUE VALUE="0" DESCRIPTION="Place the description of the  
bitfield value of 0 here." />  
        <BFVALUE VALUE="1" DESCRIPTION="Place the description of the  
bitfield value of 1 here." />  
    </BITFIELD>  
  
<BITFIELD BITRANGE="24:25"  
NAME=" - "  
DESCRIPTION="Make a note that this bitfield is reserved.">  
</BITFIELD>  
  
<BITFIELD BITRANGE="26"  
NAME="WEBS"  
DESCRIPTION="Place the WEBS bitfield description here.">  
    <BFVALUE VALUE="0" DESCRIPTION="Place the description of the  
bitfield value of 0 here." />  
        <BFVALUE VALUE="1" DESCRIPTION="Place the description of the  
bitfield value of 1 here." />  
    </BITFIELD>  
  
<BITFIELD BITRANGE="27"  
NAME="TBDIP"  
DESCRIPTION="Place the TBDIP bitfield description here.">  
</BITFIELD>  
  
<BITFIELD BITRANGE="28"  
NAME="LBDIP"  
DESCRIPTION="Place the LBDIP bitfield description here.">  
    <BFVALUE VALUE="0" DESCRIPTION="Place the description of the  
bitfield value of 0 here." />
```

Register Details Window XML Specification

A Sample XML File

```
<BFVALUE VALUE="1" DESCRIPTION="Place the description of the
bitfield value of 1 here." />
</BITFIELD>

<BITFIELD BITRANGE="29"
NAME="SETA"
DESCRIPTION="Place the SETA bitfield description here.">
    <BFVALUE VALUE="0" DESCRIPTION="Place the description of the
bitfield value of 0 here." />
    <BFVALUE VALUE="1" DESCRIPTION="Place the description of the
bitfield value of 1 here." />
</BITFIELD>

<BITFIELD BITRANGE="30"
NAME="BI"
DESCRIPTION="Place the BI bitfield description here.">
    <BFVALUE VALUE="0" DESCRIPTION="Place the description of the
bitfield value of 0 here." />
    <BFVALUE VALUE="1" DESCRIPTION="Place the description of the
bitfield value of 1 here." />
</BITFIELD>

<  <BITFIELD BITRANGE="31"
NAME="V"
DESCRIPTION="Place the V bitfield description here.">
</BITFIELD>

/REGISTER>
```

Completing the New XML File

Adding multiple BITFIELD attributes and the BFVALUE attributes further refines the XML file register description. The final revision of an XML file involves completing the descriptions for each attribute.

Refining the example discussed in previous sections, [Listing A.5](#) shows the final XML file. To use this new file with the Register Details window, place the file in the Registers folder. For more information about this folder, refer to [“Accessing the XML Files from CodeWarrior” on page 568](#).

Listing A.5 A sophisticated XML file

```
<REGISTER NAME="BR2"
    BITRANGE="0:31"
```

Register Details Window XML Specification

A Sample XML File

```
DESCRIPTION="Memory Controller Base Register 2.">

<BITFIELD
  BITRANGE="0:16"
  NAME="BA"
  DESCRIPTION="Base Address. These bits are compared to the
corresponding unmasked address signals among ADDR[0:16] to determine
if a memory bank controlled by the memory controller is being accessed
by an internal bus master. (The address types are also compared.) These
bits are used in conjunction with the A<[0:16] bits in the OR">
</BITFIELD>

<BITFIELD
  BITRANGE="17:19"
  NAME="AT"
  DESCRIPTION="Address type. This field can be used to require
accesses of the memory bank to be limited to a certain address space
type. These bits are used in conjunction with the ATM bits in the OR.">
</BITFIELD>

<BITFIELD
  BITRANGE="20:21"
  NAME="PS"
  DESCRIPTION="Port size.">
  <BFVALUE VALUE="00" DESCRIPTION="32-bit port" />
  <BFVALUE VALUE="01" DESCRIPTION="8-bit port" />
  <BFVALUE VALUE="10" DESCRIPTION="16-bit port" />
  <BFVALUE VALUE="11" DESCRIPTION="Reserved" />
</BITFIELD>

<BITFIELD
  BITRANGE="22"
  NAME=" - "
  DESCRIPTION="Reserved.">
</BITFIELD>

<BITFIELD
  BITRANGE="23"
  NAME="WP"
  DESCRIPTION="Write protect. An attempt to write to the range of
addresses specified in a base address register that has this bit set
can cause the TEA signal to be asserted by the bus-monitor logic (if
enabled) causing termination of this cycle.">
  <BFVALUE VALUE="0" DESCRIPTION="Both read and write accesses
are allowed" />
  <BFVALUE VALUE="1" DESCRIPTION="Only read accesses are
allowed. The CSx signal and TA are not asserted by the memory

```

Register Details Window XML Specification

A Sample XML File

```
controller on write cycles to this memory bank. WPER is set in the
MSTAT register if a write to this memory bank is attempted." />
    </BITFIELD>

    <BITFIELD
        BITRANGE="24:25"
        NAME="-"
        DESCRIPTION="Reserved">
    </BITFIELD>

    <BITFIELD
        BITRANGE="26"
        NAME="WEBS"
        DESCRIPTION="Write-enable/byte select. This bit controls the
functionality of the WE/BE pads.">
        <BFVALUE VALUE="0" DESCRIPTION="The WE/BE pads operate as WE"
/>
        <BFVALUE VALUE="1" DESCRIPTION="The WE/BE pads operate as BE"
/>
    </BITFIELD>

    <BITFIELD
        BITRANGE="27"
        NAME="TBDIP"
        DESCRIPTION="Toggle-burst data in progress. TBDIP determines how
long the BDIP strobe will be asserted for each data beat in the burst
of cycles.">
    </BITFIELD>

    <BITFIELD
        BITRANGE="28"
        NAME="LBDIP"
        DESCRIPTION="Late-burst-data-in-progress (LBDIP). This bit
determines the timing of the first assertion of the BDIP pin in burst
cycles. Note: It is not allowed to set both LBDIP and TBDIP bits in a
region's base registers; the behavior of the design in such cases is
unpredictable.">
        <BFVALUE VALUE="0" DESCRIPTION="Normal timing for BDIP
assertion (asserts one clock after negation of TS" />
        <BFVALUE VALUE="1" DESCRIPTION="Late timing for BDIP assertion
(asserts after the programmed number of wait states" />
    </BITFIELD>

    <BITFIELD
        BITRANGE="29"
        NAME="SETA"
        DESCRIPTION="External transfer acknowledge.">
```

```
<BFVALUE VALUE="0" DESCRIPTION="TA generated internally by  
memory controller" />  
<BFVALUE VALUE="1" DESCRIPTION="TA generated by external  
logic. Note that programming the timing of CS/WE/OE strobes may have no  
meaning when this bit is set." />  
</BITFIELD>  
  
<BITFIELD  
BITRANGE="30"  
NAME="BI"  
DESCRIPTION="Burst inhibit. Note: Following a system reset, the BI  
bit is set in OR0.">  
    <BFVALUE VALUE="0" DESCRIPTION="Memory controller drives BI  
negated (high). The bank supports burst accesses." />  
    <BFVALUE VALUE="1" DESCRIPTION="Memory controller drives BI  
asserted (low). The bank does not support burst accesses." />  
</BITFIELD>  
  
<BITFIELD  
BITRANGE="31"  
NAME="V"  
DESCRIPTION="Valid bit. When set, this bit indicates that the  
contents of the base-register and option-register pair are valid. The  
CSignal does not assert until the V-bit is set. Note that an access to  
a region that has no V-bit set may cause a bus monitor timeout. Note  
also that following a system reset, the V-bit in BR0 reflects the value  
of ID3 in the reset configuration word.">  
</BITFIELD>  
  
</REGISTER>
```

References

For more information about XML, consult the following references:

- *XML: A Primer*, by Simon St. Laurent, published by IDG Books Worldwide, Inc.
- *Presenting XML*, by Richard Light, published by Macmillan Computer Publishing.
- *The XML Companion*, by Neil Bradley, published by Addison-Wesley.

Register Details Window XML Specification

References

Index

Symbols

#include files, caching 466
#pragma directives, profiler 343
%file command-line string 467
%line command-line string 468
(Scripts) folder 522, 542
. * [_] Data 396
.mcp 37
\(.*\)\ 396
^var 254
_copy_vectors() 346
__PROFILE_ENTRY 334
__PROFILE_EXIT 334
__throw() 525

A

abnormal termination 333
about
breakpoints 221
console applications 83
dockable windows 67
eventpoints 221
Files page in Project window 47
markers 116
projects 31
special breakpoints 221
watchpoints 221
workspaces 79
About Metrowerks CodeWarrior menu
command 523
Absolute Path option
in Source Trees preference panel 484
in Type list box 484
abstract, icon for 176
access breakpoint - refer to watchpoints 244
access breakpoint enabled 244
Access Filter display 178
Access Paths settings panel 394, 427
columns
Framework 430
Recursive Search 430

Search Status 429
options
Add 429
Add Default 429
Always Search User Paths 428
Change 429
Host Flags 429
Interpret DOS and Unix Paths 429
Remove 429
Require Framework Style Includes 429
System Paths 483
User Paths 488
Access Target button 301
Action option 365
actions for debugging 208
Activate Browser Coloring option 449
in Text Colors panel 464
Activate Browser option
and relation to Symbolics window 275
in Build Extras panel 535
Activate Syntax Coloring option 449, 455
in Text Colors panel 464, 467, 481
activating automatic code completion 103
Active icon 225
Add button 429
Add Default button 449
Add Files button 139
Add Files menu command 523
Add Window menu command 523
adding
gray background behind IDE. *See Use*
Multiple Document Interface, turning on.
remote connections 420
source trees 398
Address checkbox 304
Address Line fault 306
Address text box 271
advanced topics
for projects 41
Align submenu 524, 525, 538, 539, 547, 553, 554,
556

Horizontal Center command 538
Left Edges command 539
Vertical Center command 547, 553, 554, 556
All Exceptions command 524
All Info option, in Plugin Diagnostics 469
All Sectors checkbox 295
All Sectors list 295
All Text option button 125, 128, 131
alphabetical sorting of Functions list pop-up 115
Always Search User Paths option 450
Analyzer Can Cause Target Breakpoint checkbox 312
Analyzer Configuration File text box 311
Analyzer Connections target settings panel 309
Analyzer Slot text box 312
Analyzer Type list box 311
Ancestor pop-up 180
Anchor Floating Toolbar command 524
Appears in Menus 365, 366
Appears in Menus checkbox 144
Apple Help Viewer 463
Apple menu 507
Application field 450
applications
 for the console, about 83
 for the console, creating 84
Apply Address Offset checkbox 293
Apply button 153
Apply Difference command 154, 524, 555
Arguments field 450
Arithmetic Optimizations 439
Arm command 313
Array window 264
 opening 266
arrays, setting default viewing size for
 unbounded 457
arrows
 current statement 207
assigning
 Quote Key prefix 384
Attempt To Use Dynamic Type of C++, Object
 Pascal And SOM Objects option 450
Auto Indent option 450
Auto Repeat 366
Auto Target Libraries option 450
Auto, of Text View list box 274
auto-complete code. *See* code completion.
Automatic Invocation option 451
Automatically Launch Applications When SYM
 File Opened option 451
Auto-target Libraries option 450

B

Background option 452
background, desktop
 removing from behind IDE. *See* Use Multiple Document Interface, turning off.
Balance Flash Delay option 452
Editor Settings panel 452
Balance menu command 525
Balance While Typing option 452
balancing punctuation 100, 101
 toggling 101
Balloon Help 376, 459, 522
Base Classes field 192
Begin Scope Loop button 303
Begin Test button 305
bestTimeBase 331, 345
BFVALUE 567
Bit Value Modifier list box 272
Bit Value text box 272
BITFIELD 563
Bitfield Description option
 of Text View pop-up menu 274
Bitfield Name list box 272
Blank Check button 295
Bottom Edges command 525
boxes
 Destination 149
 Pane Collapse 156, 206
 Pane Expand 156, 206
 Source 149
Branch Optimizations 439
Break command 210
Break menu command 525
Break On C++ Exception menu command 525
Break on Java Exceptions command 525
Breakpoint Properties button 224
breakpoint template 231
breakpoint template, defined 222
breakpoint templates
 creating 231

deleting 232
specifying the default 233
working with 231

Breakpoints 222

breakpoints

- Breakpoint Type property 227
- clearing all 229
- Condition property 227
- conditional 222
- default template 231
- defined 222
- disabled 222, 244
- enabled 222
- File-Info property 227
- Hardware property 228
- Hit Count property 227
- Name property 227
- Original Process property 227
- Original-Target property 228
- purpose of 221
- regular 222
- saving sets of 225
- Serial Number property 227
- setting conditional 230
- setting temporary 229
- template 231
- temporary 222
- Thread property 228
- Times Hit property 228
- Times Left property 228
- viewing 225
- working with 226

Breakpoints button 206

Breakpoints column, in editor window 95

Breakpoints menu command 525

Breakpoints window 222

- Active icon 225
- Breakpoint Properties button 224
- Create Breakpoint Template button 224
- Groups tab 224
- Inactive icon 225
- Instances tab 225
- opening 225
- Rename Breakpoint button 224
- saving contents of 225
- Set Default Breakpoint Template button 224
- Templates tab 225

Breakpoints Window menu command 525

breakpoints, clearing 229

breakpoints, disabling 228

breakpoints, enabling 228, 242

breakpoints, setting 226

breakpoints, viewing properties for 227

Bring To Front menu command 525

Bring Up To Date

- menu command 56, 57

Bring Up To Date menu command 463, 526

Browse button 132, 271, 273, 288, 292, 300

Browse In Processes Window option 420, 421

browser 164

- Class Browser window 167
- Classes pane 174
- collapsing panes 173
- creating new classes 174, 189, 190
- creating new data members 197
- creating new member functions 194, 195, 197
- expanding panes 173
- hierarchy windows 180
- Member Functions pane 175
- overview 27
- printing class hierarchies 181
- purpose of 161
- setting options 161
- Source pane 177
- status area 178
- viewing data by contents 184
- viewing data by inheritance 180
- working with 161

Browser Access Filters 169

Browser Commands option 452

- Editor Settings panel 467

Browser Contents 168

Browser Contents command 526

Browser Contents window 183

- Symbols list 184

browser database

- defined 161

Browser menu 452

Browser Path option 453

Browser Wizard 189

Build Before Running option 453

Build Extras panel

- options

Initial Directory field 466
Use External Debugger 486
Use modification date caching 486

Build Extras settings panel 275, 430
options
 Application 432
 Arguments 432
 Cache Subprojects 431
 Dump internal browse information after compile 432
 Generate Brower Data From 432
 Initial directory 432
 Use External Debugger 432
 Use modification date caching 431

Build Extras target settings panel 535

Build Progress menu command 526

Build Progress Window menu command 526

Build Settings panel
options
 Include file cache 466
 Play sound after ‘Bring Up To Date’ & ‘Make’ 471
 Save open files before build 474
 Show message after building up-to-date project 478
 Success 483
 Use Local Project Data Storage 486

Build Settings preference panel 387
options
 Build before running 388
 Compiler thread stack 389
 Failure 388
 Include file cache 388
 Play sound after ‘Bring Up To Date’ & ‘Make’ 388
 Save open files before build 388
 Show message after building up-to-date project 388
 Success 388
 Use Local Project Data Storage 389

build system
 overview 27

build targets 33
 configuring 60
 creating 58
 management 53
 managing 58

moving 54
removing 53, 59
renaming 56, 60
setting default 59
strategies for 43

Bus Noise checkbox 304

Bus Noise test
subtests
 Full Range Converging 308
 Maximum Invert Convergence 308
 Sequential 307

bus noise, defined 307

Button
Choose 372
Delete 373
Export 382
Import 383
New Binding 381
Save 373

buttons
Access Target 301
Add 429
Add Default 449
Add Files 139
Apply 153
Begin Scope Loop 303
Begin Test 305
Blank Check 295
Breakpoint Properties 224
Breakpoints 206
Browse 132, 271, 273, 288, 292, 300
Calculate Checksum 297
Cancel 124, 127, 287, 298
Change 429
Clear List 139
Compare 150
Create Breakpoint Template 224
Debug 206
Details 293, 295, 297, 301, 303, 305
Edit 410
Erase 295
Export Panel 459
Expressions 206
Factory Settings 462
Find 124, 127, 130
Find All 124, 130
Installed Products 523

Kill 206
Line and Column 208
Load Settings 287, 298
Next Result 142
OK 287, 298
Previous Result 142
Program 294
Purge Cache 473
Read 273
Redo 154
Remove 429
Remove a Set 139
Rename Breakpoint 224
Replace 127, 130
Replace All 127, 130
Reset Value 273
resetting in toolbars 378
Resume 206
Revert 273
Run 206
Save Settings 287, 298
Save This Set 139
Set Default Breakpoint Template 224
Show Log 286, 305
Source File 207
Step Into 206
Step Out 206
Step Over 206
Stop 130, 142, 206
Symbolics 206
Unapply 153
Undo 153
Variables Pane Listing 207
Verify 294
Warnings 142
Write 273
By Type text/list box 132
Byte option button 301, 302, 304

C

cache
 purging 473
Cache Edited Files Between Debug Sessions
 option 453
Cache Subprojects option 453
Cache Symbolics Between Runs option 454
Cache window 315

opening 315
caching
 #include files 466
 precompiled headers 466
Calculate Checksum button 297
Can't Redo menu command 496, 510
Can't Undo menu command 496, 510
Cancel button 124, 127, 287, 298
Cancel button, in Remove Markers window 117
Cascade menu command 526
Case Sensitive checkbox 124, 127, 131, 149
Case Sensitive option 454
Change button 429
Change Program Counter menu command 526
changing
 find strings 144
 line views in a hierarchical window 182
 register data views 269
 register values 268
 remote connections 420
 source trees 398
Check Syntax command 526
Checkbox
 Numeric Keypad Bindings 381
checkboxes
 Address 304
 All Sectors 295
 Analyzer Can Cause Target Breakpoint 312
 Appears in Menus 144
 Apply Address Offset 293
 Bus Noise 304
 Case sensitive 124, 127, 131, 149
 Compare text file contents 150
 Enable Logging 289
 Erase Sectors Individually 295
 Ignore extra space 150
 Log Message 235
 Match whole word 124, 127, 131
 Only show different files 150
 Project headers 134
 Project sources 134
 Regular expression 124, 127, 131
 Restrict Address Range 293
 Search cached sub-targets 134
 Search selection only 125, 128
 Search sub-folders 132
 Search up 125, 128

Speak Message 236
Stop at end of file 125, 127
Stop in Debugger 236, 238, 240
System headers 134
Target Breakpoint Can Cause Analyzer
 Trigger checkbox 312
Treat as Expression 236
Use Custom Settings 288, 299
Use Selected File 292
Use Target CPU 305
Use Target Initialization 288, 300
View Target Memory Writes 289
Walking 1's 304
Checkout Status column
 in Files view of Project window 48
Checksum panel 295
child windows, defined 68
choosing
 a default project 39
 linkers 359
 one character from many in regular
 expressions 147
class browser
 purpose of windows 167
 working with windows 167
Class Browser menu command 527
Class Browser window 167
 Classes pane 169
 Data Members pane 169
 Member Functions pane 169
 Status area 169
class data
 viewing from hierarchy windows 172
Class Declaration 178
Class Hierarchy 168
Class Hierarchy menu command 527
Class Hierarchy Window menu command 527
class hierarchy windows
 purpose of 179
 working with 179
class view 339
classes
 creating 174, 189, 190
 hiding pane for 175
 showing pane for 175
 sorting list of 175
Classes option 410
Classes pane 174
 in Class Browser window 169
classes.zip 524
Clear All Breakpoints menu command 527
Clear All Watchpoints menu command 527
Clear Breakpoint menu command 527
Clear Eventpoint menu command 528
Clear Floating Toolbar command in Toolbar
 submenu 528
Clear List button 139
Clear Main Toolbar menu command 528
Clear menu command 527
Clear Watchpoint menu command 528
Clear Window Toolbar command in Toolbar
 submenu 528
clearing
 all breakpoints 229
 all watchpoints 247
 breakpoints 229
 watchpoints 247
client area, defined 68
Clone Existing Target option 58
Close All command 65
Close All Editor Documents menu command 528
Close All menu command 528
Close Catalog menu command 529
Close command 40, 65
Close menu command 528
Close Non-debugging Windows option 454
Close Workspace menu command 529
closing
 all files 65
 dockable windows 77
 files 65
 projects 40
 workspaces 81
Code 440
code
 adding markers to 117
 completing 102
 disabling breakpoints 228
 disabling eventpoints 242
 disabling special breakpoints 248
 disabling watchpoints 246
 enabling breakpoints 228, 242
 enabling special breakpoints 248
 enabling watchpoints 246

locating 113
navigating 113
setting breakpoints in 226
setting watchpoints in 245
viewing breakpoint properties 227
viewing eventpoint properties 242
viewing watchpoint properties 246

Code column
in Files view of Project window 48

code completion
activating automatic behavior 103
configuration 103
deactivating automatic behavior 105
for data members 109
for parameter lists 109
navigating window 107
selecting items 108
triggering by keyboard 104
triggering from IDE menu bar 104

Code Completion Delay option 454

Code Completion preference panel 400
options
Automatic Invocation 400
Case sensitive 401
Code Completion Delay 401
Display deprecated items 401
Window follows insertion point 400

Code Completion window 105
code completion, triggering from IDE menu bar 104

Code Formatting preference panel 401
options
Close Braces, Brackets, And Parentheses 403
Format Braces 402
Indent Braces 403
Indent Case Within Switch Statement 403
Indent Code Within Braces 403
Language Settings 402
Place Else On Same Line As Closing Brace 403
Place Opening Brace On Separate Line 403
Use Automatic Code Formatting 402

Code Only option button 125, 128, 131

CodeWarrior

menu reference 493
overview 23

CodeWarrior Glossary command 529

CodeWarrior Help menu command 529

CodeWarrior IDE
Apple menu 507
CodeWarrior menu 507
Data menu 504, 517
Debug menu 502, 515
Edit menu 495, 510
File menu 493, 508
Help menu 506, 522
Project menu 500, 513
Scripts menu 522
Search menu 498, 511
Tools menu 521
VCS menu 521
Window menu 497, 506, 519

CodeWarrior menu 507

CodeWarriorU.com 18

Collapse Non-debugging Windows option 455

Collapse Window menu command 529

collapsing
browser panes 173
dockable windows 76

collection method 331

COM 457

Command Actions
Arguments 368
Defining (Mac OS) 372
Defining (Windows) 368
Directory 368
Execute 368

Command Group
Delete 373

Command Groups 372
Delete 372

Command window 317
issuing command lines 318
opening 318

command-line window 317

Commands
Import 382
Modify 365

commands 175
About Metrowerks CodeWarrior 523
Add Files 523

Add Window 523
Apply Difference 154, 524
Arm 313
Balance 525
Bottom Edges 525
Break 210, 525
Break On C++ Exception 525
Break on Java Exceptions 525
Breakpoints 525
Breakpoints Window 525
Bring To Front 525
Bring Up To Date 526
Browser Contents 168, 526
Build Progress 526
Build Progress Window 526
Can't Redo 496, 510
Can't Undo 496, 510
Cascade 526
Change Program Counter 526
Check Syntax 526
Class Browser 527
Class Declaration 178
Class Hierarchy 168, 527
Class Hierarchy Window 527
Clear 527
Clear All Breakpoints 527
Clear All Watchpoints 527
Clear Breakpoint 527
Clear Eventpoint 528
Clear Main Toolbar 528
Clear Watchpoint 528
Close 40, 528
Close All 528
Close All Editor Documents 528
Close Catalog 529
Close Workspace 529
CodeWarrior Glossary 529
CodeWarrior Help 529
Collapse Window 529
Commands & Key Bindings 529
Compare Files 150, 529
Compile 530
Complete Code 529
Connect 312, 530
Copy 530
Copy To Expression 530
Create Design 530
Create Group 530
Create Target 530
Cut 531
Cycle View 531
Debug 208, 531
Delete 531
Diagonal Line 182
Disable Breakpoint 531
Disable Watchpoint 532
Disarm 313
Disassemble 532
Disconnect 314
Display Grid 532
Enable Breakpoint 532
Enable Watchpoint 532
Enter Find String 144, 533
Enter Replace String 533
Errors And Warnings 533
Errors And Warnings Window 533
Exit 533
Expand Window 534
Export Project 40, 534, 539
Export Project as GNU Makefile 534
Expressions 534
Expressions Window 534
File Path 49
Find 125, 534
Find and Open 'Filename' 536
Find and Open File 536
Find And Replace 536
Find Definition 535
Find Definition & Reference 120, 534
Find In Files 535
Find In Next File 535
Find In Previous File 535
Find Next 143, 535
Find Previous 143, 536
Find Previous Selection 536
Find Reference 120, 536
Find Selection 144, 537
Get Next Completion 537
Get Previous Completion 537
Global Variables 537
Global Variables Window 537
Go Back 168, 537
Go Forward 168, 537
Go To Line 537

Hide Breakpoints 538
Hide Classes 175
Hide Classes pane 178
Hide Window Toolbar 538
Import Components 539
Import Project 40, 539
Insert Reference Template 539
Kill 211, 539
Make 540
Maximize Window 540
Metrowerks Website 540
Minimize Window 540
New 540
New Class 540
New Class Browser 540
New Data 541
New Event 541
New Event Set 541
New Expression 541
New Item 174
New Member Function 541
New Method 541
New Property 541
New Text File 541
Open 542
Open File 177
Open In Windows Explorer 49
Open Recent 542
Open Scripts Folder 542
Open Workspace 542
Page Setup 542
Pane Collapse 173
Pane Expand 173
Precompile 543
Preferences 543
Print 543
Proceses 543
Proceses Window 543
Project Inspector 38
Redo 543
Refresh All Data 544
Register Details Window 270, 544
Register Windows 544
Registers 544
Remove Object Code 544
Remove Object Code & Compact 544
Remove Toolbar Item 377
Replace 128, 545
Replace All 545
Replace and Find Next 545
Restart 211
Resume 210, 547
Revert 547
Run 211, 472, 547
Run To Cursor 548
Save Default Window 548
Save Workspace 548
Save Workspace As 549
Select All 549
Send To Back 549
Set Breakpoint 549
Set Default Project 40, 549
Set Default Target 549
Set Eventpoint 549
Set Watchpoint 550
Shift Right 550
Show Breakpoints 527, 550
Show Classes 175
Show Classes pane 178
Show Inherited 169
Show private 170
Show protected 170
Show public 170
Show Types 550
Show Window Toolbar 538
Single Class Hierarchy Window 169
Sort Alphabetical 174, 175
Sort Hierarchical 174
Stack Editor Windows 551
Step Into 209
Step Out 209
Step Over 210, 551
Stop 210
Stop Build 551
Straight Line 182
Switch To Monitor 551
Symbolics 551
Symbolics Window 551
Synchronize Modification Dates 551
Unapply Difference 155
Update Data 313
View Array 556
View as implementor 170
View as subclass 170

View As Unsigned Decimal 556, 557
View as user 170
View Disassembly 557
View Mixed 558
View Source 558
View Variable 558
Zoom Window 558

Commands & Key Bindings menu command 529
Commands tab 363, 365, 379
Commands&KeyBindings.mkb file 382
Comments Only option button 125, 128, 131
Comments option 455
common debugging actions 208
Common Subexpression Elimination 439
Compare button 150
Compare Files command 150
Compare Files menu command 529
Compare Files Setup window 149
 Case Sensitive checkbox 149
 Compare button 150
 Compare Text File Contents checkbox 150
 Destination box 149
 Ignore Extra Space checkbox 150
 Only Show Different Files checkbox 150
 Source box 149
Compare Text File Contents checkbox 150
comparing files
 differences, applying 154
 differences, unapplying 155
 overview 148
 setup 149, 150
comparing files, explained 152
comparing folders
 examining results 157
 overview 148
 setup 149, 151
comparing folders, explained 155
comparison
 destination item 148
 source item 148
Compile menu command 530
compiler
 avoiding crashes 455
compiler directives 330, 343
Compiler option 455
Compiler option, in Generate Browser Data From menu 464

compiler thread stack
 and avoiding compiler crashes 455
Compiler Thread Stack field 455
Complete Code menu command 529
completing code 102
Component Object Model. *See COM.*
Concurrent Compiles panel
 options
 Use Concurrent Compiles 473, 485
 User Specified 488
Concurrent Compiles preference panel 389
 options
 Recommended 390
 Use Concurrent Compiles 390
 User Specified 390
condition, breakpoint property 227
conditional access breakpoint 247
conditional breakpoint, defined 230
conditional breakpoints 222
 setting 230
conditional eventpoint, defined 243
conditional eventpoints
 setting 243
conditional watchpoint, defined 247
conditional watchpoints
 setting 247
Configuration panel 299
configuring
 build targets 60
 code completion 103
 projects for a logic analyzer 309
 targets 60
Confirm “Kill Process” When Closing Or Quitting option 456
Confirm Invalid File Modification Dates When Debugging option 455
Connect command 312
Connect menu command 530
Connection list box 288, 300
Connection pop-up menu, in Remote Debugging settings panel 447
Connection Type list box 311
Connection Type option 420, 421
console applications
 creating 83, 84
 applications
 creating console

applications 84
console applications, about 83
constant
 adding to a variable 256
Constants option 410
contents
 of register 270
Context Popup Delay option 456
contextual menus 212
 File Path command 49
 Open In Windows Explorer command 49
 using 214
 using to dock a window 70
controlling program execution 203
conventions
 figures 22
 for manual 21
 keyboard shortcuts 22
Copy And Expression Propagation 439
Copy menu command 530
Copy Propagation 439
Copy To Expression command 530
cores, debugging multiple 214
Create Breakpoint Template button 224
Create Design menu command 530
Create Group menu command 530
Create Target command 58
Create Target menu command 530
creating
 a new data member 177
 build targets 58
 console application 84
 console applications 83, 84
 custom project stationery 41
 empty projects 37
 files (Macintosh) 62
 files (Windows) 61
 member functions 176
 new classes 174, 189, 190
 new data member 197
 new data members 197
 new member function 194
 new member functions 195
 projects from makefiles 36
 projects using stationery 35
 subprojects 42
 targets 58

cross-platform migration, and opening projects 38
Current Target list pop-up 46
Current Target menu 377
current-statement arrow 207
custom project stationery 41
Customize IDE Commands window 143, 363, 379, 381
 Action 365
 Appears in Menus 365, 366
 Appears in Menus checkbox 144
 Auto Repeat 366
 Key Bindings 366
 Name field 365
 New Binding 366
 New Group 366
 Numeric Keypad Bindings checkbox 384
Cut command 531
CVS 396
Cycle View menu command 531

D

dash 207
data
 finding problems 340
 sorting 337
 viewing 335
Data column
 in Files view of Project window 48
data columns
 contents 336
Data Line fault 306
data members
 completing code 109
 creating 177, 197
 identifier icons 176
Data Members pane 176
 in Class Browser window 169
Data menu 504, 517
data, for debugger, working with 275
database
 navigation for browser 164
deactivating automatic code completion 105
Dead Code Elimination 439
Dead Store Elimination 439
Debug button 206
Debug column

in Files view of Project window 48
Debug command 56, 57, 208
Debug menu 456, 502, 515
 Clear All Breakpoints command 503, 516
 Disable Watchpoint command 503, 504, 517
 Enable Breakpoint command 503, 516
 Enable Watchpoint command 503, 504, 516,
 517
 Hide Breakpoints command 503, 517
Debug menu command 531
debugger 473
 attaching to a process 281
 choosing for an opened symbolics file 420
 overview 27
 restarting 211
 starting 208
 working with data 275
 working with memory 259
 working with variables 249
Debugger Commands option 456
Debugger list box 311
Debugger section, of IDE preference panels 412
Debugger Settings panel 281, 445
 options
 Auto-target Libraries 446
 Cache symbolics between runs 446
 Default language entry point 446, 457
 Location of Relocated Libraries and
 Code Resources 446, 469
 Log System Messages 446, 469
 Program entry point 472
 Stop at Watchpoints 446, 480
 Stop on application launch 446, 481
 Update data every n seconds 485
 Update data every *n* seconds 446
 User specified 446
debugger, defined 203
debugging
 common actions 208
 multiple cores 214
 program execution 203
 restarting a session 211
 starting a session 208
Declaration File field 191
default breakpoint template 231
Default File Format option 456
default filename extensions 460
Default Language Entry Point option
 Debugger Settings panel 457
Default Project 288, 299
default projects 39
default size and position of windows, setting 548
Default Size For Unbounded Arrays option 457
Default Target 288, 299
default target, setting 59
default workspace
 definition of 79
 using 80
definition
 of breakpoint template 222
 of breakpoints 222
 of bus noise 307
 of child windows 68
 of client area 68
 of conditional breakpoint 230
 of conditional eventpoint 243
 of conditional watchpoint 247
 of debugger 203
 of default workspace 79
 of dock 67
 of eventpoints 233
 of IDE 17
 of machines 278
 of memory aliasing 307
 of non-modal 69
 of project 31
 of regular expression 145
 of special breakpoints 248
 of symbolics file 204
 of symbols 119
 of temporary breakpoint 229
 of touch 48
 of watchpoints 244
 of workspace 79
Delete menu command 531
Description 272
Description File text box 271, 273
design problems, finding 338
Design view 54
Designs view 39
desktop background
 removing from behind IDE. *See* Use
 Multiple Document Interface, turning on.

seeing behind IDE. *See Use Multiple Document Interface, turning off.*

Destination box 149

destination item, for comparison 148

Destination pane 153

detail view 338

 finding design problems 338

detailed data, collecting 331

details

 viewing for registers 270

Details button 293, 295, 297, 301, 303, 305

development-process cycle for software 23

Device pane 290

diagnostics

 disabling for plug-ins 523

 enabling for plug-ins 523

Diagonal Line 182

dialog boxes

 New Connection 420

difference from Single-Class Hierarchy window 182

Differences pane 154

directives

 C/C++ 343

 compiler 330

Disable Breakpoint menu command 531

Disable Third Party COM Plugins option 457

Disable Watchpoint menu command 532

disabled breakpoint 222, 244

disabled eventpoint 234

disabling

 plug-in diagnostics 523

Disarm command 313

Disassemble menu command 532

disclosure triangles

 Source Code pane 142

 Source pane 207

Disconnect command 314

Display Deprecated Items option 457

Display Grid menu command 532

Display Settings panel 245

 options

 Show all locals 476

 Show tasks in separate windows 479

 Show values as decimal instead of hex 479

 Show variable location 479

Show variable types 480

Show variable values in source code 480

Sort functions by method name in symbolics window 480

Variable Values Change 489

Watchpoint Indicator 489

Display Settings preference panel 412

options

 Attempt to use dynamic type of C++, Object Pascal and SOM objects 414

Default size for unbounded arrays 414

Show all locals 413

Show tasks in separate windows 416

Show values as decimal instead of hex 414

Show variable location 413

Show variable types 413

Show variable values in source code 414

Sort functions by method name in symbolics window 414

Variable values change 413

Watchpoint indicator 413

DLL 418, 450

Do Nothing option 457

Do Nothing To Project Windows option 457

dock bars 76

dock, defined 67

dockable windows 67, 70

 about 67

 closing 77

 collapsing 76

 dock bars 76

 docking windows of the same kind 72

 expanding 76

 moving 77

 suppressing 75

 turning off 75

 types 68

Document Settings list pop-up 93

document settings pop-up

 using 93

documentation

 formats 20

 structure 20

 types 21

Documents option
 IDE Extras panel 457

Don't Step Into Runtime Support Code 458

Don't Step Into Runtime Support Code
 option 458

Done button, in Remove Markers window 117

drag and drop
 using to dock a window 71

Drag And Drop Editing option 458

Dump Internal Browse Information After
 Compile option 458

dump memory 557

E

Edit button 410

Edit Commands option 458

Edit Language option 458

Edit menu 458, 495, 510

editing
 source code 97
 symbols, shortcuts for 100

editor 89
 overview 27
 third-party support 488

Editor section, of IDE preference panels 399

Editor Settings panel
 options
 Balance Flash Delay 452
 Browser Commands 467
 Font Preferences 464
 Insert Template Commands 466
 Left margin click selects line 468
 Project Commands 472
 Relaxed C popup parsing 473
 Selection position 476
 Sort function popup 483
 Use multiple undo 487
 VCS Commands 489
 Window position and size 490

Editor Settings preference panel 403
 options
 Balance Flash Delay 405
 Balance while typing 405
 Browser Commands 405
 Debugger Commands 405
 Default file format 405
 Drag and drop editing 405

Edit Commands 404

Enable Virtual Space 405

Font preferences 404

Insert Template Commands 405

Left margin click selects line 405

Project Commands 405

Relaxed C popup parsing 405

Selection position 404

Sort function popup 405

Use multiple undo 405

VCS Commands 405

Window position and size 404

editor toolbar 92

editor window 89
 adding panes to 95
 Breakpoints column 95
 collapsing toolbar in 92
 expanding toolbar in 92
 line and column indicator 95
 pane splitter controls 95
 removing panes from 96
 resizing panes 95
 text editing area 95

editor windows
 other 94
 selecting text in 97

Emacs text editor 467, 468

empty projects
 creating 37

Empty Target option 58

Enable Automatic Toolbar Help option 459

Enable Breakpoint menu command 532

Enable Browser option 526

Enable Logging checkbox 289

Enable Remote Debugging option 459

Enable Virtual Space option 459

Enable Watchpoint menu command 532

enabled breakpoint 222

enabled eventpoint 234

enabled watchpoint 244

enabling
 plug-in diagnostics 523

End text box 293, 304

end-of-line format 456

enlarging panes, in browser 173

Enter Find String command 144

Enter Find String menu command 533

Enter Replace String menu command 533
Entire Flash option button 296
Enums option 410
Environment Settings option 459
Environment Variable option
 of Source Trees preference panel 484
Environment Variable option, in Type pop-up
 menu 484
environment variables
 Macintosh limitations 484
EOL format 456
Erase / Blank Check panel 294
Erase button 295
Erase Sectors Individually checkbox 295
Errors And Warnings menu command 533
Errors And Warnings Window menu
 command 533
Errors Only option
 of Plugin Diagnostics 469
eventpoints
 defined 233
 disabled 234
 enabled 234
 Log Point 234, 235
 Log Point, clearing 236
 Log Point, setting 235
 Pause Point 234, 236
 Pause Point, clearing 237
 Pause Point, setting 236
 purpose of 221
 Script Point 234, 237
 Script Point, clearing 238
 Script Point, setting 237
 setting conditional 243
 Skip Point 234, 238
 Skip Point, clearing 239
 Skip Point, setting 239
 Sound Point 234, 239
 Sound Point, clearing 240
 Sound Point, setting 239
 Sound Point, Speak Message 239
 Trace Collection Off 234, 240
 Trace Collection Off, clearing 241
 Trace Collection Off, setting 240
 Trace Collection On 234, 241
 Trace Collection On, clearing 241
 Trace Collection On, setting 241
 working with 242
eventpoints, disabling 242
eventpoints, viewing properties for 242
examining debugger data 275
examining memory 259
examining variables 249
exceptions 333
Exceptions In Targeted Classes command in Java
 Exceptions submenu 533
executable files
 adding to the Other Executables list 444
 changing in the Other Executables list 444
 removing from the Other Executables
 list 445
execution
 of program, controlling 203
execution, killing 211
execution, resuming 210
execution, stopping 210
Exit menu command 533
exit() 334
Expand Window menu command 534
expanding
 browser panes 173
 dockable windows 76
Export 382
Export Panel button 364, 386, 459
Export Project as GNU Makefile menu
 command 534
Export Project command 40
Export Project menu command 534, 539
exporting
 projects to XML files 40
Expression Simplification 439
Expressions button 206
Expressions menu command 534
Expressions window 254
 adding expressions 255
 opening 255
Expressions Window menu command 534
Extension field 460
external editor
 using on the Macintosh 392
external editor support 488

F

Factory Settings button 462

Failure option 463
FDI 391, 487
 and dockable windows 67
fields
 Application 450
 Arguments 450
 Base Classes 192
 Compiler thread stack 455
 Declaration File 191
 Extension 460
 File Type 463
 IP Address 420
 Relative to class 191
 Run App/Script 372
figure conventions 22
File
 Commands&KeyBindings.mkb 382
File column
 in Files view of Project window 48
%file command-line string 467
File Compare Results window 152
 Apply button 153
 Destination pane 153
 Differences pane 154
 pane resize bar 153
 Redo button 154
 Source pane 153
 Unapply button 153
 Undo button 153
File list 134
file management 53
File Mappings list 455
File Mappings settings panel 434
 options
 Add 436
 Change 436
 Compiler 436
 Edit Language 436
 Extension 435
 File Mappings list 435
 File Type 435
 Flags 436
 Ignored By Make flag 436
 Launchable flag 436
 Precompiled File flag 436
 Remove 436
 Resource File flag 435
File menu 493, 508
 New Text File command 508
file modification icon 94
File On Host option button 296
File On Target option button 296
File Path command 49
file paths
 viewing 49
File Set list 139
File Set list box 139
File Type field 463
File Type option 455
file-info, breakpoint property 227
filename extensions
 default settings 460
files
 close all 65
 closing 65
 comparing 152
 creating (Macintosh) 62
 creating (Windows) 61
 destination (for a comparison) 148
 inspecting 38
 moving 54
 opening 62
 print selections 66
 printing 66
 renaming 55
 replacing text in 128
 reverting 66
 save all 64
 saving 63
 saving copies 64
 searching (multiple) 139
 searching (single) 125
 source (for a comparison) 148
 touching 56
 touching all 56
 untouching 57
 untouching all 57
 working with 61
Files In Both Folders pane 157
Files Only In Destination pane 157
Files Only In Source pane 157
Files page, about 47
Files tab 53
Files view 39, 54, 57

Checkout Status column 48
Code column 48
Data column 48
Debug column 48
File column 48
Interfaces list pop-up 49
Sort Order button 49
Target column 48
Touch column 48
files, tasks for managing 61
Find
 by text selection 143
 single-file 123
Find All button 124, 130
Find and compare operations option
 Shielded Folders panel 463
Find And Open ‘Filename’ menu command 536
Find and Open File command 536
Find and Replace
 multiple-file 129
 single-file 126
Find And Replace menu command 536
Find and Replace window
 All Text option button 128
 Cancel button 127
 Case Sensitive checkbox 127
 Code Only option button 128
 Comments Only option button 128
 Find button 127
 Find text/list box 127
 Match Whole Word checkbox 127
 Regular Expression checkbox 127
 Replace All button 127
 Replace button 127
 Replace With text/list box 127
 Search Selection Only checkbox 128
 Search Up checkbox 128
 Stop At End Of File checkbox 127
Find button 124, 127, 130
Find command 125, 534
Find Definition & Reference command 120
Find Definition & Reference menu command 534
Find Definition menu command 535
Find In Files menu command 535
Find in Files window
 All Text option button 131
 Case Sensitive checkbox 131
Code Only option button 131
Comments Only option button 131
Find All button 130
Find button 130
Find text/list box 130
In Files page 138, 139
 Add Files button 139
 Clear List button 139
 File Set list 139
 File Set list box 139
 Remove A Set button 139
 Save This Set button 139
In Files tab 131
In Folders page 131, 133
 Browse button 132
 By Type text/list box 132
 Search In text/list box 132
 Search Sub-Folders checkbox 132
In Folders tab 131
In Projects page 134, 135
 File list 134
 Project Headers checkbox 134
 Project list box 134
 Project Sources checkbox 134
 Search Cached Sub-Targets
 checkbox 134
 System Headers checkbox 134
 Target list box 134
In Projects tab 131
In Symbolics page 136, 137
 Symbolics list 136
 Symbolics list box 136
In Symbolics tab 131
 Match Whole Word checkbox 131
 Regular Expression checkbox 131
 Replace All button 130
 Replace button 130
 Replace With text/list box 130
 Stop button 130
Find In Next File menu command 535
Find In Previous File menu command 535
Find Next
 using 143
Find Next command 143
Find Next menu command 535
Find Previous
 using 143

Find Previous command 143
enabling in the Customize IDE Commands window 143
Find Previous menu command 536
Find Previous Selection menu command 536
Find Reference command 120
Find Reference menu command 536
Find Reference using option
 IDE Extras panel 463
Find Selection command 144
Find Selection menu command 537
Find symbols with prefix 100
Find symbols with substring 100
Find text/list box 124, 127, 130
Find window
 All Text option button 125
 Cancel button 124
 Case Sensitive checkbox 124
 Code Only option button 125
 Comments Only option button 125
 Find All button 124
 Find button 124
 Find text/list box 124
 Match Whole Word checkbox 124
 Regular Expression checkbox 124
 Search Selection Only checkbox 125
 Search Up checkbox 125
 Stop At End Of File checkbox 125
finding problems 340
finding text
 overview 123
Flags pop-up menu 436
 Ignored By Make flag 436
 Launchable flag 436
 Precompiled File flag 436
 Resource File flag 435
Flash Base + Offset 293
Flash Base Address 293
Flash Configuration panel 289
Flash Memory Base Address text box 290
Flash Programmer pane 286
flash programmer panels
 Checksum 295
 Erase / Blank Check 294
 Flash Configuration 289
 Program / Verify 291
 Target Configuration 287

Flash Programmer window 285
 Cancel button 287
 Checksum panel
 Calculate Checksum button 297
 Details button 297
 Entire Flash option button 296
 File On Host option button 296
 File On Target option button 296
 Memory Range On Target option button 296
 Size text box 297
 Start text box 297
 Status 297
 Erase / Blank Check panel
 All Sectors checkbox 295
 All Sectors list 295
 Blank Check button 295
 Details button 295
 Erase button 295
 Erase Sectors individually checkbox 295
 Status 295
 Flash Configuration panel
 Device pane 290
 Flash Memory Base Address text box 290
 Organization pane 291
 Sector Address Map pane 291
 Flash Programmer pane 286
 Load Settings button 287
 OK button 287
 opening 285
 Program / Verify panel
 Apply Address Offset checkbox 293
 Browse button 292
 Details button 293
 End text box 293
 Flash Base + Offset 293
 Flash Base Address 293
 Offset text box 293
 Program button 294
 Restrict Address Range checkbox 293
 Start text box 293
 Status 293
 Use Selected File checkbox 292
 Use Selected File text box 292
 Verify button 294

Save Settings button 287
Show Log button 286
Target Configuration panel
 Browse button 288
 Connection list box 288
 Default Project 288
 Default Target 288
 Enable Logging checkbox 289
 Target Memory Buffer Address text
 box 288
 Target Memory Buffer Size text
 box 288
 Target Processor text/list box 288
 Use Custom Settings checkbox 288
 Use Target Initialization checkbox 288
 Use Target Initialization text box 288
 View Target Memory Writes
 checkbox 289

flat view 337
floating a window 74
Floating Document Interface. *See FDI.*
floating window type 68
focus bar 55
Folder Compare Results window 155
 Files In Both Folders pane 157
 Files Only In Destination pane 157
 Files Only In Source pane 157
 Pane Collapse box 156
 Pane Expand box 156
 pane resize bar 157
 Selected Item group 157

folders
 comparing 155
 Registers 270
 searching (multiple) 133

Font & Tabs panel 407
options
 Font 464
 Scripts 475
 Size 480
 Tab indents selection 484
 Tab Inserts Spaces 484
 Tab Size 484

Font & Tabs preference panel 406, 408
options
 Auto Indent 407
 Font 406

Script 406
Size 406
Tab indents selection 406
Tab Inserts Spaces 407
Tab Size 407

Font option
 Font & Tabs panel 464

Font Preferences option
 Editor Settings panel 464

Font Settings 407

Foreground option
 Text Colors panel 464

Format list box 271

format, for end of line (EOL) 456

formats
 for documentation 20

FPU Registers 267

Framework column, in Access Paths panel 430

Full Range Converging subtest 308

function
 New Data Member 176

function-level profiling 326, 343

functions
 creating new member 176
 locating 113, 114

Functions list box 208

Functions list pop-up 92
 sorting alphabetically 115
 using 114

Functions option 410

G

General Registers 267

General section, of IDE preference panels 387

Generate Browser Data From option 464
 Compiler 464
 Language Parser 465
 Language Parser, Macro file 465
 Language Parser, Prefix file 465
 None 464

Generate Constructor and Destructor 192

Get Next Completion menu command 537

Get next symbol 100

Get Previous Completion menu command 537

Get previous symbol 100

Global Optimizations settings panel 437
options

Details 438
Faster Execution Speed 438
Optimization Level slider 438
Smaller Code Size 438
Global Register Allocation 439
Global Register Allocation Only For Temporary Values 439
Global Settings panel
options
 Maintain Files in Cache 470
 Select stack crawl window when task is stopped 475
Global Settings preference panel
options
 Auto Target Libraries 418
 Automatically launch applications when SYM file opened 417
 Cache Edited Files Between Debug Sessions 417
 Confirm "Kill Process" when closing or quitting 417
 Confirm invalid file modification dates when debugging 417
 Don't step into runtime support code 418
 Maintain files in cache 417
 Purge Cache 417
 Select stack crawl window when task is stopped 418
Global Variables menu command 537
Global Variables window 249
 opening 250
 viewing for different processes 250
Global Variables Window menu command 537
Globals option 410
Go Back 168
Go Back menu command 537
Go Forward 168
Go Forward menu command 537
Go To Line menu command 537
going back 116
going forward 116
going to a particular line 116
gray background, adding behind IDE. *See* Use Multiple Document Interface, turning on.
gray background, removing from behind IDE. *See* Use Multiple Document Interface, turning off.

Grid Size X option
 Layout Editor panel 465
Grid Size Y option
 Layout Editor panel 465
group management 53
grouping
 regular expressions 147
groups
 moving 54
 removing 53
 renaming 55
 Selected Item 157
 touching 56
 touching all 56
 untouching 57
 untouching all 57
Groups tab 224

H

hardware diagnostic panels
 Configuration 299
 Memory Read / Write 300
 Memory Tests 303
 Address 307
 Bus Noise 307
 Bus Noise in address lines 307
 Bus Noise in data lines 308
 Walking Ones 306
 Scope Loop 301
Hardware Diagnostics pane 298
Hardware Diagnostics window 297
 Cancel button 298
 Configuration panel
 Browse button 300
 Connection list box 300
 Default Project 299
 Default Target 299
 Target Processor text/list box 300
 Use Custom Settings checkbox 299
 Use Target Initialization checkbox 300
 Use Target Initialization text box 300
 Hardware Diagnostics pane 298
 Load Settings button 298
 Memory Read / Write panel
 Access Target button 301
 Byte option button 301
 Details button 301

Long Word option button 301
Read option button 301
Status 301
Target Address text box 301
Value to write text box 301
Word option button 301
Write option button 301

Memory Tests panel
Address checkbox 304
Begin Test button 305
Bus Noise checkbox 304
Byte option button 304
Details button 305
End text box 304
Long Word option button 305
Passes text box 305
Show Log button 305
Start text box 304
Status 305
Target Scratch Memory End text box 305
Target Scratch Memory Start text box 305
Use Target CPU checkbox 305
Walking 1's checkbox 304
Word option button 305

OK button 298
opening 297
Save Settings button 298
Scope Loop panel
Begin Scope Loop button 303
Byte option button 302
Details button 303
Long Word option button 303
Read option button 302
Speed slider 303
Status 303
Target Address text box 303
Value to write text box 303
Word option button 303
Write option button 302

hardware tools, working with 285
hardware, breakpoint property 228
headers
caching precompiled headers 466
Help menu 506, 522
Help Preferences panel 393

options
Browser Path 393
Set 393
Hide Breakpoints menu command 538
Hide Classes 175
Hide Classes pane 178
Hide Floating Toolbar command 538
Hide Main Toolbar command in Toolbar submenu 538
Hide non-debugging windows option
Windowing panel 465
Hide Window Toolbar command 538

hiding
classes pane 175
Hierarchy Control 180
hierarchy window 180
hierarchy windows
changing line views 182
using to view class data 172
hit count, breakpoint property 227
Horizontal Center command 538
Host Application for Libraries & Code Resources option
Runtime Settings panel 465
Host Application For Libraries And Code Resources field
of Runtime Settings panel 465
Host Name text box 311
host-specific registers 267
how to
activate automatic code completion 103
add a constant to a variable 256
add a keyword to a keyword set 441
add an executable file 444
add expressions (Expressions window) 255
add markers to a source file 117
add panes to an editor window 95
add remote connections 420
add source trees 398
adding subprojects to a project 42
alphabetize Functions list pop-up order 115
apply file differences 154
arm a logic analyzer 313
attach the debugger to a process 281
balance punctuation 101
change an executable file 444

change line views in a hierarchical window 182
change register data views 269
change register values 268
change remote connections 420
change source trees 398
change the find string 144
choose a default project 39
choose files to compare 150
choose folders to compare 151
clear a breakpoint 229
clear a Log Point 236
clear a Pause Point 237
clear a Script Point 238
clear a Skip Point 239
clear a Sound Point 240
clear a Trace Collection Off eventpoint 241
clear a Trace Collection On eventpoint 241
clear a watchpoint 247
clear all breakpoints 229
clear all watchpoints 247
close a docked window 77
close a workspace 81
close projects 40
collapse a docked window 76
collapse browser panes 173
collapse the editor window toolbar 92
complete code for data members 109
complete code for parameter lists 109
connect to a logic analyzer 313
create a breakpoint template 231
create a console application 84
create a new class 174, 189, 190
create a new data member 197
create a new data members 177
create a new member function 176, 194, 195
create custom project stationery 41
create empty projects 37
create new projects from makefiles 36
create new projects using project stationery 35
deactivate automatic code completion 105
delete a breakpoint template 232
disable a breakpoint 228
disable a watchpoint 246
disable an eventpoint 242
disarm a logic analyzer 313
disconnect from a logic analyzer 314
dock a window by using a contextual menu 70
dock a window by using drag and drop 71
dock windows of the same kind 72
enable a breakpoint 228, 242
enable a watchpoint 246
examine items in the Folder Compare Results window 157
expand a docked window 76
expand browser panes 173
expand the editor window toolbar 92
export projects to XML files 40
float a window 74
generate project link maps 360
go to a particular line 116
hide the classes pane 175
import projects saved as XML files 40
indent text blocks 100
insert a reference template 120
issue command lines 318
kill program execution 211
look up symbol definitions 119
make a summation of two variables 256
make a window an MDI Child 75
manipulate variable formats 252
move a docked window 77
navigate browser data 164
navigate Code Completion window 107
navigate to a marker 118
open a recent workspace 82
open a single-class hierarchical window 183
open a workspace 81
open an Array window 266
open projects 37
open projects created on other hosts 38
open registers in a separate Registers window 270
open subprojects 43
open the Breakpoints window 225
open the Cache window 315
open the Command window 318
open the Expressions window 255
open the Flash Programmer window 285
open the Global Variables window 250
open the Hardware Diagnostics window 297
open the Log window 282

open the Processes window 278, 280
open the Profile window 316
open the Registers window 268
open the Symbolics window 277
open the symbols window 187
open the Target Settings window 425
open the Trace window 314
overstrike text (Windows) 99
print class hierarchies 181
print projects 39
remove a keyword from a keyword set 442
remove a marker from a source file 118
remove all markers from a source file 118
remove an executable file 445
remove panes from an editor window 96
remove remote connections 421
remove source trees 399
replace text in a single file 128
resize panes in an editor window 95
restart the debugger 211
resume program execution 210
run a program 211
save a copy of a workspace 81
save a workspace 80
save projects 38
save the contents of the Breakpoints
 window 225
search a single file 125
search for text across multiple files 139
search for text across multiple folders 133
search for text across multiple projects 135
search for text across multiple symbolics
 files 137
search with a text selection 144
select entire routines 98
select item in Code Completion window 108
select lines 98
select multiple lines 98
select rectangular portions of lines 98
select text in editor windows 97
set a breakpoint 226
set a conditional breakpoint 230
set a conditional eventpoint 243
set a conditional watchpoint 247
set a Log Point 235
set a Pause Point 236
set a Script Point 237
set a Skip Point 239
set a Sound Point 239
set a temporary breakpoint 229
set a Trace Collection Off eventpoint 240
set a Trace Collection On eventpoint 241
set a watchpoint 245
show the classes pane 175
sort the classes list 175
specify the default breakpoint template 233
start the debugger 208
step into a routine 209
step out of a routine 209
step over a routine 210
stop program execution 210
suppress dockable windows 75
toggle automatic punctuation balancing 101
toggle the symbol hint 212
trigger code completion by keyboard 104
trigger code completion from IDE menu
 bar 104
unapply file differences 155
undock a window 73
unfloat a window 75
unindent text blocks 100
update data from a logic analyzer 313
use an external editor on the Macintosh 392
use contextual menus 214
use the default workspace 80
use the document settings pop-up 93
use the Executables pane in the Symbolics
 window 277, 278
use the Files pane in the Symbolics
 window 277
use the Find Next command 143
use the Find Previous command 143
use the Functions list pop-up 114
use the Functions pane in the Symbolics
 window 278
use the Interfaces list pop-up 114
use the symbol hint 212
use the VCS pop-up 94
use virtual space 99
view a file path 49
view breakpoint properties 227
view browser data by contents 184
view browser data by inheritance 180
view class data from hierarchy window 172

view eventpoint properties 242
view global variables for different processes 250
view registers 268
view watchpoint properties 246

I

icon
for Tools menu 521
for VCS menu 521

icons
Active 225
file modification 94
for data members 176
for member functions 176
Inactive 225

IDE
and threading 455
Apple menu 507
Code Completion window 105
CodeWarrior menu 507
Data menu 504, 517
Debug menu 502, 515
defined 17
Edit menu 495, 510
editing source code 97
editor 89
File menu 493, 508
Flash Programmer window 285
Hardware Diagnostics window 297
hardware tools 285
Help menu 506, 522
linkers 359
Mac-hosted 376
menu reference 493
preferences, working with 385
project manager and build targets 31
Project menu 500, 513
Scripts menu 522
Search menu 498, 511
target settings, working with 423
Tools menu 521
tools overview 26
User's Guide overview 17
VCS menu 521
Window menu 497, 506, 519
Windows-hosted 376

workspaces 79
IDE Extras 521, 522
IDE Extras panel
options
 Documents 457
 Find Reference using 463
 Launch Editor 467
 Launch Editor w/ Line # 468
 Menu bar layout 470
 Projects 472
 Symbolics 483
 Use Default Workspace' 485
 Use External Editor 486
 Use Multiple Document Interface 487
 Use Script menu 487
 Use ToolServer menu 488
 Workspaces 490
 Zoom windows to full screen 490
IDE Extras preference panel 390
options
 Context popup delay 391
 Documents 391
 Enable automatic Toolbar help 392
 Find Reference using 392
 Launch Editor 391
 Launch Editor w/ Line # 391
 Menu bar layout 391
 Projects 391
 Recent symbolics 391
 Use Default workspace 392
 Use External Editor 392
 Use Multiple Document Interface 391
 Use Script menu 392
 Use Third Party Editor 391
 Use ToolServer menu 392
 Zoom windows to full screen 391
 Use Third Party Editor option 488
IDE Preference Panels list 386
IDE Preference Panels, Font & Tabs 407
IDE Preference Panels, Font Settings 407
IDE preferences
 Activate Browser Coloring 410
 Activate Syntax Coloring 409
 Add 396, 397, 419
 Attempt to use dynamic type of C++, Object Pascal and SOM objects 414
 Auto Indent 407

Auto Target Libraries 418
Automatic Invocation 400
Automatically launch applications when
 SYM file opened 417
Background 409
Balance Flash Delay 405
Balance while typing 405
Browser Commands 405
Browser Path 393
Build before running 388
Cache Edited Files Between Debug
 Sessions 417
Case sensitive 401
Change 396, 397, 419
Choose 397
Classes 410
Close Braces, Brackets, And
 Parentheses 403
Close non-debugging windows 415
Code Completion Delay 401
Collapse non-debugging windows 415
Comments 409
Compiler thread stack 389
Confirm "Kill Process" when closing or
 quitting 417
Confirm invalid file modification dates when
 debugging 417
Constants 410
Context popup delay 391
Debugger Commands 405
Default file format 405
Default size for unbounded arrays 414
Disable third party COM plugins 394
Display deprecated items 401
Do nothing 415
Do nothing to project windows 415
Documents 391
Don't step into runtime support code 418
Drag and drop editing 405
Edit 410
Edit Commands 404
Enable automatic Toolbar help 392
Enable Virtual Space 405
Enums 410
Failure 388
Find and compare operations 395
Find Reference using 392
Font 406
Font preferences 404
Foreground 409
Format Braces 402
Functions 410
Globals 410
Hide non-debugging windows 415
Include file cache 388
Indent Braces 403
Indent Case Within Switch Statement 403
Indent Code Within Braces 403
Insert Template Commands 405
Keywords 409
Language Settings 402
Launch Editor 391
Launch Editor w/ Line # 391
Left margin click selects line 405
Level 394
Macros 410
Maintain files in cache 417
Menu bar layout 391
Minimize non-debugging windows 415
Monitor for debugging 415
Move open windows to debugging monitor
 when debugging starts 416
Name 397
Open windows on debugging monitor during
 debugging 416
Other 410
Place Else On Same Line As Closing
 Brace 403
Place Opening Brace On Separate Line 403
Play sound after 'Bring Up To Date' &
 'Make' 388
Project Commands 405
Project operations 395
Projects 391
Purge Cache 417
Recent symbolics 391
Recommended 390
Regular Expression 395
Relaxed C popup parsing 405
Remote Connection list 419
Remove 396, 397, 419
Save open files before build 388
Script 406

Select stack crawl window when task is stopped 418
Selection position 404
SEt 393
Set 1, Set 2, Set 3, Set 4 410
Shielded folder list 395
Show all locals 413
Show message after building up-to-date project 388
Show tasks in separate windows 416
Show values as decimal instead of hex 414
Show variable location 413
Show variable types 413
Show variable values in source code 414
Size 406
Sort function popup 405
Sort functions by method name in symbolics window 414
Source Tree list 397
Strings 410
Success 388
Tab indents selection 406
Tab Inserts Spaces 407
Tab Size 407
Templates 410
Type 397
TypeDefs 410
Use Automatic Code Formatting 402
Use Concurrent Compiles 390
Use Debugging Monitor 415
Use Default workspace 392
Use External Editor 392
Use Local Project Data Storage 389
Use Multiple Document Interface 391
Use multiple undo 405
Use Script menu 392
Use Third Party Editor 391
Use ToolServer menu 392
User Specified 390
Variable values change 413
VCS Commands 405
Watchpoint indicator 413
Window follows insertion point 400
Window position and size 404
Zoom windows to full screen 391
IDE Preferences window 245, 364, 385, 386
 Apply button 387
Cancel button 387
Factory Settings button 364, 386
IDE Preference Panels list 386
Import Panel 466
Import Panel button 365, 386
OK button 387
Revert Panel button 364, 386
Save button 365, 387
Ignore Extra Space checkbox 150
Ignored By Make File flag 436
Import button 383
Import Commands 382
Import Components menu command 539
Import Panel 466
Import Project command 40
Import Project menu command 539
importing
 projects saved as XML files 40
In Files page 138, 139
In Files tab 131
In Folders page 131, 133
In Folders tab 131
In Projects page 134, 135
In Projects tab 131
In Symbolics page 136, 137
In Symbolics tab 131
Inactive icon 225
Include file cache option
 Build Settings panel 466
Include Files 196
Include files 193
#include files, caching 466
indenting
 text blocks 100
Initial Directory field
 Build Extras panel 466
Initializer 198
Insert Reference Template 121
Insert Reference Template menu command 539
Insert Template Commands option
 Editor Settings panel 466
inserting a reference template 120
inspecting
 project files 38
Installed Products button 523
Instances tab 225
Instruction Scheduling 440

Integrated Development Environment. *See* IDE.

interface files

 locating 113, 114

Interface menu 56

Interfaces list pop-up

 in Files view of Project window 49

 using 114

interfaces list pop-up 92

interrupt time

 and profiler 348

interrupt time, and profiler 348

IP Address field 420

J

Java Exceptions Submenu

 No Exceptions command 541

Java Exceptions submenu

 All Exceptions command 524

 Exceptions In Targeted Classes
 command 533

 Uncaught Exceptions Only command 555

Java submenu 524, 533, 541, 555

K

Key Bindings 363, 366

 Add 380

 Customize 379

key bindings 120

keyboard conventions 22

keyboard shortcuts

 Find symbols with prefix 100

 Find symbols with substring 100

 Get next symbol 100

 Get previous symbol 100

keys

 Quote Key prefix 383

keywords

 adding to a keyword set 441

 removing from a keyword set 442

Keywords option

 Text Colors panel 467

Kill button 206

Kill command 211

Kill menu command 539

killing program execution 211

L

Language Parser option, in Generate Browser

 Data From menu 465

Launch Editor option

 IDE Extras panel 467

Launch Editor w/ Line # option

 IDE Extras panel 468

Launch Remote Host Application option

 Remote Debugging settings panel 468

Launchable flag 436

Layout Editor panel

 options

 Grid Size X 465

 Grid Size Y 465

 Show the component palette when
 opening a form 479

 Show the object inspector when
 opening a form 479

layout management 53

layouts

 moving 54

 removing 53

 renaming 55

least significant bit 306

Left Edges command 539

Left margin click selects line option

 Editor Settings panel 468

Level option

 Plugin Settings panel 468

libraries

 profiler 329

Lifetime Based Register Allocation 440

line

 going to in source code 116

Line And Column button 208

line and column indicator, in editor window 95

%line command-line string 468

Line Display 180

lines, selecting 98

lines, selecting multiple 98

lines, selecting rectangular portions of 98

link maps

 generating for projects 360

Link Order page 51

Link Order tab 53

Link Order view 39, 54, 55

Linker option

Target Settings panel 469
linkers 359
choosing 359
linking projects 360
Linux
modifier key mappings 22
list
of symbols in Browser Contents
window 184
list boxes
Analyzer Type 311
Bit Value Modifier 272
Bitfield Name 272
Connection 288, 300
Connection Type 311
Debugger 311
File Set 139
Format 271
Functions 208
Project 134
Source 208
Symbolics 136
Target 134
Text View 273, 274
list menus
document settings 93
functions 92
interfaces 92
markers 93
VCS 93
list pop-up menus
Current Target 46
list pop-ups
Ancestor 180
Browser Access Filters 169
document settings 93
functions 92
interfaces 92
markers 93
Symbols 184
VCS 170
lists
All Sectors 295
File 134
File Mappings 455
File Set 139
Symbolics 136
Live Range Splitting 439
Load Settings button 287, 298
locating functions 113, 114
locating interface files 113, 114
locating source code 113
Location of Relocated Libraries and Code
Resources option
Debugger Settings panel 469
Log Message checkbox 235
Log Point 234, 235
Log Point Settings window 235
Message text box 235, 236
Speak Message checkbox 236
Stop in Debugger checkbox 236
Treat as Expression checkbox 236
Log Point, clearing 236
Log Point, setting 235
Log System Messages 281
Log System Messages option
Debugger Settings panel 469
Log Window
Log System Messages option 281
Log window 281
opening 282
logic analyzer 309
Arm command 313
arming 313
configuring a project 309
Connect command 312
connecting to 313
Disarm command 313
disarming 313
Disconnect command 314
disconnect from 314
Update Data command 313
updating data from 313
using 312
Logic Analyzer connection options
Analyzer Can Cause Target Breakpoint
checkbox 312
Analyzer Configuration File text box 311
Analyzer Slot text box 312
Analyzer Type list box 311
Connection Type list box 311
Debugger list box 311
Host Name text box 311
Name text box 311

Target Breakpoint Can Cause Analyzer
Trigger checkbox 312
Trace Support File text box 312
Long Word option button 301, 303, 305
`longjmp()` 333
looking up symbol definitions 119
Loop Transformations 440
Loop Unrolling 440
Loop Unrolling (Opt For Speed Only) 440
Loop-Invariant Code Motion 440
LSB 306

M

Mac OS
 QuickHelp 119
 QuickView 119, 120
 THINK Reference 120
Mac OS X API 463
machines, defined 278
Macintosh
 creating files 62
 using an external editor 392
Macintosh menu layout 507
Macro file option, in Generate Browser Data
 From menu 465
Macros option 410
Maintain Files In Cache option 453
Maintain Files in Cache option
 Global Settings panel 470
Make command 55, 56, 57
Make menu command 540
Make option 463
Make toolbar button 46
Makefile Importer wizard 36
makefiles
 converting into projects 36
managing
 build targets 58
 projects 35
 targets 58
managing files, tasks 61
manipulating program execution 221
 Breakpoints window 222
manual conventions 21
markers 116
 adding to a source file 117
 navigating to 118

removing all from source files 118
removing from source files 118
Markers list pop-up 93
Markers list, in Remove Markers window 117
Match Whole Word checkbox 124, 127, 131
matching
 any character with regular expressions 146
 replace strings to find strings with regular
 expressions 147, 148
 with simple regular expressions 146
Maximize Window menu command 540
Maximum Invert Convergence subtest 308
.mcp 37
MDI 391, 470, 487
 and dockable windows 67
 making a window an MDI child 75
Member Function Declaration 196
member functions
 creating 176, 195
 identifier icons 176
Member Functions pane 175
 in Class Browser window 169
memory aliasing, defined 307
memory dump 557
Memory Range On Target option button 296
Memory Read / Write panel 300
memory tests
 Address 307
 Bus Noise 307
 address lines 307
 data lines 308
 Bus Noise test
 Full Range Converging subtest 308
 Maximum Invert Convergence
 subtest 308
 Sequential subtest 307
 Walking Ones 306
 Walking Ones test
 Address Line fault 306
 Data Line fault 306
 Ones Retention subtest 306
 Retention fault 306
 Walking Ones subtest 306
 Walking Zeros subtest 306
 Zeros Retention subtest 306
 Memory Tests panel 303
 Address test 307

Bus Noise test 307
 address lines 307
 data lines 308
Walking Ones test 306
memory usage 344
Memory window 259
memory, working with 259
Menu
 Current Target 377
menu
 Search 119
Menu bar layout option
IDE Extras panel 470
menu commands
 About Metrowerks CodeWarrior 523
 Add Files 523
 Add Window 523
 Apply Difference 154, 524
 Arm 313
 Balance 525
 Bottom Edges 525
 Break 525
 Break On C++ Exception 525
 Break on Java Exceptions 525
 Breakpoints 525
 Breakpoints Window 525
 Bring To Front 525
 Bring Up To Date 526
 Browser Contents 526
 Build Progress 526
 Build Progress Window 526
 Can't Redo 496, 510
 Can't Undo 496, 510
 Cascade 526
 Change Program Counter 526
 Check Syntax 526
 Class Browser 527
 Class Hierarchy 527
 Class Hierarchy Window 527
 Clear 527
 Clear All Breakpoints 527
 Clear All Watchpoints 527
 Clear Breakpoint 527
 Clear Eventpoint 528
 Clear Watchpoint 528
 Close 528
 Close All 528
Close All Editor Documents 528
Close Catalog 529
Close Workspace 529
CodeWarrior Help 529
Collapse Window 529
Commands & Key Bindings 529
Compare Files 150, 529
Compile 530
Complete Code 529
Connect 312, 530
Copy 530
Copy To Expression 530
Create Design 530
Create Group 530
Create Target 530
Cycle View 531
Debug 531
Delete 531
Disable Breakpoint 531
Disable Watchpoint 532
Disarm 313
Disassemble 532
Disconnect 314
Display Grid 532
Enable Breakpoint 532
Enable Watchpoint 532
Enter Find String 144, 533
Enter Replace String 533
Errors And Warnings 533
Errors And Warnings Window 533
Exit 533
Expand Window 534
Export Project 534, 539
Export Project as GNU Makefile 534
Expressions 534
Expressions Window 534
Find 125, 534
Find and Open 'Filename' 536
Find and Open File 536
Find And Replace 536
Find Definition 535
Find Definition & Reference 534
Find In Files 535
Find In Next File 535
Find In Previous File 535
Find Next 143, 535
Find Previous 143, 536

Find Previous Selection 536
Find Reference 536
Find Selection 144, 537
Get Next Completion 537
Get Previous Completion 537
Global Variables 537
Global Variables Window 537
Go Back 537
Go Forward 537
Go To Line 537
Hide Breakpoints 538
Hide Window Toolbar 538
Import Components 539
Import Project 539
Insert Reference Template 121, 539
Kill 539
Make 540
Maximize Window 540
Metrowerks Website 540
Minimize Window 540
New 540
New Class 540
New Class Browser 540
New Data 541
New Event 541
New Event Set 541
New Expression 541
New Member Function 541
New Method 541
New Property 541
New Text File 541
Open 542
Open Recent 542
Open Scripts Folder 542
Open Workspace 542
Page Setup 542
Precompile 543
Preferences 543
Print 543
Processes 543
Processes Window 543
Redo 543
Refresh All Data 544
Register Details Window 270, 544
Register Windows 544
Registers 544
Remove Object Code 544
Remove Object Code & Compact 544
Remove Toolbar Item 377
Replace 128, 545
Replace All 545
Replace and Find Next 545
Resume 547
Revert 547
Run 472, 547
Run To Cursor 548
Save Default Window 548
Save Workspace 548
Save Workspace As 549
Select All 549
Send To Back 549
Set Breakpoint 549
Set Default Project 549
Set Default Target 549
Set Eventpoint 549
Set Watchpoint 550
Shift Right 550
Show Breakpoints 527, 550
Show Types 550
Show Window Toolbar 538
Stack Editor Windows 551
Step Over 551
Stop Build 551
Switch To Monitor 551
Symbolics 551
Symbolics Window 551
Synchronize Modification Dates 551
Unapply Difference 155
Update Data 313
View Array 556
View As Unsigned Decimal 556, 557
View Disassembly 557
View Mixed 558
View Source 558
View Variable 558
Zoom Window 558
menu layouts
 Macintosh 507
 Windows 493
menu reference
 for IDE 493
menus 169
 contextual 212
 VCS 177

Message text box 235, 236
Metrowerks Website command 540
microsecondsTimeBase 331, 345
Minimize non-debugging windows option
 Windowing panel 470
Minimize Window menu command 540
Monitor for debugging option
 Windowing panel 470
most significant bit 306
Move open windows to debugging monitor when
 debugging starts option
 Windowing panel 471
moving
 build targets 54
 dockable windows 77
 files 54
 groups 54
 layouts 54
 targets 54
MSB 306
Multi-Class Hierarchy window 179, 182
multi-core debugging 214
Multiple Document Interface. *See* MDI.
multiple files, searching 139
multiple folders, searching 133
multiple projects, searching 135
multiple Redo 544
multiple symbolics files, searching 137
multiple Undo 544
multiple-file Find and Replace window 129

N

Name field 365
Name text box 311
name, breakpoint property 227
navigating
 browser data 164
 Code Completion window 107
 to markers 118
navigating data 164
navigating source code 113
New Binding 366, 381
New C++ Class window 191
New C++ Data Member window 198
New C++ Member Function window 196
New Class Browser menu command 540
New Class menu command 540

New Class wizard 174, 189, 190
New Command 366
New command 61, 84
New Command Group
 Create 366
New Connection dialog box 420
New Data Member 176, 195, 198
new data member functions
 creating 197
New Data Member wizard 177, 197
New Data menu command 541
New Event menu command 541
New Event Set menu command 541
New Expression menu command 541
New Group 366
New Item 174
New Member Function menu command 541
New Member Function wizard 176, 194, 195
new member functions
 creating 194
New Menu Command
 Create 366, 371
New menu command 540
New Method menu command 541
New Property menu command 541
New Text File command 62
New Text File menu command 541
Next Result button 142
No Exceptions command 541
None option
 of Plugin Diagnostics 468
None option, in Generate Browser Data From
 menu 464
non-modal, defined 69
notes
 for the latest release 17
Numeric Keypad Bindings 381
Numeric Keypad Bindings checkbox
 of Customize IDE Commands window 384

O

object performance 339
Offset text box 293
OK button 287, 298
Ones Retention subtest 306
Only Show Different Files checkbox 150
Open command 62

Open File 177
Open In Windows Explorer command 49
Open menu command 542
Open Recent menu command 542
Open Scripts Folder menu command 542
Open windows on debugging monitor during debugging option
 Windowing panel 471
Open Workspace menu command 542
opening 187
 a recent workspace 82
 a single-class hierarchical window 183
 files 62
 Flash Programmer window 285
 Hardware Diagnostics window 297
 projects 37
 projects from other hosts 38
 subprojects 43
 Symbolics window 277
 symbols window 187
 workspaces 81
opening last project (default workspace) 485
opening last project, preventing (default workspace) 486
openings
 registers in a separate Registers window 270
optimizations
 Arithmetic Optimizations 439
 Branch Optimizations 439
 Common Subexpression Elimination 439
 Copy And Expression Propagation 439
 Copy Propagation 439
 Dead Code Elimination 439
 Dead Store Elimination 439
 Expression Simplification 439
 Global Register Allocation 439
 Global Register Allocation Only For
 Temporary Values 439
 Instruction Scheduling 440
 Lifetime Based Register Allocation 440
 Live Range Splitting 439
 Loop Transformations 440
 Loop Unrolling 440
 Loop Unrolling (Opt For Speed Only) 440
 Loop-Invariant Code Motion 440
 Peephole Optimization 439
 Register Coloring 440
Repeated 440
Strength Reduction 440
Vectorization 440
option buttons
 All text 125, 128, 131
 Byte 301, 302, 304
 Code Only 125, 128, 131
 Comments Only 125, 128, 131
 Entire Flash 296
 File on Host 296
 File on Target 296
 Long Word 301, 303, 305
 Memory Range on Target 296
 Read 301, 302
 Word 301, 303, 305
 Write 301, 302
options 458
 Access Paths settings panel 394, 427
 Activate Browser 535
 Activate Browser Coloring 449
 Activate Syntax Coloring 449, 455
 Add Default 449
 Always Search User Paths 450
 Application 450
 Arguments 450
 Attempt to use dynamic type of C++, Object Pascal and SOM objects 450
 Auto Indent 450
 Auto Target Libraries 450
 Automatic Invocation 451
 Automatically Launch Applications When SYM File Opened 451
 Auto-target Libraries 450
 Background 452
 Balance Flash Delay 452
 Balance while typing 452
 Bring Up To Date 463
 Browse in processes window 420, 421
 Browser Commands 452
 Browser Path 453
 Build before running 453
 Build Extras settings panel 430
 Build Settings preference panel 387
 Cache Edited Files Between Debug Sessions 453
 Cache Subprojects 453
 Cache symbolics between runs 454

Case Sensitive 454
Checksum panel 295
choosing host application for non-executable files 465
Classes 410
Close non-debugging windows 454
Code Completion Delay 454
Code Completion preference panel 400
Code Formatting preference panel 401
Collapse non-debugging windows 455
Comments 455
Compiler 455
Compiler thread stack 455
Concurrent Compiles preference panel 389
Configuration panel 299
Confirm “Kill Process” when closing or quitting 456
Confirm invalid file modification dates when debugging 455
Connection Type 420, 421
Constants 410
Context popup delay 456
Debugger Commands 456
Debugger preference panels 412
Debugger Settings 281
Debugger Settings panel 445
Default File Format 456
Default size for unbounded arrays 457
Disable third party COM plugins 457
Display Deprecated Items 457
Display Settings preference panel 412
Do nothing 457
Do nothing to project windows 457
Drag and drop editing 458
Dump internal browse information after compile 458
Edit Commands 458
Edit Language 458
Editor preference panels 399
Editor Settings preference panel 403
Enable automatic Toolbar help 459
Enable remote debugging 459
Enable Virtual Space 459
Enums 410
Environment Settings 459
Erase / Blank Check panel 294
Failure 463
File Mappings settings panel 434
File Type 455
Flash Configuration panel 289
Font & Tabs preference panel 406, 408
Functions 410
General preference panels 387
Generate Browser Data From 464
Global Optimizations settings panel 437
Globals 410
Help Preferences panel 393
IDE Extras preference panel 390
Import Panel 466
Macros 410
Maintain files in cache 453
Make 463
Memory Read / Write panel 300
Memory Tests panel 303
Other 410
Other Executables settings panel 443
Plugin Settings preference panel 394
Program / Verify panel 291
Purge Cache 453
Remote Connections preference panel 418
Remote Debugging settings panel 447
Require Framework Style Includes 474
Runtime Settings panel 432
Scope Loop panel 301
Set 1, Set 2, Set 3, Set 4 410
setting for browser 161
Shielded Folders preference panel 394
Source Trees preference panel 396
Target Configuration panel 287
Target Settings panel 426
Templates 410
TypeDefs 410
Use Multiple Document Interface 67
User specified 446
Window Follows Insertion Point 490
Window Settings preference panel 414
Organization pane 291
original process, breakpoint property 227
original-target, breakpoint property 228
other editor windows 94
Other Executables settings panel 443
Other option 410
Output Directory option
Target Settings panel 471

- Overlays tab 53
overstrike 99
overstriking text (Windows) 99
overtype. *See* overstrike.
overview
 of browser 27
 of build system 27
 of CodeWarrior 23
 of debugger 27
 of editor 27
 of IDE project manager and build targets 31
 of IDE tools 26
 of IDE User's Guide 17
 of project manager 27
 of search engine 27
- P**
- Page Setup command 542
pages
 In Files 138
 In Folders 131
 in project window 47
 In Projects 134
 In Symbolics 136
PalmQuest reference 463
Pane Collapse 173
Pane Collapse box 156, 206
Pane Expand 173
Pane Expand box 156, 206
Pane resize bar 142, 153, 157, 207
pane resize bar
 in File Compare Results window 153
 in Folder Compare Results window 157
pane splitter controls, in editor window 95
panel
 Display Settings 245
panels
 Analyzer Connections 309
 Font & Tabs 407
panes
 adding to editor window 95
 Destination 153
 Device 290
 Differences 154
 Files in Both Folders 157
 Files Only in Destination 157
 Files Only in Source 157
- Flash Programmer 286
Hardware Diagnostics 298
Organization 291
removing from editor window 96
resizing in an editor window 95
Results 142
Sector Address Map 291
Source 153, 207
Source Code 142
Stack 207
Variables 207
parameter lists
 completing code 109
Passes text box 305
path caption 94
Pause Point 234, 236
Pause Point, clearing 237
Pause Point, setting 236
Peephole Optimization 439
Play sound after 'Bring Up To Date' & 'Make' option
 Build Settings panel 471
Plugin Diagnostics
 All Info option 469
 Errors Only option 469
 None option 468
plug-in diagnostics
 disabling 523
 enabling 523
Plugin Settings panel
 options
 Level 468
Plugin Settings preference panel 394
 options
 Disable third party COM plugins 394
 Level 394
plug-ins
 saving information about those installed in IDE 523
 viewing those installed in IDE 523
pop-up menus
 document settings 93
 functions 92
 interfaces 92
 markers 93
 VCS 93
pop-ups

Ancestor 180
Browser Access Filters 169
Symbols 184
VCS 170
Post-linker option
 Target Settings panel 472
PPCTimeBase 331, 345
Precompile menu command 543
Precompiled File flag 436
precompiled headers
 caching 466
preference panels
 Build Settings 387
 Code Completion 400
 Code Formatting 401
 Concurrent Compiles 389
 Display Settings 412
 Editor Settings 403
 Font & Tabs 406, 408
 Help Preferences 393
 IDE Extras 390
 Plugin Settings 394
 Remote Connections 418
 reverting 474
 Shielded Folders 394
 Source Trees 396
 Window Settings 414
preferences
 Activate Browser Coloring 410
 Activate Syntax Coloring 409
 Add 396, 397, 419
 Apply button 387
 Attempt to use dynamic type of C++, Object
 Pascal and SOM objects 414
 Auto Indent 407
 Auto Target Libraries 418
 Automatic Invocation 400
 Automatically launch applications when
 SYM file opened 417
 Background 409
 Balance Flash Delay 405
 Balance while typing 405
 Browser Commands 405
 Browser Path 393
 Build before running 388
 Cache Edited Files Between Debug
 Sessions 417
 Cancel button 387
 Case sensitive 401
 Change 396, 397, 419
 Choose 397
 Classes 410
 Close Braces, Brackets, And
 Parentheses 403
 Close non-debugging windows 415
 Code Completion Delay 401
 Collapse non-debugging windows 415
 Comments 409
 Compiler thread stack 389
 Confirm "Kill Process" when closing or
 quitting 417
 Confirm invalid file modification dates when
 debugging 417
 Constants 410
 Context popup delay 391
 Debugger 412
 Debugger Commands 405
 Default file format 405
 Default size for unbounded arrays 414
 Disable third party COM plugins 394
 Display deprecated items 401
 Do nothing 415
 Do nothing to project windows 415
 Documents 391
 Don't step into runtime support code 418
 Drag and drop editing 405
 Edit 410
 Edit Commands 404
 Editor 399
 Enable automatic Toolbar help 392
 Enable Virtual Space 405
 Enums 410
 Export Panel button 364, 386
 Factory Settings button 364, 386
 Failure 388
 Find and compare operations 395
 Find Reference using 392
 Font 406
 Font preferences 404
 for IDE 385
 Foreground 409
 Format Braces 402
 Functions 410
 General 387

Globals 410
Hide non-debugging windows 415
IDE Preference Panels list 386
IDE window 385
Import Panel button 365, 386
Include file cache 388
Indent Braces 403
Indent Case Within Switch Statement 403
Indent Code Within Braces 403
Insert Template Commands 405
Keywords 409
Language Settings 402
Launch Editor 391
Launch Editor w/ Line # 391
Left margin click selects line 405
Level 394
Macros 410
Maintain files in cache 417
Menu bar layout 391
Minimize non-debugging windows 415
Monitor for debugging 415
Move open windows to debugging monitor
when debugging starts 416
Name 397
OK button 387
Open windows on debugging monitor during
debugging 416
Other 410
Place Else On Same Line As Closing
Brace 403
Place Opening Brace On Separate Line 403
Play sound after ‘Bring Up To Date’ &
‘Make’ 388
Project Commands 405
Project operations 395
Projects 391
Purge Cache 417
Recent symbolics 391
Recommended 390
Regular Expression 395
Relaxed C popup parsing 405
Remote Connection list 419
Remove 396, 397, 419
Revert Panel button 364, 386
Save button 365, 387
Save open files before build 388
Script 406
Select stack crawl window when task is
stopped 418
Selection position 404
Set 393
Set 1, Set 2, Set 3, Set 4 410
Shielded folder list 395
Show all locals 413
Show message after building up-to-date
project 388
Show tasks in separate window 416
Show values as decimal instead of hex 414
Show variable location 413
Show variable types 413
Show variable values in source code 414
Size 406
Sort function popup 405
Sort functions by method name in symbolics
window 414
Source Tree list 397
Strings 410
Success 388
Tab indents selection 406
Tab Inserts Spaces 407
Tab Size 407
Templates 410
Type 397
TypeDefs 410
Use Automatic Code Formatting 402
Use Concurrent Compiles 390
Use Debugging Monitor 415
Use Default workspace 392
Use External Editor 392
Use Local Project Data Storage 389
Use Multiple Document Interface 391
Use multiple undo 405
Use Script menu 392
Use Third Party Editor 391
Use ToolServer menu 392
User Specified 390
Variable values change 413
VCS Commands 405
Watchpoint indicator 413
Window follows insertion point 400
Window position and size 404
Zoom windows to full screen 391
Preferences menu command 543

Prefix file option, in Generate Browser Data From menu 465
prefix keys
 Quote Key 383
Pre-linker option
 Target Settings panel 472
preprocessor directives 330
 C/C++ 343
Previous Result button 142
print
 file selections 66
Print command 66, 543
printing
 class hierarchies 181
 files 66
 projects 39
process
 attaching debugger to 281
process cycle
 of software development 23
processes
 related to machines 278
 viewing global variables for 250
Processes menu command 543
Processes window 278, 420
 opening 278, 280
Processes Window menu command 543
products
 saving information about those installed in IDE 523
 viewing those installed in IDE 523
Profile window
 opening 316
profiler
 libraries 329
Profiler Function Reference 345
 ProfilerClear() 349
 ProfilerDump() 349
 ProfilerGetDataSizes() 348
 ProfilerGetStatus() 348
 ProfilerInit() 346
 ProfilerSetStatus() 347
 ProfilerTerm() 347
ProfilerClear() 349
ProfilerDump() 334, 349
ProfilerGetDataSizes() 348
ProfilerGetStatus() 348
ProfilerInit() 346
ProfilerSetStatus() 347
ProfilerTerm() 347
ProfilerClear() 349
ProfilerDump() 334, 349
ProfilerGetDataSizes() 348
ProfilerGetStatus() 348
ProfilerInit() 346
ProfilerSetStatus() 347
ProfilerTerm() 347
ProfilerClear() 349
ProfilerDump() 334, 349
ProfilerGetDataSizes() 348
ProfilerGetStatus() 348
ProfilerInit() 346
ProfilerSetStatus() 347
ProfilerTerm() 347
Profiling
 activating 325
 by function 326, 343
 exceptions 333
 inline functions 342
 setjmp() 333
program
 killing execution 211
 resuming execution 210
 running 211
 stopping execution 210
Program / Verify panel 291
Program Arguments field
 of Runtime Settings panel (Windows) 472
Program Arguments option
 Runtime Settings panel 472
Program button 294
Program Entry Point option
 Debugger Settings panel 472
program execution, manipulating 221
project
 configuring for a logic analyzer 309
Project Commands option
 Editor Settings panel 472
project data folder 486
Project Headers checkbox 134
Project Inspector command 38
Project list box 134
project manager 31
 overview 27
Project menu 472, 500, 513
 Remove Object Code command 501, 515
 Stop Build command 501, 514
Project operations option
 Shielded Folders panel 473
Project Settings 325
Project Sources checkbox 134
project stationery
 creating 41
 custom 41
Project window
 about Files page 47

Current Target list pop-up 46
Files view
 Checkout Status column 48
 Code column 48
 Data column 48
 Debug column 48
 File column 48
 Interfaces list pop-up 49
 Sort Order button 49
 Target column 48
 Touch column 48
Make toolbar button 46
Synchronize Modification Dates toolbar
 button 46
Target Settings toolbar button 46
project window 45
 Link Order page 51
 pages 47
 Targets page 52
project window, about 45
project, defined 31
projects
 about subprojects 42
 advanced topics 41
 choosing default 39
 closing 40
 creating custom stationery 41
 creating empty 37
 creating subprojects 42
 creating using makefiles 36
 creating using stationery 35
 data folder 486
 exporting to XML files 40
 generating link maps for 360
 importing XML versions of 40
 inspecting files 38
 linking 360
 managing 35
 opening 37
 opening from other hosts 38
 printing 39
 project window 45
 project window pages 47
 project window, about 45
 reopening last one used (default
 workspace) 485
reopening last one used, preventing (default
 workspace) 486
saving 38
searching (multiple) 135
strategies for 43
subprojects, strategies for 43
working with 31
Projects option
 IDE Extras panel 472
properties
 condition, breakpoint 227
 file-info, breakpoint 227
 hardware, breakpoint 228
 hit count, breakpoint 227
 name, breakpoint 227
 original process, breakpoint 227
 original-target, breakpoint 228
 serial number, breakpoint 227
 thread, breakpoint 228
 times hit, breakpoint 228
 times left, breakpoint 228
 type, breakpoint 227
punctuation balancing, toggling 101
punctuation, balancing 100, 101
pure virtual
 icon for 176
Purge Cache button 473
Purge Cache option 453
purging cache 473
purpose
 of breakpoints 221
 of Browser Contents window 183
 of Classes pane in browser 174
 of Data Members pane 176
 of eventpoints 221
 of Member functions pane 175
 of Multi-Class Hierarchy window 179
 of Single-Class Hierarchy window 182
 of Source pane 177
 of special breakpoints 221
 of status area in browser 178
 of Symbols window 185
 of watchpoints 221

Q

QuickDraw 470
QuickHelp (Mac OS) 119

QuickView 119, 120, 463

QuickView, Mac OS 120

QuickView, THINK Reference 120

Quote Key prefix 383

 assigning 384

R

Read button 273

Read option button 301, 302

Recursive Search column, in Access Paths panel 430

Redo button 154

Redo menu command 543

reference information

 for IDE menus 493

reference template 120

reference template, inserting 120

reference templates (Macintosh) 120

Refresh All Data menu command 544

REGISTER 561

Register Coloring 440

Register Description option

 of Text View pop-up menu 274

Register Details option

 of Text View pop-up menu 274

Register Details window 270

 Address text box 271

 Bit Value Modifier list box 272

 Bit Value text box 272

 Bitfield Description text view option 274

 Bitfield Name list box 272

 Browse button 271, 273

 Description 272

 Description File text box 271, 273

 Format list box 271

 Read button 273

 Register Description text view option 274

 Register Details text view option 274

 Register display 272, 274

 Register Name 271

 Reset Value button 273

 Revert button 273

 Text View list box 273, 274

 using 273

 Write button 273

XML

 file locations 568

sample files 569

specification 559

Register Details Window command 270

Register Details Window menu command 544

Register display 272, 274

Register Name 271

Register Windows menu command 544

registers

 changing data views of 269

 changing values of 268

 FPU Registers 267

 General Registers 267

 host-specific 267

 Register Details window 270

 viewing 268

 viewing details of 270

Registers folder 270

Registers menu command 544

Registers window 266

 opening 268

 opening more than one 270

Registry Key option

 of Source Trees preference panel 485

Registry Key option, in Type pop-up menu 485

regular breakpoints 222

Regular Expression checkbox 124, 127, 131

Regular Expression option

 Shielded Folders panel 473

regular expressions 145

 .*[_]Data 396

 \(.*\) 396

 choosing one character from many 147

 CVS 396

 defined 145

 grouping 147

 matching any character 146

 matching simple expressions 146

 using the find string in the replace

 string 147, 148

Relative to class field 191

Relaxed C popup parsing option

 Editor Settings panel 473

release notes 17

remembering last project (default

 workspace) 485

remembering last project, turning off (default

 workspace) 486

remote connections
 adding 420
 changing 420
 removing 421
Remote Connections preference panel 418
options
 Add 419
 Change 419
 Remote Connection list 419
 Remove 419
Remote Debugging settings panel 447
 Connection pop-up menu 447
options
 Launch remote host application 468
Remove A Set button 139
Remove button 429
Remove button, in Remove Markers window 117
Remove command 53
Remove Markers window 117
 Cancel button 117
 Done button 117
 Markers list 117
 Remove button 117
Remove Object Code & Compact menu
 command 544
Remove Object Code menu command 544
Remove Toolbar Item 377
removing
 build targets 53, 59
 desktop background from behind IDE. *See*
 Use Multiple Document Interface, turning
 on.
 files 53
 gray background from behind IDE. *See* Use
 Multiple Document Interface, turning off.
 groups 53
 layouts 53
 remote connections 421
 source trees 399
 targets 53, 59
Rename Breakpoint button 224
Rename command 55, 60
renaming
 build targets 56, 60
 files 55
 groups 55
 layouts 55
targets 55, 60
reopening last project used
 in default workspace 485
 suppressing in the default workspace 486
Repeated optimizations 440
Replace All button 127, 130
Replace All menu command 545
Replace and Find Next menu command 545
Replace and Find Previous command 545
Replace button 127, 130
Replace command 128
Replace menu command 545
Replace With text/list box 127, 130
replacing
 text in a single file 128
 text, overview 123
Require Framework Style Includes 474
Reset Value button 273
Reset Window Toolbar command in Toolbar
 submenu 51, 546
resetting
 toolbars 378
resize bars
 Pane 142, 207
Resize submenu
 To Smallest Height command 554
 To Smallest Width command 555
resizing
 panes in an editor window 95
Resource File flag 435
Restart command 211
restarting
 debugger 211
Restore Window command (Windows) 547
Restrict Address Range checkbox 293
Result Count text box 142
results
 finding problems 340
 of multi-item search 141
 opening 335
 sorting 337
Results pane 142
Resume button 206
Resume command 210
Resume menu command 547
resuming program execution 210
Retention fault 306

Revert button 273
Revert command 66
Revert menu command 547
reverting
 files 66
 preference panels 474
 settings panels 474
revision control 489, 521
routine
 stepping into 209
 stepping out of 209
 stepping over 210
routine, selecting entirely 98
Run App/Script 372
Run button 206
Run command 56, 57, 211
Run menu command 472, 547
Run To Cursor menu command 548
running
 a program 211
Runtime Settings panel 432
 Host Application For Libraries And Code
 Resources field 465
 options
 Add 433
 Change 434
 Environment Settings 433
 Host Application for Libraries & Code
 Resources 433, 465
 Program Arguments 433, 472
 Remove 434
 Value 434
 Variable 434
 Working Directory 433, 490
 Program Arguments field (Windows) 472

S

Save a Copy As command 64
Save All command 64
Save command 63
Save Default Window menu command 548
Save open files before build option
 Build Settings panel 474
Save project entries using relative paths option
 Target Settings panel 50, 475
Save Settings button 287, 298
Save This Set button 139
Save Workspace As menu command 549
Save Workspace menu command 548
saving
 a copy of a workspace 81
 all files 64
 file copies 64
 files 63
 information about installed plug-ins 523
 information about installed products 523
 projects 38
 workspaces 80
Scope Loop panel 301
Script Point 234, 237
Script Point Settings window
 Stop in Debugger checkbox 238
Script Point, clearing 238
Script Point, setting 237
 (Scripts) folder 522, 542
Scripts menu 522
Scripts option
 Font & Tabs panel 475
search
 single characters with regular
 expressions 146
 using finds strings in replace strings with
 regular expressions 148
Search Cached Sub-Targets checkbox 134
Search Criteria text box 142
search engine
 overview 27
Search In text/list box 132
Search menu 119, 498, 511
Search Results window 141
 Next Result button 142
 Pane resize bar 142
 Previous Result button 142
 Result Count text box 142
 Results pane 142
 Search Criteria text box 142
 setting default size and position of 548
 Source Code pane 142
 Source Code Pane disclosure triangle 142
 Stop button 142
 Warnings button 142
Search Selection Only checkbox 125, 128
Search Status column, in Access Paths panel 429
Search Sub-Folders checkbox 132

Search Up checkbox 125, 128
searching
choosing one character from many in regular expressions 147
grouping regular expressions 147
multiple files 139
multiple folders 133
multiple projects 135
multiple symbolics files 137
single characters with regular expressions 146
single files 125
using finds strings in replace strings with regular expressions 147
using regular expressions 145
with simple regular expressions 146
Sector Address Map pane 291
seeing desktop background behind IDE. *See Use Multiple Document Interface*, turning off.
Segments tab 53
Select All menu command 549
Select stack crawl window when task is stopped option
 Global Settings panel 475
Selected Item group 157
selecting
 Code Completion window items 108
 text in editor windows 97
selecting entire routines 98
selecting lines 98
selecting multiple lines 98
selecting rectangular portions of lines 98
Selection position option
 Editor Settings panel 476
selections
 searching (text) 144
Send To Back menu command 549
Sequential subtest 307
serial number, breakpoint property 227
Set 1, Set 2, Set 3, Set 4 410
Set Breakpoint menu command 549
Set Default Breakpoint Template button 224
Set Default Project command 40
Set Default Project menu command 549
Set Default Target menu command 549
Set Eventpoint menu command 549
Set Watchpoint menu command 550
setjmp() 333
setting
 browser options 161
 temporary breakpoints 229
setting access breakpoint 245
setting default size and position of windows 548
settings
 Add 397, 429, 433, 436
 Add Default 429
 Always Search User Paths 428
 Application 432
 Apply button 425
 Arguments 432
 Auto-target Libraries 446
 Cache subprojects 431
 Cache symbolics between runs 446
 Cancel button 425
 Change 397, 429, 434, 436
 Choose 397, 427
 Clear 427
 Compiler 436
 Default language entry point 446
 Details 438
 Dump internal browse information after compile 432
 Edit Language 436
 Environment Settings 433
 Export Panel button 425
 Extension 435
 Factory Settings button 425
 Faster Execution Speed 438
 File Mappings list 435
 File Type 435
 Flags 436
 Generate Browser Data From 432
 Host Application for Libraries & Code Resources 433
 Host Flags 429
 IDE window 423
 Ignored By Make flag 436
 Import Panel button 425
 Initial directory 432
 Interpret DOS and Unix Paths 429
 Launchable flag 436
 Linker 427
 Log System Messages 446
 Name 397

OK button 425
Optimization Level slider 438
Output Directory 427
Post-linker 427
Precompiled File flag 436
Pre-linker 427
Program Arguments 433
Program entry point 446
Remove 397, 429, 434, 436
Require Framework Style Includes 429
Resource File flag 435
Revert Panel button 425
Save button 425
Save project entries using relative paths 427
Smaller Code Size 438
Source Tree list 397
Stop at Watchpoints 446
Stop on application launch 446
Target Name 427
Target Settings Panels list 425
Type 397
Update data every *n* seconds 446
Use External Debugger 432
Use modification date caching 431
User specified 446
Value 434
Variable 434
Working Directory 433
settings panels
Access Paths 394, 427
Build Extras 430, 535
Debugger Settings 281, 445
File Mappings 434
Global Optimizations 437
Other Executables 443
Remote Debugging 447
reverting 474
Runtime Settings 432
Source Trees 396
Target Settings 426
setup
code completion 103
Shielded Folders panel
options
Find and compare operations 463
Project operations 473
Regular Expression 473
Shielded Folders preference panel 394
options
Add 396
Change 396
Find and compare operations 395
Project operations 395
Regular Expression 395
Remove 396
Shielded folder list 395
Shift Right menu command 550
shortcut conventions 22
Show all locals option
Display Settings panel 476
Show Breakpoints menu command 527, 550
Show Classes 175
Show Classes pane 178
Show Floating Toolbar command 538
Show Floating Toolbar command in Toolbar submenu 550
Show Inherited 169
Show Log button 286, 305
Show Main Toolbar command 538
Show message after building up-to-date project option
Build Settings panel 478
Show private 170
Show protected 170
Show public 170
Show tasks in separate windows option
Display Settings panel 479
Show the component palette when opening a form option
Layout Editor panel 479
Show the object inspector when opening a form option
Layout Editor panel 479
Show Types menu command 550
Show values as decimal instead of hex option
Display Settings panel 479
Show variable location option
Display Settings panel 479
Show variable types option
Display Settings panel 480
Show variable values in source code option
Display Settings panel 480
Show Window Toolbar command 538

Show Window Toolbar command in Toolbar
submenu 550

showing

- classes pane 175

shrinking panes, in browser 173

Single Class Hierarchy Window 169

single files, searching 125

single-class hierarchical window

- opening 183

Single-Class Hierarchy window 182

- difference from Multi-Class Hierarchy window 182

single-file Find and Replace window 126

single-file Find window 123

size

- setting default for unbounded arrays 457

Size option

- Font & Tabs panel 480

Size text box 297

Skip Point 234, 238

Skip Point, clearing 239

Skip Point, setting 239

software

- development process cycle 23

Solaris

- modifier key mappings 22

Sort Alphabetical 174, 175

Sort function popup option

- Editor Settings panel 483

Sort functions by method name in symbolics window option

- Display Settings panel 480

Sort Hierarchical 174, 175

Sort Order button

- in Files view of Project window 49

sorting

- classes list 175
- Functions list pop-up (alphabetically) 115

sorting data 337

Sound Point 234, 239

Sound Point Settings window

- Stop in Debugger checkbox 240

Sound Point, clearing 240

Sound Point, setting 239

Sound Point, Speak Message 239

Source box 149

source code

disabling breakpoints 228

disabling eventpoints 242

disabling special breakpoints 248

disabling watchpoints 246

editing 97

enabling breakpoints 228, 242

enabling special breakpoints 248

enabling watchpoints 246

going to a particular line 116

locating 113

setting breakpoints in 226

setting watchpoints in 245

viewing breakpoint properties 227

viewing eventpoint properties 242

viewing watchpoint properties 246

Source Code pane 142

Source Code Pane disclosure triangle 142

source code, navigating 113

source file

- adding markers to 117

Source File button 207

source files

- removing all markers from 118
- removing markers from 118

source item, for comparison 148

Source list box 208

Source pane 153, 177, 207

- in Symbols window 187

Source Pane disclosure triangle 207

source relative includes 483

source trees

- adding 398
- changing 398
- removing 399

Source Trees panel

options

- Add 397
- Change 397
- Choose 397
- Name 397
- Remove 397
- Source Tree list 397
- Type 397, 484

Source Trees preference panel 396

- Absolute Path option 484
- Environment Variable option 484
- Registry Key option 485

Source Trees settings panel 396
Speak Message checkbox 236
special breakpoints
 defined 248
 purpose of 221
special breakpoints, disabling 248
special breakpoints, enabling 248
Speed slider 303
Stack Editor Windows menu command 551
Stack pane 207
stack space, finding problems 340
Start text box 293, 297, 304
starting
 debugger 208
state
 disabled, for breakpoints 222, 244
 disabled, for eventpoints 234
 enabled, for breakpoints 222
 enabled, for eventpoints 234
 enabled, for watchpoints 244
static
 icon for 176
stationery
 creating for projects 41
 creating projects 35
 custom 41
Status 293, 295, 297, 301, 303, 305
Status area
 in Class Browser window 169
status area 178
Step Into button 206
Step Into command 209
Step Out button 206
Step Out command 209
Step Over button 206
Step Over command 210
Step Over menu command 551
stepping into a routine 209
stepping out of a routine 209
stepping over a routine 210
Stop At End Of File checkbox 125, 127
Stop at Watchpoints option
 Debugger Settings panel 480
Stop Build menu command 551
Stop button 130, 142, 206
Stop command 210, 551
Stop in Debugger checkbox 236, 238, 240
Stop On Application Launch option
 Debugger Settings panel 481
stopping program execution 210
Straight Line 182
strategies
 for build targets 43
 for projects 43
 for subprojects 43
Strength Reduction 440
Strings option
 Text Colors panel 481
structure
 of documentation 20
submenus
 Align 524, 525
subproject, defined 42
subprojects
 creating 42
 opening 43
 strategies for 43
Success option
 Build Settings panel 483
summary data 331
summation, of two variables 256
Switch To Monitor menu command 551
symbol definitions 119
symbol definitions, looking up 119
Symbol hint 211
symbol hint
 toggling 212
 turning off 212
 turning on 212
 using 212
symbol implementations
 viewing all 187
symbol-editing shortcuts 100
Symbolics button 206
symbolics file, defined 204
symbolics files
 choosing a debugger for 420
 searching (multiple) 137
Symbolics list 136
Symbolics list box 136
Symbolics menu command 551
Symbolics option
 IDE Extras panel 483
Symbolics window 275

opening 277
using the Executables pane 277, 278
using the Files pane 277
 using the Functions pane 278

Symbolics Window menu command 551

symbols
 shortcuts for editing 100
 viewing all implementations 187

Symbols list
 in Browser Contents window 184

Symbols pane 187

Symbols pop-up 184

Symbols window 185
 Source pane 187
 Symbols pane 187
 toolbar 187

symbols window 187

Synchronize Modification Dates command 52

Synchronize Modification Dates menu
 command 551

Synchronize Modification Dates toolbar
 button 46

System Headers checkbox 134

System Paths list
 Framework column 430
 Recursive Search column 430
 Search Status column 429

System Paths option
 Access Paths panel 483

T

Tab indents selection option
 Font & Tabs panel 484

Tab Inserts Spaces option
 Font & Tabs panel 484

Tab Size option
 Font & Tabs panel 484

tabs
 Groups 224
 In Files 131
 In Folders 131
 In Projects 131
 In Symbolics 131
 Instances 225
 Templates 225

Target Address text box 301, 303

Target Breakpoint Can Cause Analyzer Trigger checkbox 312

Target column
 in Files view of Project window 48

Target Configuration panel 287

Target list box 134

target management 53

Target Memory Buffer Address text box 288

Target Memory Buffer Size text box 288

Target Name option
 Target Settings panel 484

Target Processor text/list box 288, 300

Target Scratch Memory End text box 305

Target Scratch Memory Start text box 305

target settings
 Add 397, 429, 433, 436
 Add Default 429
 Always Search User Paths 428
 Application 432
 Apply button 425
 Arguments 432
 Auto-target Libraries 446
 Cache subprojects 431
 Cache symbolics between runs 446
 Cancel button 425
 Change 397, 429, 434, 436
 Choose 397, 427
 Clear 427
 Compiler 436
 Connection pop-up menu 447
 Default language entry point 446
 Details 438
 Dump internal browse information after compile 432
 Edit Language 436
 Environment Settings 433
 Export Panel button 425
 Extension 435
 Factory Settings button 425
 Faster Execution Speed 438
 File Mappings list 435
 File Type 435
 Flags 436
 for IDE 423
 Generate Browser Data From 432
 Host Application for Libraries & Code Resources 433

Host Flags 429
Ignored By Make flag 436
Import Panel button 425
Initial directory 432
Interpret DOS and Unix Paths 429
Launchable flag 436
Linker 427
Log System Messages 446
Name 397
OK button 425
Optimization Level slider 438
Output Directory 427
Post-linker 427
Precompiled File flag 436
Pre-linker 427
Program Arguments 433
Program entry point 446
Remove 397, 429, 434, 436
Require Framework Style Includes 429
Resource File flag 435
Revert Panel button 425
Save button 425
Save project entries using relative paths 427
Smaller Code Size 438
Source Tree list 397
Source Trees 396
Stop at Watchpoints 446
Stop on application launch 446
Target Name 427
Target Settings Panels list 425
Type 397
Update data every *n* seconds 446
Use External Debugger 432
Use modification date caching 431
User specified 446
Value 434
Variable 434
Working Directory 433
Target Settings command 552
Target Settings panel 60, 426
options
 Choose 427
 Clear 427
 Linker 427, 469
 Output Directory 427, 471
 Post-linker 427, 472
 Pre-linker 427, 472
Save project entries using relative paths 50, 427, 475
Target Name 427, 484
target settings panels
 Access Paths 427
 Analyzer Connections 309
 Build Extras 430, 535
 Debugger Settings 281, 445
 File Mappings 434
 Global Optimizations 437
 Other Executables 443
 Remote Debugging 447
 Runtime Settings 432
 Target Settings 426
 Target Settings Panels list 425
 Target Settings toolbar button 46
 Target Settings window 423
 Apply button 425
 Cancel button 425
 Export Panel button 425
 Factory Settings button 425
 Import Panel button 425
 OK button 425
 opening 425
 Revert Panel button 425
 Save button 425
 Target Settings Panels list 425
targets 33
 configuring 60
 creating 58
 files 53
 managing 58
 moving 54
 removing 53, 59
 renaming 55, 56, 60
 setting default 59
 strategies for 43
Targets page 52
Targets tab 60
Targets view 39, 54, 58
tasks
 activating automatic code completion 103
 adding a constant to a variable 256
 adding a keyword to a keyword set 441
 adding an executable file 444
 adding expressions (Expressions window) 255

adding markers to a source file 117
adding panes to an editor window 95
adding remote connections 420
adding source trees 398
adding subprojects to a project 42
alphabetizing Functions list pop-up
 order 115
applying file differences 154
arming a logic analyzer 313
attaching the debugger to a process 281
balancing punctuation 101
changing an executable file 444
changing line views in a hierarchical
 window 182
changing register data views 269
changing register values 268
changing remote connections 420
changing source trees 398
changing the find string 144
choosing a default project 39
choosing files to compare 150
choosing folders to compare 151
clearing a breakpoint 229
clearing a Log Point 236
clearing a Pause Point 237
clearing a Script Point 238
clearing a Skip Point 239
clearing a Sound Point 240
clearing a Trace Collection Off
 eventpoint 241
clearing a Trace Collection On
 eventpoint 241
clearing a watchpoint 247
clearing all breakpoints 229
clearing all watchpoints 247
closing a docked window 77
closing a workspace 81
closing projects 40
collapsing a docked window 76
collapsing browser panes 173
collapsing the editor window toolbar 92
completing code for data members 109
completing code for parameter lists 109
connecting to a logic analyzer 313
creating a breakpoint template 231
creating a console application 84
creating a new class 174, 189, 190
creating a new data member 177, 197
creating a new member function 176, 194,
 195
creating custom project stationery 41
creating empty projects 37
creating new projects from makefiles 36
creating new projects using project
 stationery 35
deactivating automatic code completion 105
deleting a breakpoint template 232
disabling a breakpoint 228
disabling a watchpoint 246
disabling an eventpoint 242
disarming a logic analyzer 313
disconnecting from a logic analyzer 314
docking a window by using a contextual
 menu 70
docking a window by using drag and
 drop 71
docking windows of the same kind 72
enabling a breakpoint 228, 242
enabling a watchpoint 246
examining items in the Folder Compare
 Results window 157
expanding a docked window 76
expanding browser panes 173
expanding the editor window toolbar 92
exporting projects to XML files 40
floating a window 74
for managing files 61
generating project link maps 360
going to a particular line 116
hiding the classes pane 175
importing projects saved as XML files 40
indenting text blocks 100
inserting a reference template 120
issuing command lines 318
killing program execution 211
looking up symbol definitions 119
making a summation of two variables 256
making a window an MDI child 75
manipulating variable formats 252
moving a docked window 77
navigating browser data 164
navigating Code Completion window 107
navigating to a marker 118
opening a recent workspace 82

opening a single-class hierarchical window 183
opening a workspace 81
opening an Array window 266
opening projects 37
opening projects created on other hosts 38
opening registers in a separate Registers window 270
opening subprojects 43
opening the Breakpoints window 225
opening the Cache window 315
opening the Command window 318
opening the Expressions window 255
opening the Flash Programmer window 285
opening the Global Variables window 250
opening the Hardware Diagnostics window 297
opening the Log window 282
opening the Processes window 278, 280
opening the Profile window 316
opening the Registers window 268
opening the Symbolics window 277
opening the symbols window 187
opening the Target Settings window 425
opening the Trace window 314
overstriking text (Windows) 99
printing class hierarchies 181
printing projects 39
removing a keyword from a keyword set 442
removing a marker from a source file 118
removing all markers from a source file 118
removing an executable file 445
removing panes from an editor window 96
removing remote connections 421
removing source trees 399
replacing text in a single file 128
resizing panes in an editor window 95
restarting the debugger 211
resuming program execution 210
running a program 211
saving a copy of a workspace 81
saving a workspace 80
saving projects 38
saving the contents of the Breakpoints window 225
searching a single file 125
searching for text across multiple files 139
searching for text across multiple folders 133
searching for text across multiple projects 135
searching for text across multiple symbolics files 137
searching with a text selection 144
selecting entire routines 98
selecting item in Code Completion window 108
selecting lines 98
selecting multiple lines 98
selecting rectangular portions of lines 98
selecting text in editor windows 97
setting a breakpoint 226
setting a conditional breakpoint 230
setting a conditional eventpoint 243
setting a conditional watchpoint 247
setting a Log Point 235
setting a Pause Point 236
setting a Script Point 237
setting a Skip Point 239
setting a Sound Point 239
setting a temporary breakpoint 229
setting a Trace Collection Off eventpoint 240
setting a Trace Collection On eventpoint 241
setting a watchpoint 245
showing the classes pane 175
sorting the classes list 175
specifying the default breakpoint template 233
starting the debugger 208
stepping into a routine 209
stepping out of a routine 209
stepping over a routine 210
stopping program execution 210
suppressing dockable windows 75
toggling automatic punctuation balancing 101
toggling the symbol hint 212
triggering code completion by keyboard 104
triggering code completion from IDE menu bar 104
unapplying file differences 155

undocking a window 73
unfloating a window 75
unindenting text blocks 100
updating data from a logic analyzer 313
using an external editor on the Macintosh 392
using contextual menus 214
using the default workspace 80
using the document settings pop-up 93
using the Executables pane in the Symbolics window 277, 278
using the Files pane in the Symbolics window 277
using the Find Next command 143
using the Find Previous command 143
using the Functions list pop-up 114
using the Functions pane in the Symbolics window 278
using the Interfaces list pop-up 114
using the symbol hint 212
using the VCS pop-up 94
using virtual space 99
viewing a file path 49
viewing breakpoint properties 227
viewing browser data by contents 184
viewing browser data by inheritance 180
viewing class data from hierarchy windows 172
viewing eventpoint properties 242
viewing global variables for different processes 250
viewing registers 268
viewing watchpoint properties 246
template, default for breakpoints 231
template, for breakpoints 231
Templates option 410
Templates tab 225
templates, creating for breakpoints 231
templates, deleting for breakpoints 232
templates, reference (Macintosh) 120
templates, specifying the default for breakpoints 233
temporary breakpoint, defined 229
temporary breakpoints 222
 setting 229
text
 changing a find string 144
find by selecting 143
finding 123
overstriking (Windows) 99
replacing 123
searching with a selection 144
text blocks, indenting 100
text blocks, unindenting 100
text boxes
 Address 271
 Analyzer Configuration File text box 311
 Analyzer Slot 312
 Bit Value 272
 Description File 271, 273
 End 293, 304
 Flash Memory Base Address 290
 Host Name 311
 Message 235, 236
 Name 311
 Offset 293
 Passes 305
 Result Count 142
 Search Criteria 142
 Size 297
 Start 293, 297, 304
 Target Address 301, 303
 Target Memory Buffer Address 288
 Target Memory Buffer Size 288
 Target Scratch Memory End 305
 Target Scratch Memory Start 305
 Trace Support File 312
 Use Selected File 292
 Use Target Initialization 288, 300
 Value to Write 301, 303
Text Colors panel
options
 Activate Browser Coloring 464
 Activate Syntax Coloring 464, 467, 481
 Foreground 464
 Keywords 467
 Strings 481
Text Colors preference panel
options
 Activate Browser Coloring 410
 Activate Syntax Coloring 409
 Background 409
 Classes 410
 Comments 409

Constants 410
Edit 410
Enums 410
Foreground 409
Functions 410
Globals 410
Keywords 409
Macros 410
Other 410
Set 1, Set 2, Set 3, Set 4 410
Strings 410
Templates 410
TypeDefs 410

text editing area, in editor window 95
Text View list box 273, 274
 Auto 274
Text View pop-up menu
 Bitfield Description option 274
 Register Description option 274
 Register Details option 274
text/list boxes
 By Type 132
 Find 124, 127, 130
 Replace With 127, 130
 Search in 132
 Target Processor 288, 300
text-selection Find 143
THINK Reference 119, 120, 463
third-party editor support 488
third-party text editors
 Emacs 467, 468

Thread window
 Breakpoints button 206
 current-statement arrow 207
 dash 207
 debug button 206
 Expressions button 206
 Functions list box 208
 Kill button 206
 Line And Column button 208
 Pane Collapse box 206
 Pane Expand box 206
 Pane resize bar 207
 Resume button 206
 run button 206
 Source File button 207
 Source list box 208

Source pane 207
Source Pane disclosure triangle 207
Stack pane 207
Step Into button 206
Step Out button 206
Step Over button 206
Stop button 206
Symbolics button 206
Variables pane 207
Variables Pane Listing button 207

thread window 204
thread, breakpoint property 228
threading in IDE 455
 `__throw()` 525
ticksTimeBase 331, 336, 345

Tile Editor Windows command 552
Tile Editor Windows Vertically command 552
Tile Horizontally command 552
Tile Vertically command 553
time hogs, finding 340
timebase 331, 345
timeMgrTimeBase 331, 345
times hit, breakpoint property 228
times left, breakpoint property 228
To Smallest Height command in Resize
 submenu 554
To Smallest Width command in Resize
 submenu 555

toggling
 symbol hint 212

toolbar
 collapsing in editor window 92
 expanding in editor window 92

Toolbar (Editor Window) Elements
 Document Settings 377
 File Dirty Indicator 377
 File Path field 377
 Functions 377
 Header Files 377
 Markers 377
 Version Control Menus 377

toolbar buttons
 Browser Contents 168
 Class Hierarchy 168
 Go Back 168
 Go Forward 168
 Make 46

Single Class Hierarchy Window 169
Synchronize Modification Dates 46
Target Settings 46
Toolbar Items 363, 376
Toolbar submenu
 Anchor Floating Toolbar command 524
 Clear Floating Toolbar command 528
 Clear Main Toolbar command 528
 Clear Window Toolbar command 528
 Hide Floating Toolbar command 538
 Hide Main Toolbar command 538
 Reset Window Toolbar command 51, 546
 Show Floating Toolbar command 538, 550
 Show Main Toolbar command 538
 Show Window Toolbar command 550
Toolbars
 Add element 375, 376
 Clear Elements 377
 Customize 373
 Elements 373, 375
 Icons 376
 Instances of 375
 Main (floating) 374
 Modify 375
 Project and Window 374
 Remove single element 375
 Toolbar Items tab 375
 Types 374
toolbars
 editor 92
 for Symbols window 187
 resetting 378
tools
 browser 27
 build system 27
 debugger 27
 editor 27
 project manager 27
 search engine 27
Tools menu 521
 icon 521
tools, for hardware 285
ToolServer menu 488
ToolServer Worksheet command 554
ToolTip 376
touch
 defined 48
Touch column 56, 57
 in Files view of Project window 48
Touch command 56
touching
 all files 56
 all groups 56
 files 56
 groups 56
trace
 working with logic analyzer 309
Trace Collection Off 240
Trace Collection Off eventpoint 234
Trace Collection Off eventpoint, clearing 241
Trace Collection Off eventpoint, setting 240
Trace Collection On 241
Trace Collection On eventpoint 234
Trace Collection On eventpoint, clearing 241
Trace Collection On eventpoint, setting 241
Trace Support File text box 312
Trace window 314
 opening 314
Treat as Expression checkbox 236
triggering
 code completion by keyboard 104
 code completion from IDE menu bar 104
turning off
 symbol hint 212
turning on
 symbol hint 212
Type list box
 Absolute Path option 484
Type option
 Source Trees panel 484
Type pop-up menu
 Environment Variable option 484
 Registry Key option 485
type, breakpoint property 227
TypeDefs option 410
types
 of documentation 21

U

Unanchor Floating Toolbar command 555
Unapply button 153
Unapply Difference command 155, 555
unbounded arrays, setting default size for
 viewing 457

Uncaught Exceptions Only command 555
Undo button 153
Undo command 555
undocking windows 73
unfloating windows 75
Ungroup command 555
unindenting text blocks 100
Untouch command 57
untouching
 a file 57
 a group 57
 all files 57
 all groups 57
Update Data command 313
Update Data Every n Seconds option 485
Use Concurrent Compiles option 473, 485
Use Custom Settings checkbox 288, 299
Use Debugging Monitor option 485
Use Default Workspace option 485
Use External Debugger option 486
Use External Editor option 486
Use Local Project Data Storage option 486
Use modification date caching option 486
Use Multiple Document Interface option 67, 487
 turning off 391
 turning on 391
Use multiple undo option 555
 in Editor Settings panel 487
Use Script menu option 487
Use Scripts Menu option 522
Use Selected File checkbox 292
Use Selected File text box 292
Use Target CPU checkbox 305
Use Target Initialization checkbox 288, 300
Use Target Initialization text box 288, 300
Use Third Party Editor option 488
Use ToolServer Menu option 521
Use ToolServer menu option
 IDE Extras panel 488
User Paths list
 Framework column 430
 Recursive Search column 430
 Search Status column 429
User Paths option 488
User Specified option 488
User specified option 446
using
document settings pop-up 93
Executables pane in the Symbolics window 277, 278
Files pane in the Symbolics window 277
Find Next command 143
Find Previous command 143
Functions list pop-up 114
Functions pane in the Symbolics window 278
Interfaces list pop-up 114
logic analyzer 312
Register Details window 273
symbol hint 212
VCS pop-up 94
virtual space 99

V

Value to Write text box 301, 303
variable formatting 252
Variable Values Change option
 Display Settings panel 489
Variable window 251
variables
 ^var placeholder 254
 adding a constant to 256
 making a summation of 256
 manipulating formats 252
 symbol hint 211
Variables pane 207
Variables Pane Listing button 207
variables, working with 249
VCS 94
 list pop-up 170
 menu 521
 pop-up 93
VCS Commands option
 Editor Settings panel 489
VCS menu 177, 489
 icon 521
VCS pop-up
 using 94
Vectorization 440
Verify button 294
version control 489, 521
Version Control Settings command 555
Version Control System. *See VCS.*

Vertical Center command in Align submenu 547, 553, 554, 556
View Array menu command 556
View as implementor 170
View as subclass 170
View As Unsigned Decimal menu command 556, 557
View as user 170
View Disassembly menu command 557
view in profiler
 class 339
 detail 338
 flat 337
View Memory As command 557
View Memory command 557
View Mixed menu command 558
View Source menu command 558
View Target Memory Writes checkbox 289
View Variable menu command 558
viewing
 all symbol implementations 187
 breakpoints 225
 browser data by contents 184
 browser data by inheritance 180
 file paths 49
 register details 270
 registers 268
viewing access breakpoint 246
viewing installed plug-ins 523
viewing installed products 523
virtual
 icon for 176
virtual space, using 99

W

Walking 1's checkbox 304
Walking Ones subtest 306
Walking Ones test
 Address Line fault 306
 Data Line fault 306
 Retention fault 306
 subtests
 Ones Retention 306
 Walking Ones 306
 Walking Zeros 306
 Zeros Retention 306
Walking Zeros subtest 306

Warnings button 142
Watchpoint Indicator option
 Display Settings panel 489
watchpoints
 access breakpoint 244
 clearing all 247
 defined 244
 enabled 244
 purpose of 221
 setting conditional 247
watchpoints, clearing 247
watchpoints, disabling 246
watchpoints, enabling 246
watchpoints, setting 245
watchpoints, viewing properties for 246
what is
 a debugger 203
 a symbolics file 204
win32TimeBase 331, 345
window
 Customize IDE Commands 379
Window Follows Insertion Point option 490
Window menu 497, 506, 519
 Restore Window command (Windows) 547
Window position and size option
 Editor Settings panel 490
Window Settings preference panel 414
options
 Close non-debugging windows 415
 Collapse non-debugging windows 415
 Do nothing 415
 Do nothing to project windows 415
 Hide non-debugging windows 415
 Minimize non-debugging windows 415
 Monitor for debugging 415
 Move open windows to debugging
 monitor when debugging starts 416
 Open windows on debugging monitor
 during debugging 416
 Use Debugging Monitor 415
window types
 docked 68
 floating 68
 MDI child 68
Windowing panel
options
 Hide non-debugging windows 465

Minimize non-debugging windows 470
Monitor for debugging 470
Move open windows to debugging monitor when debugging starts 471
Open windows on debugging monitor during debugging 471
Use Debugging Monitor 485

Windows
 creating files 61
windows 222
 Array 264
 Browser Contents 183
 Cache 315
 Class Browser 167
 Code Completion 105
 Command 317
 Compare Files Setup 149
 Customize IDE Commands 143
 dock bars in dockable windows 76
 dockable 67
 dockable, about 67
 dockable, turning off 75
 dockable, working with 70
 docking the same kind of 72
 docking with a contextual menu 70
 docking with drag and drop 71
 editor 89
 editor, other 94
 Expressions 254
 File Compare Results 152
 Find (single-file) 123
 Find and Replace (multiple-file) 129
 Find and Replace (single-file) 126
 Flash Programmer 285
 floating 74
 Folder Compare Results 155
 Global Variables 249
 Hardware Diagnostics 297
 hierarchy 180
 IDE Preferences 245, 385
 Log 281
 making MDI children of 75
 Memory 259
 New C++ Class 191
 New C++ Data Member 198
 New C++ Member Function 196
 Processes 278

project window 45
Registers 266
remembering size and position of 548
Remove Markers 117
saving default size and position of 548
Search results 141
Symbolics 275
Target Settings 423
Trace 314
undocking 73
unfloating 75
variable 251
Windows menu layout 493
WinHelp (Windows) 119
Wizards
 Browser 189
wizards
 New Class 174, 189, 190
 New Data Member 177, 197
 New Member Function 194, 195
 New Member Functions 176
Word option button 301, 303, 305
working
 with IDE preferences 385
 with IDE target settings 423
Working Directory option
 Runtime Settings panel 490
working with 70
 browser 161
 class browser windows 167
 class hierarchy windows 179
 IDE hardware tools 285
 logic analyzer 309, 312
working with breakpoint templates 231
working with breakpoints 226
working with debugger data 275
working with dockable windows 70
working with eventpoints 242
working with files 61
working with memory 259
working with projects 31
working with variables 249
workspace, defined 79
workspaces 79
 closing 81
 opening 81
 opening recent 82

saving 80
saving copies of 81
using default 80
Workspaces option
 IDE Extras panel 490
workspaces, about 79
Write button 273
Write option button 301, 302

X

XML

exporting projects 40
importing projects 40
Register Details Window
 BFVALUE 567
 BITFIELD 563
 file locations 568
 REGISTER 561
 sample files 569
Register Details window
 specification 559

Z

Zeros Retention subtest 306
Zoom Window menu command 558
Zoom windows to full screen option
 IDE Extras panel 490

