CodeWarrior™
Development Studio
IDE 5.7 User’s Guide

Revised: 20 February 2007
Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. CodeWarrior is a trademark or registered trademark of Freescale Semiconductor, Inc. in the United States and/or other countries. All other product or service names are the property of their respective owners.

Copyright © 2004–2007 by Freescale Semiconductor, Inc. All rights reserved.

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

How to Contact Us

<table>
<thead>
<tr>
<th>Corporate Headquarters</th>
<th>Freescale Semiconductor, Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7700 West Parmer Lane</td>
</tr>
<tr>
<td></td>
<td>Austin, TX 78729</td>
</tr>
<tr>
<td></td>
<td>U.S.A.</td>
</tr>
<tr>
<td>World Wide Web</td>
<td><a href="http://www.freescale.com/codewarrior">http://www.freescale.com/codewarrior</a></td>
</tr>
<tr>
<td>Technical Support</td>
<td><a href="http://www.freescale.com/support">http://www.freescale.com/support</a></td>
</tr>
</tbody>
</table>
Table of Contents

I Introduction

1 IDE User's Guide Overview .......................................................... 17
   Release Notes .............................................................................. 17
   Licensing ...................................................................................... 17
   Documentation Structure ............................................................ 18
      Documentation Formats ............................................................. 18
      Documentation Types ................................................................. 19
   Manual Conventions .................................................................... 19
      Figure Conventions ................................................................... 19
   Keyboard Conventions ................................................................. 20
      Special note for Solaris and Linux users ..................................... 20

2 CodeWarrior IDE Overview .......................................................... 21
   Development Cycle ....................................................................... 21
   CodeWarrior IDE Advantages ...................................................... 23
   IDE Tools Overview ..................................................................... 24

II Projects

3 Working with Projects .................................................................. 29
   About Projects ............................................................................. 29
      Project Manager ......................................................................... 29
      Build Targets ............................................................................ 31
   Managing Projects ....................................................................... 32
   Advanced Projects ...................................................................... 38
      Custom Project Stationery .......................................................... 38
      Subprojects .............................................................................. 39
   Strategies ..................................................................................... 40
# Table of Contents

## 4 Project Window

- About the Project Window ............................................. 41
- Project Window Pages .................................................. 43
  - Files Page .............................................................. 43
  - Link Order Page ...................................................... 47
  - Targets Page .......................................................... 47
- File, Group, Layout, and Target Management ..................... 48
- Build-Target Management ............................................. 53

## 5 Working with Files

- Managing Files .......................................................... 57

## 6 Dockable Windows

- About Dockable Windows .............................................. 63
- Working with Dockable Windows ..................................... 66
- Dock Bars ................................................................. 70

## 7 Workspaces

- About Workspaces ...................................................... 73
- Using Workspaces ....................................................... 73

## 8 Creating Console Applications

- About Console Applications .......................................... 77
- Creating Console Applications ...................................... 77

## III Editor

## 9 CodeWarrior Editor

- Editor Window ........................................................... 83
- Editor Toolbar ............................................................ 86
  - Interfaces Menu ....................................................... 86
  - Functions Menu ...................................................... 86
Table of Contents

Markers Menu ......................................................... 86
Document Settings Menu ........................................... 87
Version Control System Menu .................................... 87
Other Editor Window Components ................................. 88
Path Caption .......................................................... 88
File Modification Icon ............................................... 88
Breakpoints Column .................................................. 88
Text Editing Area ..................................................... 88
Line and Column Indicator ......................................... 89
Pane Splitter Controls ................................................. 89

10 Editing Source Code ............................................... 91
Text Manipulation ...................................................... 91
Symbol Editing Shortcuts ............................................ 94
Punctuation Balancing ............................................... 94
Code Completion ....................................................... 96
Code Completion Configuration .................................... 96
Code Completion Window ........................................... 98

11 Navigating Source Code ........................................... 105
Finding Interface Files, Functions, Lines ......................... 105
Finding Interface Files ................................................ 105
Locating Functions .................................................... 106
Going Back and Forward ............................................. 107
Using Markers .......................................................... 108
Remove Markers Window ............................................. 108
Symbol Definitions ..................................................... 110
Reference Templates (Macintosh) .................................. 111

12 Finding and Replacing Text ........................................ 113
Single-File Find ......................................................... 113
Single-File Find and Replace ....................................... 115
Multiple-File Find and Replace .................................... 118
In Folders ............................................................... 120
In Projects ............................................................... 122
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Symbolics</td>
<td>124</td>
</tr>
<tr>
<td>In Files</td>
<td>126</td>
</tr>
<tr>
<td>Search Results Window</td>
<td>128</td>
</tr>
<tr>
<td>Text-Selection Find</td>
<td>130</td>
</tr>
<tr>
<td>Regular-Expression Find</td>
<td>132</td>
</tr>
<tr>
<td>Using the Find String in the Replace String</td>
<td>134</td>
</tr>
<tr>
<td>Remembering Sub-expressions</td>
<td>135</td>
</tr>
<tr>
<td>Comparing Files and Folders</td>
<td>135</td>
</tr>
<tr>
<td>Comparison Setup</td>
<td>136</td>
</tr>
<tr>
<td>File Comparison</td>
<td>139</td>
</tr>
<tr>
<td>Folder Comparison</td>
<td>142</td>
</tr>
</tbody>
</table>

IV Browser

13 Using the Browser 147

Browser Database ................................................................. 147
Browser Data ................................................................. 147
Browser Symbols .............................................................. 150
Browser Contextual Menu ...................................................... 150

14 Using Class Browser Windows 153

Class Browser window ......................................................... 153
Classes Pane ................................................................. 159
Member Functions Pane ......................................................... 161
Data Members Pane ............................................................ 162
Source Pane ................................................................. 163
Status Area ................................................................. 163

15 Using Other Browser Windows 165

Multiple-Class Hierarchy Window ........................................... 165
Single-Class Hierarchy Window ............................................. 168
Browser Contents window .................................................... 169
Symbols Window ................................................................. 171
Table of Contents

Symbols toolbar. ................................................. 173
Symbols pane ................................................. 173
Source pane ................................................. 173

16 Using Browser Wizards 175
New Class Wizard ........................................ 175
The New Member Function Wizard ................. 180
The New Data Member Wizard .................... 183

V Debugger

17 Working with the Debugger 187
About the Debugger ....................................... 187
About Symbolics Files ................................... 188
Thread Window ............................................. 188
Common Debugging Actions ......................... 192
Symbol Hint ............................................... 195
Contextual Menus .......................................... 196
Multi-Core Debugging .................................... 197
Data Viewer Plugins ..................................... 198
External Builds Support ................................. 198
  External Build Wizard ................................. 200

18 Manipulating Program Execution 203
Breakpoints ................................................. 203
Breakpoints Window ..................................... 204
Working with Breakpoints ............................... 207
Working with Breakpoint Templates ................ 212
Eventpoints ................................................ 214
  Log Point ............................................. 216
  Pause Point ......................................... 217
  Script Point ......................................... 218
  Skip Point .......................................... 219
# Table of Contents

- Sound Point (Windows OS) .................................................. 220
- Trace Collection Off ...................................................... 221
- Trace Collection On ....................................................... 221
- Working with Eventpoints ............................................... 222
- Watchpoints .................................................................. 224
- Special Breakpoints ....................................................... 228

## 19 Working with Variables .......................... 229

- Global Variables Window ............................................... 229
- Variable Window ............................................................ 231
- Expressions Window ...................................................... 233

## 20 Working with Memory ......................... 237

- Memory Window ............................................................ 237
- Array Window ................................................................ 242
- Registers Window .......................................................... 244
  - General Registers ....................................................... 245
  - FPU Registers ........................................................... 245
  - Host-specific Registers ............................................... 245
- Register Details Window (Windows OS) ...................... 247
- Description File ............................................................ 250
- Register Display ............................................................ 251
- Text View .................................................................... 251

## 21 Working with Debugger Data ................. 253

- Symbolics Window ........................................................ 253
- System Browser Window .............................................. 256
- Log Window .................................................................. 258

## 22 Working with Hardware Tools ............... 261

- Flash Programmer Window .......................................... 261
- Target Configuration ..................................................... 263
- Flash Configuration ....................................................... 265
- Program / Verify ........................................................... 267
- Erase / Blank Check ...................................................... 270
<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksum ................................. 272</td>
</tr>
<tr>
<td>Hardware Diagnostics Window .................. 274</td>
</tr>
<tr>
<td>Configuration .................................. 275</td>
</tr>
<tr>
<td>Memory Read / Write .......................... 276</td>
</tr>
<tr>
<td>Scope Loop .................................... 278</td>
</tr>
<tr>
<td>Memory Tests ................................ 280</td>
</tr>
<tr>
<td>Working with a Logic Analyzer .................. 285</td>
</tr>
<tr>
<td>Configuring the Project ....................... 285</td>
</tr>
<tr>
<td>Using the Logic Analyzer ........................ 288</td>
</tr>
<tr>
<td>Trace Window ................................... 290</td>
</tr>
<tr>
<td>Cache Window .................................. 290</td>
</tr>
<tr>
<td>Profile Window .................................. 291</td>
</tr>
<tr>
<td>Command Window ................................ 292</td>
</tr>
<tr>
<td>23 Profiler .................................... 295</td>
</tr>
<tr>
<td>Overview ...................................... 295</td>
</tr>
<tr>
<td>What Is a Profiler? ............................. 295</td>
</tr>
<tr>
<td>Types of Profilers .............................. 296</td>
</tr>
<tr>
<td>A Profiling Strategy ............................ 297</td>
</tr>
<tr>
<td>Profiling Code ................................... 298</td>
</tr>
<tr>
<td>Using the Profiler .............................. 298</td>
</tr>
<tr>
<td>What It Does .................................... 299</td>
</tr>
<tr>
<td>How It Works .................................... 299</td>
</tr>
<tr>
<td>Profiling Made Easy ............................. 300</td>
</tr>
<tr>
<td>Configuring ...................................... 303</td>
</tr>
<tr>
<td>Profiler Libraries and Interface Files ....... 303</td>
</tr>
<tr>
<td>Profiling Special Cases ........................ 303</td>
</tr>
<tr>
<td>Viewing Results ................................ 308</td>
</tr>
<tr>
<td>What It Does .................................... 309</td>
</tr>
<tr>
<td>How It Works .................................... 309</td>
</tr>
<tr>
<td>Finding Performance Problems ................... 313</td>
</tr>
<tr>
<td>Troubleshooting ................................. 314</td>
</tr>
<tr>
<td>Profile Times Vary Between Runs ............... 314</td>
</tr>
<tr>
<td>Problems while Profiling Inline Functions .... 315</td>
</tr>
<tr>
<td>Profiling Library Could not be Found .......... 315</td>
</tr>
</tbody>
</table>
Table of Contents

Reference ................................................................. 316
  Compiler Directives .................................................. 316
  Memory Usage .......................................................... 317
  Time and Timebases ................................................... 317
  Profiler Function Reference ....................................... 318
  ProfilerInit() .......................................................... 318
  ProfilerTerm() ......................................................... 320
  ProfilerSetStatus() ................................................... 320
  ProfilerGetStatus() ................................................... 320
  ProfilerGetDataSizes() ............................................... 321
  ProfilerDump() ......................................................... 321
  ProfilerClear() ........................................................ 322

VI Compilers and Linkers

24 Compilers ............................................................ 325
  Choosing a Compiler .................................................. 325
  Compiling Projects ................................................... 325

25 Linkers ............................................................... 329
  Choosing Linkers ..................................................... 329
  Linking Projects ........................................................ 330

VII Preferences and Target Settings

26 Customizing the IDE .................................................. 333
  Customizing IDE Commands ......................................... 333
  Commands Tab .......................................................... 335
  Pre-Defined Variables in Command Definitions .................. 338
  Customize Toolbars .................................................... 342
  Kinds of Toolbars ...................................................... 343
27 Working with IDE Preferences

28 Working with Target Settings
# Table of Contents

- Code Generation Panels ........................................... 402
  - Global Optimizations ........................................ 402
- Editor Panels ......................................................... 405
  - Custom Keywords ............................................. 406
- Debugger Panels ..................................................... 407
  - Other Executables ........................................... 407
  - Debugger Settings ............................................ 410
  - Remote Debugging ........................................... 411

## 29 Preference and Target Settings Options  ................................................................. 415

- A ................................................................. 415
- B ................................................................. 417
- C ................................................................. 419
- D ................................................................. 421
- E ................................................................. 423
- F ................................................................. 427
- G-I ............................................................... 429
- K-L ............................................................... 431
- M ................................................................. 433
- O ................................................................. 435
- P ................................................................. 435
- R ................................................................. 437
- S ................................................................. 438
- T ................................................................. 446
- U ................................................................. 447
- V ................................................................. 450
- W-Z ............................................................... 451

## 30 Register Details Window XML Specification .............................................................. 453

- Register Details Window XML Specification .................................................. 453
  - REGISTER ......................................................... 455
  - BITFIELD ......................................................... 457
  - BFVALUE ......................................................... 460
- Accessing the XML Files from CodeWarrior ...................................................... 461
- A Sample XML File .................................................. 462
## Table of Contents

- Creating the New XML File .................................................. 462
- Adding Multiple BITFIELD Attributes ................................. 463
- Adding BFVALUE Attributes .............................................. 465
- Completing the New XML File. ........................................... 467
- References ........................................................................... 470

### VIII Menus

#### 31 IDE Menus

<table>
<thead>
<tr>
<th>Menu Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Menu Layout</td>
<td>473</td>
</tr>
<tr>
<td>File Menu</td>
<td>473</td>
</tr>
<tr>
<td>Edit Menu</td>
<td>475</td>
</tr>
<tr>
<td>View Menu</td>
<td>477</td>
</tr>
<tr>
<td>Search Menu</td>
<td>478</td>
</tr>
<tr>
<td>Project Menu</td>
<td>479</td>
</tr>
<tr>
<td>Debug Menu</td>
<td>482</td>
</tr>
<tr>
<td>Data Menu</td>
<td>483</td>
</tr>
<tr>
<td>Window Menu</td>
<td>485</td>
</tr>
<tr>
<td>Help Menu</td>
<td>486</td>
</tr>
<tr>
<td>Macintosh Menu Layout</td>
<td>486</td>
</tr>
<tr>
<td>Apple Menu</td>
<td>486</td>
</tr>
<tr>
<td>CodeWarrior Menu</td>
<td>487</td>
</tr>
<tr>
<td>File Menu</td>
<td>487</td>
</tr>
<tr>
<td>Edit Menu</td>
<td>489</td>
</tr>
<tr>
<td>Search Menu</td>
<td>490</td>
</tr>
<tr>
<td>Project Menu</td>
<td>492</td>
</tr>
<tr>
<td>Debug Menu</td>
<td>495</td>
</tr>
<tr>
<td>Data Menu</td>
<td>497</td>
</tr>
<tr>
<td>Window Menu</td>
<td>498</td>
</tr>
<tr>
<td>VCS Menu</td>
<td>500</td>
</tr>
<tr>
<td>Tools Menu</td>
<td>500</td>
</tr>
<tr>
<td>Scripts Menu</td>
<td>501</td>
</tr>
<tr>
<td>Help Menu</td>
<td>501</td>
</tr>
</tbody>
</table>
Table of Contents

32 Menu Commands 503

A ......................................................... 503
B ......................................................... 504
C ......................................................... 506
D ......................................................... 511
E ......................................................... 512
F ......................................................... 514
G ......................................................... 516
H ......................................................... 517
I ......................................................... 518
K-L ...................................................... 518
M-N ..................................................... 519
O ......................................................... 521
P-Q ..................................................... 521
R ......................................................... 522
S ......................................................... 527
T-U ...................................................... 531
V-Z ..................................................... 535

Index 539
Introduction

This section consists of 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.

- **Documentation Structure**—a guide to the various CodeWarrior manuals available. This guide notes the location of generic and specific product documentation.
- **Manual Conventions**—some common typographical conventions used in this manual and other CodeWarrior documentation.

Release Notes

Please read the release notes, which 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.

CodeWarrior 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.

During product installation you will be instructed to register at [http://www.freescale.com/cwregister](http://www.freescale.com/cwregister). 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 list, if one exists. After entering the authorization code, the CodeWarrior IDE will make an HTTP call to the CodeWarrior 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.

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.
IDE User’s Guide Overview

Manual Conventions

- 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Macintosh Name</th>
<th>Solaris Equivalent</th>
<th>Linux Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>⌘</td>
<td>Control</td>
<td>Control</td>
<td>Ctrl</td>
</tr>
<tr>
<td>⌘</td>
<td>Option</td>
<td>Alt</td>
<td>Alt</td>
</tr>
<tr>
<td>⌘</td>
<td>Command</td>
<td>Meta</td>
<td>Alt</td>
</tr>
<tr>
<td>⌘</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
</tr>
</tbody>
</table>
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 consists of these sections:

- Development Cycle
- CodeWarrior IDE Advantages
- IDE Tools Overview

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 depicts the development cycle as a flowchart. Table 2.1 details the different stages and their corresponding sections in this manual.
CodeWarrior IDE Overview
Development Cycle

Figure 2.1 Development Cycle Diagram

Legend

- Start or End of the process
- Development Cycle stage
- Decision stage
- Process flow

Start

Create

Edit

Compile

Yes

Compile Errors?

No

Link

Link Errors?

Yes

No

Debug

Debug Errors?

Yes

No

Release

End
CodeWarrior IDE Overview

CodeWarrior IDE Advantages

Table 2.1 Stage Descriptions, Related User’s Guide Sections

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
<th>Related Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Create the initial project, source files, and build targets.</td>
<td>Projects, Preferences and Target Settings, Menus</td>
</tr>
<tr>
<td>Edit</td>
<td>Transform your project into working source code, organize interface elements, and correct errors.</td>
<td>Editor, Browser</td>
</tr>
<tr>
<td>Compile</td>
<td>Compile the source code into machine format that operates on the target host.</td>
<td>Compilers and Linkers</td>
</tr>
<tr>
<td>Link</td>
<td>Link the separate compiled modules into a single binary executable file.</td>
<td>Compilers and Linkers</td>
</tr>
<tr>
<td>Debug</td>
<td>Find and resolve all coding and logic errors that prevent the program from operating as designed.</td>
<td>Debugger</td>
</tr>
<tr>
<td>Release</td>
<td>Release for public use.</td>
<td>Beyond the scope of this manual.</td>
</tr>
</tbody>
</table>

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.

- plugin tool support
  Extend the capabilities of the IDE by adding a plugin tool that supports new services. The IDE currently supports plugins for compilers, linkers, pre-linkers, post-linkers, preference panels, version controls, and other tools. Plugins 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 explains the purpose of these tools and lists corresponding CodeWarrior IDE features.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
<th>CodeWarrior IDE Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>Manipulate items associated with a project</td>
<td>• Handles top-level file management for the software developer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Organizes project items by major group, such as files and targets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Tracks state information (such as file-modification dates)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Determines build order and files to be included in each build</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Coordinates with plugins to provide version-control services</td>
</tr>
<tr>
<td>Editor</td>
<td>Create and modify source code</td>
<td>• Uses color to differentiate programming-language keywords</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allows definition of custom keywords for additional color schemes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Automatically verifies parenthesis, brace, and bracket balance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allows use of menus for navigation to any function or into the header files used by the program</td>
</tr>
<tr>
<td>Search Engine</td>
<td>Find and replace text</td>
<td>• Finds a specific text string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Replaces found text with substitute text</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allows use of regular expressions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Provides file-comparison and differencing functionality</td>
</tr>
<tr>
<td>Source Browser</td>
<td>Manage and view program symbols</td>
<td>• Maintains a symbolics database for the program. Sample symbols include names and values of variables and functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Uses the symbolics database to assist code navigation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Links every symbol to other locations in the code related to that symbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Processes both object-oriented and procedural languages</td>
</tr>
</tbody>
</table>
### CodeWarrior IDE Overview

#### IDE Tools Overview

#### Table 2.2 IDE Tools and Features (continued)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
<th>CodeWarrior IDE Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Build System</strong></td>
<td>Convert source code into an executable file</td>
<td>• Uses compiler to generate object code from source code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Uses linker to generate final executable file from object code</td>
</tr>
<tr>
<td><strong>Debugger</strong></td>
<td>Resolve errors</td>
<td>• Uses symbolics database to provide source-level debugging</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Supports symbol formats such as CodeView, DWARF (Debug With Arbitrary Records Format), and SYM (SYMbolic information format)</td>
</tr>
</tbody>
</table>
Projects

This section consists of 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 consists of these sections:

- About Projects
- Managing Projects
- Advanced Projects

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 diagrams Project Manager interactions with IDE tools. Table 3.1 explains the interactions.
Figure 3.1 Project Manager

![Diagram of Project Manager interactions]

Table 3.1 Project Manager Interactions

<table>
<thead>
<tr>
<th>IDE Tool</th>
<th>Project Manager Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor</td>
<td>• Coordinates internal data flow among editor windows, search engine, and source browser</td>
</tr>
<tr>
<td></td>
<td>• Matches find-and-replace results between related header files and source files</td>
</tr>
<tr>
<td></td>
<td>• Associates functions and variables with corresponding source code</td>
</tr>
<tr>
<td>Compiler</td>
<td>• Synchronizes a symbolics database of program functions, variables, and values with source code</td>
</tr>
<tr>
<td></td>
<td>• Coordinates internal data flow between symbolics database and source browser</td>
</tr>
<tr>
<td></td>
<td>• Determines files to include in build process</td>
</tr>
</tbody>
</table>
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 shows a sample project with debug and release build targets.
Managing Projects

Use these tasks to manage projects:

- Create a new project

Note that both build targets share File #1 and File #2.
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.

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

Use the Makefile Importer wizard to convert most Visual C \texttt{make} or GNU \texttt{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

2. Click the Project tab.
3. Select Makefile Importer Wizard.
Working with Projects

Managing Projects

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.
   - **CodeWarrior 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.

**NOTE** Avoid creating empty projects. Instead, modify a project created with project stationery. 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 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.

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
Working with Projects
Managing Projects

- 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

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Project Inspector.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Project Inspector.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; Project Inspector.</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; Project Inspector.</td>
</tr>
</tbody>
</table>

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.
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

Use custom project stationery to develop streamlined templates to meet 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 modify existing project stationery and save it under a new name in the **Stationery** or **Project Stationery** folder.
1. Follow the usual process for creating a project from project stationery. See Creating New Projects using Project Stationery for more information.

2. Choose File > Save A Copy As.

3. Find the Project Stationery folder in the CodeWarrior installation.

4. Create a folder inside the Project 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 Project 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 plugins 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
Working with Projects
Advanced Projects

- 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:
- Subprojects separate distinct parts of a complex program, such as an application and its various plugins.
- Using subprojects streamlines a complicated build. For example, create a project that builds all plugins for an application. Add this project as a subproject of the main application. The IDE then builds all plugins 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 consists of these sections:

- About the Project Window
- Project Window Pages
- File, Group, Layout, and Target Management
- Build-Target Management

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 shows a sample Project window. Table 4.1 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 plugins.
Project Window

About the Project Window

Figure 4.1 Project Window

Table 4.1 Project Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Target</td>
<td><img src="image" alt="Icon" /></td>
<td>Use to specify the build target that you want to modify.</td>
</tr>
<tr>
<td>Target Settings</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to view and edit the settings for the current build target. You can also display settings for a target selected in Targets tab.</td>
</tr>
<tr>
<td>Synchronize Modification Dates</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to check the modification dates of each project file and mark those files that need compilation.</td>
</tr>
<tr>
<td>Make</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to compile and link all modified and manually selected (touched) project files.</td>
</tr>
<tr>
<td>Debug</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to debug the current build target.</td>
</tr>
<tr>
<td>Run</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to compile and link the current build target, then run the program.</td>
</tr>
</tbody>
</table>
Project Window

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:

- 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>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>File</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Target</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Debug</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Checkout Status</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Table 4.3 Viewing a File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>On this host...</strong></td>
</tr>
<tr>
<td>Windows</td>
</tr>
<tr>
<td>Macintosh</td>
</tr>
<tr>
<td>Solaris</td>
</tr>
<tr>
<td>Linux</td>
</tr>
</tbody>
</table>

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.

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 > Re-search 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 > Re-search 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 explains the items in the Link Order page.

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronize Modification Dates</td>
<td>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.</td>
</tr>
<tr>
<td>Synchronize Status</td>
<td>To update version-control status information, click the Pencil icon.</td>
</tr>
</tbody>
</table>

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.
Project Window
File, Group, Layout, and Target Management

Table 4.5 Targets Page Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Targets</td>
<td>Displays all build targets and subprojects that the IDE processes to create a binary file. These icons denote build-target status:</td>
</tr>
<tr>
<td></td>
<td>• <img src="image" alt="active build target" /> active build target</td>
</tr>
<tr>
<td></td>
<td>• <img src="image" alt="inactive build target" /> inactive build target</td>
</tr>
<tr>
<td>Link</td>
<td>Indicates the dependencies between build targets and subprojects.</td>
</tr>
</tbody>
</table>

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 Table 4.6 explains.

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 Table 4.7 explains.

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.
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.

### Table 4.8 Touching All Project Files for Recompiling

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Alt-click the Touch column.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Option-click the Touch column.</td>
</tr>
</tbody>
</table>
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 checkmark 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

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Alt-click a red checkmark in the Touch column.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Option-click a red checkmark in the Touch column.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Alt-click a red checkmark in the Touch column.</td>
</tr>
<tr>
<td>Linux</td>
<td>Alt-click a red checkmark in the Touch column.</td>
</tr>
</tbody>
</table>

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.
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.

### Table 4.10 Removing Selected Build Target

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select Edit &gt; Delete</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Edit &gt; Clear</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Edit &gt; Delete</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Edit &gt; Delete</td>
</tr>
</tbody>
</table>

The IDE removes the build target.
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.
   The targetname value changes to reflect the name of the active build target in the project.
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.
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 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.
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,
   • Select a filename from the Group list box, 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.

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.
Working with Files
Managing Files

1. Save all currently opened and modified files, as explained in Table 5.1.

Table 5.1 Saving All Currently Opened and Modified Files

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select File &gt; Save All</td>
</tr>
<tr>
<td>Macintosh</td>
<td>While pressing Option, select File &gt; Save All</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select File &gt; Save All</td>
</tr>
<tr>
<td>Linux</td>
<td>Select File &gt; Save All</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select Window &gt; Close All or Window &gt; Close All Editor Windows.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>While pressing Option, select File &gt; Close All.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select File &gt; Close All or File &gt; Close All Editor Windows</td>
</tr>
<tr>
<td>Linux</td>
<td>Select File &gt; Close All or File &gt; Close All Editor Windows</td>
</tr>
</tbody>
</table>

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 consists of these sections:

- About Dockable Windows
- Working with Dockable Windows
- Dock Bars

**About Dockable Windows**

You can dock certain windows to the edges of the main frame window of the IDE. Table 6.1 explains possible states for dockable windows. Figure 6.1 shows the different window states.

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.
Dockable Windows

About Dockable Windows

Table 6.1  Window States

<table>
<thead>
<tr>
<th>State</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docked</td>
<td>• Attached to the left, right, top, or bottom edge of the client area</td>
</tr>
<tr>
<td></td>
<td>• restricted to the client area</td>
</tr>
<tr>
<td></td>
<td>• resizable</td>
</tr>
<tr>
<td></td>
<td>• has a dock bar instead of a title bar</td>
</tr>
<tr>
<td>Floating</td>
<td>• Rests above all docked windows and MDI child windows</td>
</tr>
<tr>
<td></td>
<td>• movable outside the client area, like a floating palette</td>
</tr>
<tr>
<td></td>
<td>• has a thin title bar</td>
</tr>
<tr>
<td></td>
<td>• does not have Minimize or Maximize buttons</td>
</tr>
<tr>
<td>MDI Child</td>
<td>• Normal child window of the client area, when running in MDI mode</td>
</tr>
<tr>
<td></td>
<td>• restricted to the client area</td>
</tr>
</tbody>
</table>
Dockable Windows
About Dockable Windows

Figure 6.1 Window States

Docked window  MDI child window  Floating 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.
Dockable Windows

Working with Dockable Windows

Table 6.2 Differences between Dockable, Non-Dockable Windows

<table>
<thead>
<tr>
<th>Window Type</th>
<th>Required Criteria</th>
<th>Sample Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dockable</td>
<td>All of these:</td>
<td>• Thread</td>
</tr>
<tr>
<td></td>
<td>• non-modal</td>
<td>• Project</td>
</tr>
<tr>
<td></td>
<td>• resizable</td>
<td>• Component Catalog</td>
</tr>
<tr>
<td></td>
<td>• maximizable</td>
<td></td>
</tr>
<tr>
<td>Non-dockable</td>
<td>Any of these:</td>
<td>• IDE Preferences</td>
</tr>
<tr>
<td></td>
<td>• modal</td>
<td>• Find</td>
</tr>
<tr>
<td></td>
<td>• non-resizable</td>
<td>• About Box</td>
</tr>
<tr>
<td></td>
<td>• non-maximizable</td>
<td></td>
</tr>
</tbody>
</table>

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.
Dockable Windows

Working with Dockable Windows

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.
Dockable Windows

Working with Dockable Windows

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.

---

IDE 5.7 User’s Guide

68
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
Dockable Windows

Dock Bars

- MDI child—unfloat the window so that it becomes an MDI child window
The 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.
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.
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.
Dockable Windows
Dock Bars

1. Click the close button on the dock bar: 

2. The docked window closes.
   Re-opening the window restores its docked position.
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 consists of these sections:

- About Workspaces
- Using Workspaces

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.
2. The IDE closes the current workspace.

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 consists of 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

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.

   - 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

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Press Ctrl-N</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Press Command-N</td>
</tr>
<tr>
<td>Solaris</td>
<td>Press Meta-N</td>
</tr>
<tr>
<td>Linux</td>
<td>Press Meta-N (File &gt; New Text File)</td>
</tr>
</tbody>
</table>
12. Enter source code.
   For example, enter the source code of Listing 8.1.

Listing 8.1 Sample Source Code

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

13. Save the source file, as Table 8.2 explains.

Table 8.2 Saving the Source File

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Press Ctrl-S (File &gt; Save)</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Press Command-S</td>
</tr>
<tr>
<td>Solaris</td>
<td>Press Meta-S</td>
</tr>
<tr>
<td>Linux</td>
<td>Press Meta-S (File &gt; Save)</td>
</tr>
</tbody>
</table>

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.

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:

- CodeWarrior Editor
- Editing Source Code
- Navigating Source Code
- Finding and Replacing Text
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 consists of these sections:

• Editor Window
• Editor Toolbar
• Other Editor Window Components

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 shows the editor window. Table 9.1 explains the items in the editor window.
Figure 9.1 Editor Window

Table 9.1 Editor Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces Menu</td>
<td>![Interface Icon]</td>
<td>Displays a list of referenced interface files or header files for the source file.</td>
</tr>
<tr>
<td>Functions Menu</td>
<td>![Function Icon]</td>
<td>Displays a list of functions defined in the source file.</td>
</tr>
<tr>
<td>Markers Menu</td>
<td>![Marker Icon]</td>
<td>Displays a list of markers defined in the file.</td>
</tr>
<tr>
<td>Document Settings Menu</td>
<td>![Setting Icon]</td>
<td>Displays file-format options and a syntax-coloring toggle.</td>
</tr>
</tbody>
</table>
## Table 9.1 Editor Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version Control System Menu</strong></td>
<td><img src="image" alt="Icon" /></td>
<td>Displays a list of available Version Control System (VCS) commands. Choose a command to apply to the source file.</td>
</tr>
<tr>
<td><strong>Path Caption</strong></td>
<td><img src="image" alt="Path" /></td>
<td>Displays the complete path to the file.</td>
</tr>
<tr>
<td><strong>File Modification Icon</strong></td>
<td><img src="image" alt="Icon" /></td>
<td>This icon indicates an unchanged file since the last save.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Icon" /></td>
<td>This icon indicates a file with modifications not yet saved.</td>
</tr>
<tr>
<td><strong>Breakpoints Column</strong></td>
<td><img src="image" alt="Image" /></td>
<td>Displays breakpoints for the file. Red dot indicates a user-specified breakpoint. Right-click on breakpoints column to bring up context menu.</td>
</tr>
<tr>
<td><strong>Text Editing Area</strong></td>
<td><img src="image" alt="Text" /></td>
<td>Shows the text or source-code content of the file.</td>
</tr>
<tr>
<td><strong>Line and Column Indicator</strong></td>
<td><img src="image" alt="Line and Column" /></td>
<td>Displays the current line and column number of the text-insertion cursor.</td>
</tr>
<tr>
<td><strong>Pane Splitter Controls</strong></td>
<td><img src="image" alt="Pane Splitter" /></td>
<td>Drag to split the window into panes.</td>
</tr>
</tbody>
</table>
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 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 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 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 $\Diamond$ icon indicates an unchanged file since the last Save.

• The $\Box$ 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.
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 consists of these sections:

- Text Manipulation
- Punctuation Balancing
- Code Completion

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.
Editing Source Code

Text Manipulation

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>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Alt-drag the cursor over the portion of text.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Command-drag the cursor over the portion of text.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Alt-drag the cursor over the portion of text.</td>
</tr>
<tr>
<td>Linux</td>
<td>Alt-drag the cursor over the portion of text.</td>
</tr>
</tbody>
</table>

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

```cpp
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

Figure 10.1 shows the Editor Settings. Use these settings to enable or disable the punctuation balancing feature.

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.

**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 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**

<table>
<thead>
<tr>
<th>Host</th>
<th>Get Next Completion</th>
<th>Get Previous Completion</th>
<th>Complete Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Alt-/</td>
<td>Alt-Shift-/</td>
<td>Alt-</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-/</td>
<td>Control-Shift-/</td>
<td>Control-</td>
</tr>
<tr>
<td>Linux/Solaris</td>
<td>Control-/</td>
<td>Control-Shift-/</td>
<td>Control-</td>
</tr>
</tbody>
</table>

**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** explains the items in the Code Completion window. **Table 10.4** explains the icons that appear in the Code Completion list.
Figure 10.2 Code Completion Window

```java
public static void main(String args[]) {
    System.out.println( "Hello World!" );
}
```

Table 10.3 Code Completion Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Completion list</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Disclosure Triangle</td>
<td>![Icon]</td>
<td>Click to toggle display of Documentation pane for programming languages that support it.</td>
</tr>
<tr>
<td>Resize Bar</td>
<td>![Icon]</td>
<td>Drag to resize the Code Completion list and the Documentation pane.</td>
</tr>
<tr>
<td>Documentation pane</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
Editing Source Code

Code Completion

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

   Table 10.5 explains how to navigate the Code Completion list by keyboard.

---

Table 10.4 Code Completion Window Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Code Type</th>
<th>Icon</th>
<th>Code Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>![icon]</td>
<td>Class</td>
<td>![icon]</td>
<td>Method</td>
</tr>
<tr>
<td>![icon]</td>
<td>Function</td>
<td>![icon]</td>
<td>Namespace</td>
</tr>
<tr>
<td>![icon]</td>
<td>Global Variable</td>
<td>![icon]</td>
<td>None</td>
</tr>
<tr>
<td>![icon]</td>
<td>Language Keyword</td>
<td>![icon]</td>
<td>Package</td>
</tr>
<tr>
<td>![icon]</td>
<td>Local Variable</td>
<td>![icon]</td>
<td>Variable</td>
</tr>
<tr>
<td>![icon]</td>
<td>Constant</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Table 10.5 explains how to navigate the Code Completion list by keyboard.
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.
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.
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.
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.
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.
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 consists of these sections:

- Finding Interface Files, Functions, Lines
- Going Back and Forward
- Using Markers
- Symbol Definitions
- Reference Templates (Macintosh)

Finding Interface Files, Functions, 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

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.
Navigating Source Code
Finding Interface Files, Functions, Lines

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 explains how to use the mouse and keyboard to alphabetize functions in the Functions menu.
Navigating Source Code

Going Back and Forward

1. Open the IDE Preferences window.
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

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Ctrl-click the Functions menu.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Option-click the Functions menu.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Alt-click the Functions menu.</td>
</tr>
<tr>
<td>Linux</td>
<td>Alt-click the Functions menu.</td>
</tr>
</tbody>
</table>

Table 11.1 Alphabetizing the Functions list

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.

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.
Navigating Source Code

Using Markers

- 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
Navigating Source Code
Using Markers

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.
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.

Table 11.2 Remove Markers Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Markers list</td>
<td>Displays a list of all markers in the current source file.</td>
</tr>
<tr>
<td>Remove button</td>
<td>Click to remove all selected markers.</td>
</tr>
<tr>
<td>Cancel button</td>
<td>Click to close the Remove Markers window without applying changes.</td>
</tr>
<tr>
<td>Done button</td>
<td>Click to close the Remove Markers window and apply changes.</td>
</tr>
</tbody>
</table>
Navigating Source Code

Symbol Definitions

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.
3. Click **Remove**.
   
   The IDE removes all markers.

Table 11.3  Selecting All Markers in Markers List

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Shift-click each marker name in the list.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select <strong>Edit &gt; Select All.</strong></td>
</tr>
<tr>
<td>Solaris</td>
<td>Select <strong>Edit &gt; Select All.</strong></td>
</tr>
<tr>
<td>Linux</td>
<td>Select <strong>Edit &gt; Select All.</strong></td>
</tr>
</tbody>
</table>

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.
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

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

Inserting a Reference Template

To insert a reference template into your code, follow these steps:
Navigating Source Code
Reference Templates (Macintosh)

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 consists of these sections:

- Single-File Find
- Single-File Find and Replace
- Multiple-File Find and Replace
- Search Results Window
- Text-Selection Find
- Regular-Expression Find
- Comparing Files and Folders

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 explains the items in the Find window.

Figure 12.1 Find Window
# Finding and Replacing Text

## Single-File Find

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find text/list box</td>
<td>Enter a search string. Click the arrow symbol to select a search string that you entered previously.</td>
</tr>
<tr>
<td>Find button</td>
<td>Click to start a search operation using the string in the Find text/list box.</td>
</tr>
<tr>
<td>Find All button</td>
<td>Click to search for all matches in the active editor window.</td>
</tr>
<tr>
<td>Cancel button</td>
<td>Click to close the Find window without performing a search.</td>
</tr>
<tr>
<td>Match whole word checkbox</td>
<td>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.</td>
</tr>
<tr>
<td>Case sensitive checkbox</td>
<td>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.</td>
</tr>
<tr>
<td>Regular expression checkbox</td>
<td>Check to treat the search string as a regular expression. Clear to treat the search string as plain text.</td>
</tr>
<tr>
<td>Stop at end of file checkbox</td>
<td>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.</td>
</tr>
<tr>
<td>Search up checkbox</td>
<td>Check to perform a search operation back from the current selection. Clear to perform a search operation forward of the current selection</td>
</tr>
<tr>
<td>Search selection only checkbox</td>
<td>Check to search only the currently selected text and not the entire file. Clear to search the entire file.</td>
</tr>
<tr>
<td>All text option button</td>
<td>Select to search all text in the file.</td>
</tr>
</tbody>
</table>
Finding and Replacing Text

Single-File Find and Replace

Table 12.1  Find Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code only option button</td>
<td>Select to search only source code in the file.</td>
</tr>
<tr>
<td>Comments only option button</td>
<td>Select to search only comments in the file.</td>
</tr>
</tbody>
</table>

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.
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.
Finding and Replacing Text

Single-File Find and Replace

Figure 12.2  Find and Replace Window

![Find and Replace Window Image]

Table 12.2  Find and Replace Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find text/list box</td>
<td>Enter a search string. Click the arrow symbol to select a search string that you entered previously.</td>
</tr>
<tr>
<td>Replace with text/list box</td>
<td>Enter the replacement string. Click the arrow symbol to select a replacement string that you entered previously.</td>
</tr>
<tr>
<td>Find button</td>
<td>Click to start a search operation using the string in the Find text/list box.</td>
</tr>
<tr>
<td>Replace button</td>
<td>Click to replace the current match with the replacement string.</td>
</tr>
<tr>
<td>Replace All button</td>
<td>Click to replace all matches with the replacement string.</td>
</tr>
<tr>
<td>Cancel button</td>
<td>Click to close the Find and Replace window without performing a search.</td>
</tr>
<tr>
<td>Match whole word checkbox</td>
<td>Check to search for whole-word matches only, ignoring matches within words.</td>
</tr>
<tr>
<td></td>
<td>Clear to search for all matches of the search string, including matches within words.</td>
</tr>
<tr>
<td>Case sensitive checkbox</td>
<td>Check to consider text case during the search. The search operation distinguishes between a capital letter and the same letter in lower case.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

### Table 12.2 Find and Replace Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular expression checkbox</td>
<td>Check to treat the search string as a regular expression. Clear to treat the search string as plain text.</td>
</tr>
<tr>
<td>Stop at end of file checkbox</td>
<td>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.</td>
</tr>
<tr>
<td>Search up checkbox</td>
<td>Check to perform a search operation back from the current selection. Clear to perform a search operation forward of the current selection</td>
</tr>
<tr>
<td>Search selection only checkbox</td>
<td>Check to search only the currently selected text and not the entire file. Clear to search the entire file.</td>
</tr>
<tr>
<td>All text option button</td>
<td>Select to search all text in the file.</td>
</tr>
<tr>
<td>Code only option button</td>
<td>Select to search only source code in the file.</td>
</tr>
<tr>
<td>Comments only option button</td>
<td>Select to search only comments in the file.</td>
</tr>
</tbody>
</table>
Finding and Replacing Text
Multiple-File Find and Replace

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find text/list box</td>
<td>Enter a search string. Click the arrow symbol to select a search string that you entered previously.</td>
</tr>
<tr>
<td>Replace with text/list box</td>
<td>Enter the replacement string. Click the arrow symbol to select a replacement string that you entered previously.</td>
</tr>
<tr>
<td>Find button</td>
<td>Click to start a search operation using the string in the Find text/list box.</td>
</tr>
<tr>
<td>Find All button</td>
<td>Click to search for all matches in the selected items.</td>
</tr>
<tr>
<td>Replace button</td>
<td>Click to replace the current match with the replacement string.</td>
</tr>
<tr>
<td>Replace All button</td>
<td>Click to replace all matches with the replacement string.</td>
</tr>
<tr>
<td>Stop button</td>
<td>Click to stop the current operation.</td>
</tr>
</tbody>
</table>
Finding and Replacing Text
Multiple-File Find and Replace

Table 12.3 Find in Files Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match whole word checkbox</td>
<td>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.</td>
</tr>
<tr>
<td>Case sensitive checkbox</td>
<td>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.</td>
</tr>
<tr>
<td>Regular expression checkbox</td>
<td>Check to treat the search string as a regular expression. Clear to treat the search string as plain text.</td>
</tr>
<tr>
<td>All text option button</td>
<td>Select to search all text in the selected items.</td>
</tr>
<tr>
<td>Code only option button</td>
<td>Select to search only source code in selected items.</td>
</tr>
<tr>
<td>Comments only option button</td>
<td>Select to search only comments in selected items.</td>
</tr>
<tr>
<td>In Folders tab</td>
<td>Click to bring forward the In Folders page. Use this page to search specific folders in the host file system.</td>
</tr>
<tr>
<td>In Projects tab</td>
<td>Click to bring forward the In Projects page. Use this page to search active projects and build targets.</td>
</tr>
<tr>
<td>In Symbolics tab</td>
<td>Click to bring forward the In Symbolics page. Use this page to search files containing symbolics (debugging and browsing) information generated by the IDE.</td>
</tr>
<tr>
<td>In Files tab</td>
<td>Click to bring forward the In Files page. Use this page to search files contained in custom file sets.</td>
</tr>
</tbody>
</table>

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

![Find in Files Window, In Folders Page](image)

Table 12.4 Find in Files Window, In Folders Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search in text/list box</td>
<td>Enter the path to the folder that you want to search. Click the arrow symbol to select a path that you entered previously.</td>
</tr>
<tr>
<td>Browse button</td>
<td>Click to open a dialog box that lets you pick the folder that you want to search.</td>
</tr>
<tr>
<td>Search sub-folders checkbox</td>
<td>Check to search sub-folders of the selected folder. Clear to search the selected folder only, ignoring any sub-folders it may contain.</td>
</tr>
<tr>
<td>By type text/list box</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
Finding and Replacing Text
Multiple-File Find and Replace

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.

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project list box</td>
<td>Specify the projects that you want to search.</td>
</tr>
<tr>
<td>Target list box</td>
<td>Specify the build targets that you want to search.</td>
</tr>
<tr>
<td>Project sources checkbox</td>
<td>Check to search the source-code files of selected projects. Clear to ignore source-code files of selected projects.</td>
</tr>
<tr>
<td>Project headers checkbox</td>
<td>Check to search the header files of selected projects. Clear to ignore header files of selected projects.</td>
</tr>
<tr>
<td>System headers checkbox</td>
<td>Check to search system header files. Clear to ignore system header files.</td>
</tr>
<tr>
<td>Search cached sub-targets checkbox</td>
<td>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.</td>
</tr>
<tr>
<td>File list</td>
<td>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.</td>
</tr>
</tbody>
</table>

Searching for Text across Multiple Projects

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

Multiple-File Find and Replace

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.

   **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.
Finding and Replacing Text
Multiple-File Find and Replace

Figure 12.6  Find in Files Window, In Symbolics Page

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbolics list box</td>
<td>Specify the symbolics files that you want to search.</td>
</tr>
<tr>
<td>Symbolics list</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 12.6  Find in Files Window, In Symbolics Items

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.
   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 explains the items in the page.
Finding and Replacing Text
Multiple-File Find and Replace

Figure 12.7 Find in Files Window, In Files Page

Table 12.7 Find in Files Window, In Files Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Set list box</td>
<td>Specify the file set that you want to search. Select New File Set to create a new set.</td>
</tr>
<tr>
<td>File Set list</td>
<td>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.</td>
</tr>
<tr>
<td>Add Files button</td>
<td>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.</td>
</tr>
<tr>
<td>Clear List button</td>
<td>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.</td>
</tr>
<tr>
<td>Save This Set button</td>
<td>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.</td>
</tr>
<tr>
<td>Remove a Set button</td>
<td>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.</td>
</tr>
</tbody>
</table>

Searching for Text across Multiple Files

Use the In Files page to search for text in file sets.
Finding and Replacing Text

Search Results Window

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.
   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

Table 12.8 Search Results Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result Count text box</td>
<td><img src="image" alt="2" /></td>
<td>Shows the total number of search results.</td>
</tr>
<tr>
<td>Search Criteria text box</td>
<td>![Occurrences of &quot;cout&quot;]</td>
<td>Shows the search criteria.</td>
</tr>
<tr>
<td>Warnings button</td>
<td><img src="image" alt="Warning" /></td>
<td>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.</td>
</tr>
<tr>
<td>Stop button</td>
<td><img src="image" alt="Stop" /></td>
<td>Click to stop the search in progress.</td>
</tr>
</tbody>
</table>
Finding and Replacing Text

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.
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.
Finding and Replacing Text

Regular-Expression Find

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>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>match any</td>
<td>Matches any single printing or non-printing character except newline and null.</td>
</tr>
<tr>
<td>*</td>
<td>match zero or more</td>
<td>Replaces the smallest/preceding regular expression with a sub-expression.</td>
</tr>
</tbody>
</table>
Table 12.9 Recognized Regular-Expression Operators (continued)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>match one or more</td>
<td>Repeats the preceding regular expression at least once and then as many times as necessary to match the pattern.</td>
</tr>
<tr>
<td>?</td>
<td>match zero or one</td>
<td>Repeats the preceding regular expression once or not at all.</td>
</tr>
<tr>
<td>\n</td>
<td>back reference</td>
<td>Refers to a specified group (a unit expression enclosed in parentheses) in the find string. The digit n identifies the n-th group, from left to right, with a number from 1 to 9.</td>
</tr>
<tr>
<td></td>
<td>alteration</td>
<td>Matches one of a choice of regular expressions. If this operator appears between two regular expressions, the IDE matches the largest union of strings.</td>
</tr>
<tr>
<td>^</td>
<td>match beginning of line</td>
<td>Matches items from the beginning of a string or following a newline character. This operator also represents a NOT operator when enclosed within brackets.</td>
</tr>
<tr>
<td>$</td>
<td>match end of line</td>
<td>Matches items from the end of a string or preceding a newline character.</td>
</tr>
<tr>
<td>[...]</td>
<td>list</td>
<td>Defines a set of items to use as a match. The IDE does not allow empty lists.</td>
</tr>
<tr>
<td>(...)</td>
<td>group</td>
<td>Defines an expression to be treated as a single unit elsewhere in the regular expression.</td>
</tr>
<tr>
<td>-</td>
<td>range</td>
<td>Specifies a range. The range starts with the character preceding the operator and ends with the character following the operator.</td>
</tr>
</tbody>
</table>

Table 12.10 shows various examples of using regular expressions to match particular text in a text sample.

Table 12.10 Regular Expression Examples

<table>
<thead>
<tr>
<th>Example Type</th>
<th>This regular expression...</th>
<th>...matches this text...</th>
<th>...in this text sample:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matching simple expressions</td>
<td>ex</td>
<td>ex</td>
<td>sample text</td>
</tr>
<tr>
<td></td>
<td>[[[]].stack[]]</td>
<td>(.stack)</td>
<td>ADDR(.stack)</td>
</tr>
</tbody>
</table>
Finding and Replacing Text
Regular-Expression Find

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 12.10 Regular Expression Examples (continued)

<table>
<thead>
<tr>
<th>Example Type</th>
<th>This regular expression...</th>
<th>...matches this text...</th>
<th>...in this text sample:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matching any character</td>
<td>var.</td>
<td>var1</td>
<td>cout &lt;&lt; var1;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>var2</td>
<td>cout &lt;&lt; var2;</td>
</tr>
<tr>
<td></td>
<td>c.t</td>
<td>cut</td>
<td>cin &gt;&gt; cutF;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cot</td>
<td>cin &gt;&gt; cotG;</td>
</tr>
<tr>
<td>Repeating expressions</td>
<td>s*ion</td>
<td>ion</td>
<td>information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ssion</td>
<td>the session</td>
</tr>
<tr>
<td></td>
<td>s+ion</td>
<td>sion</td>
<td>confusion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ssion</td>
<td>the session</td>
</tr>
<tr>
<td>Grouping expressions</td>
<td>ris</td>
<td>ris</td>
<td>surprise</td>
</tr>
<tr>
<td></td>
<td>r( i)s</td>
<td>r is</td>
<td>theVar is</td>
</tr>
<tr>
<td>Choosing one character from many</td>
<td>[bls]ag</td>
<td>sag bag lag</td>
<td>sagging bag lagged</td>
</tr>
<tr>
<td></td>
<td>[^bls]ag</td>
<td>rag</td>
<td>sagging rag lagged</td>
</tr>
<tr>
<td></td>
<td>[-ab]V</td>
<td>aV -V</td>
<td>aVal-Val;</td>
</tr>
<tr>
<td>Matching line beginnings and endings</td>
<td>^{t}{cout}</td>
<td>cout</td>
<td>cout &lt;&lt; &quot;no tab&quot;;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cout</td>
<td>cout &lt;&lt; &quot;tab&quot;;</td>
</tr>
<tr>
<td></td>
<td>(l*;)$</td>
<td>1;</td>
<td>a-ct; a = battLvl;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>;</td>
<td>b-ct;</td>
</tr>
</tbody>
</table>

IDE 5.7 User’s Guide
Table 12.11 shows examples of using the find string in the replace string of regular expressions.

<table>
<thead>
<tr>
<th>Find string</th>
<th>Replace string</th>
<th>Matching text</th>
<th>After replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>var[0-9]</td>
<td>my_&amp;</td>
<td>var1</td>
<td>my_var1</td>
</tr>
<tr>
<td>tgt</td>
<td>&amp;target</td>
<td>tgt</td>
<td>&amp;target</td>
</tr>
</tbody>
</table>

**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 n-th 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

<table>
<thead>
<tr>
<th>Before replacement</th>
<th>\1 matches this text</th>
<th>\2 matches this text</th>
<th>After replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define var1 10;</td>
<td>var1</td>
<td>10</td>
<td>const int var1 = 10;</td>
</tr>
<tr>
<td>#define a 100;</td>
<td>a</td>
<td>100</td>
<td>const int a = 100;</td>
</tr>
</tbody>
</table>

**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.
Finding and Replacing Text
Comparing Files and 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 shows the Compare Files Setup window. Table 12.13 explains items in the window.

Figure 12.9 Compare Files Setup Window

Table 12.13 Compare Files Setup Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source box</td>
<td>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.</td>
</tr>
<tr>
<td>Destination box</td>
<td>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.</td>
</tr>
<tr>
<td>Case sensitive checkbox</td>
<td>Check to consider text case during the compare operation. The comparison distinguishes between a capital letter and the same letter in lower case. 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.</td>
</tr>
</tbody>
</table>
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.
Finding and Replacing Text
Comparing Files and Folders

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 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 shows the File Compare Results window. Table 12.14 explains the items in the window.
Finding and Replacing Text
Comparing Files and Folders

Figure 12.10  File Compare Results Window

Table 12.14  File Compare Results Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source pane</td>
<td>N/A</td>
<td>Shows the contents of the source file. You cannot edit the contents of this pane.</td>
</tr>
<tr>
<td>Destination pane</td>
<td>N/A</td>
<td>Shows the contents of the destination file. You can edit the contents of this pane.</td>
</tr>
<tr>
<td>Pane resize bar</td>
<td>![Bar Icon]</td>
<td>Drag to resize the Source and Destination panes.</td>
</tr>
<tr>
<td>Apply button</td>
<td>![Apply Icon]</td>
<td>Click to apply the selected Differences pane items to the destination file.</td>
</tr>
<tr>
<td>Unapply button</td>
<td>![Unapply Icon]</td>
<td>Click to unapply the selected Differences pane items from the destination file.</td>
</tr>
</tbody>
</table>
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.
Finding and Replacing Text
Comparing Files and Folders

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.

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.
Finding and Replacing Text
Comparing Files and Folders

Figure 12.11  Folder Compare Results Window

Table 12.15  Folder Compare Results Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pane Expand box</td>
<td>![icon]</td>
<td>Click to enlarge the pane to fill the window.</td>
</tr>
<tr>
<td>Pane Collapse box</td>
<td>![icon]</td>
<td>Click to reduce an expanded pane to its original size.</td>
</tr>
<tr>
<td>Pane resize bar</td>
<td>![icon]</td>
<td>Drag to resize the panes on either side of the bar.</td>
</tr>
<tr>
<td>Files in Both Folders pane</td>
<td>N/A</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files Only in Source</td>
<td>N/A</td>
<td>Shows the items that are in the source folder only.</td>
</tr>
<tr>
<td>pane</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Files Only in</td>
<td>N/A</td>
<td>Shows the items that are in the destination folder only.</td>
</tr>
<tr>
<td>Destination pane</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selected item group</td>
<td>N/A</td>
<td>Shows file and difference information for the selected item in the window</td>
</tr>
<tr>
<td>group</td>
<td></td>
<td>panes.</td>
</tr>
</tbody>
</table>
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 consists of these sections:

- Browser Database
- Browser Symbols

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.
Using the Browser

Browser Database

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 plugin 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.

- **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 plugin 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 plugin 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 plugin 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.

---

**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.
Using the Browser
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 Browser Contextual Menu**

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click a symbol name.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Click and hold on a symbol name.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Click and hold on a symbol name.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on a symbol name.</td>
</tr>
</tbody>
</table>

2. Select a command from the contextual menu.
Identifying Symbols in 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 consists of these sections:

- Class Browser window
- Classes Pane
- Member Functions Pane
- Data Members Pane
- Source Pane
- Status Area

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.

Figure 14.1 shows the Class Browser window. Table 14.1 explains the items in the window. Table 14.2 explains the options in the Browser Access Filters list box.
Using Class Browser Windows

Class Browser window

Figure 14.1 Class Browser Window

![Class Browser Window](image)

Table 14.1 Class Browser Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go Back button</td>
<td>![icon]</td>
<td>Click to return to the preceding browser view.</td>
</tr>
<tr>
<td>Go Forward button</td>
<td>![icon]</td>
<td>Click to move to the succeeding browser view.</td>
</tr>
<tr>
<td>Browser Contents button</td>
<td>![icon]</td>
<td>Click to open the Browser Contents window.</td>
</tr>
<tr>
<td>Class Hierarchy button</td>
<td>![icon]</td>
<td>Click to open the Multi-class Hierarchy window.</td>
</tr>
<tr>
<td>Single Class Hierarchy Window button</td>
<td>![icon]</td>
<td>Click to open the Single-class Hierarchy window for the selected class.</td>
</tr>
</tbody>
</table>
### Using Class Browser Windows

#### Class Browser window

Table 14.1 Class Browser Window Items *(continued)*

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser&lt;br&gt;Access Filters list box</td>
<td><img src="image1.png" alt="View as implementer" /></td>
<td>Select filters for displaying items in class-browser panes.</td>
</tr>
<tr>
<td>Show Inherited</td>
<td><img src="image2.png" alt="Show Inherited" /></td>
<td>Select to show inherited items in the Member Functions Pane and Data Members Pane. Clear to hide inherited items from these panes.</td>
</tr>
<tr>
<td>Classes Pane</td>
<td></td>
<td>Lists all classes in the project browser database.</td>
</tr>
<tr>
<td>Member Functions Pane</td>
<td></td>
<td>Lists all member functions defined in the currently selected class.</td>
</tr>
<tr>
<td>Data Members Pane</td>
<td></td>
<td>Lists all data members defined in the selected class.</td>
</tr>
<tr>
<td>Source Pane</td>
<td></td>
<td>Displays source code for the currently selected item.</td>
</tr>
<tr>
<td>Status Area</td>
<td></td>
<td>Displays various status messages and other information.</td>
</tr>
<tr>
<td>Display toggle buttons&lt;br&gt;Alphabetical</td>
<td><img src="image3.png" alt="Alphabetical" /></td>
<td>Toggles the Classes display between alphabetical and hierarchical listings.</td>
</tr>
<tr>
<td>Hierarchical</td>
<td><img src="image4.png" alt="Hierarchical" /></td>
<td></td>
</tr>
<tr>
<td>New Item button</td>
<td><img src="image5.png" alt="New Item" /></td>
<td>Opens wizards to create new items (e.g., classes, data members, member functions).</td>
</tr>
<tr>
<td>Pane Expand box</td>
<td><img src="image6.png" alt="Pane Expand" /></td>
<td>Expands the pane to the width of the full window.</td>
</tr>
<tr>
<td>Pane Collapse Box</td>
<td><img src="image7.png" alt="Pane Collapse" /></td>
<td>Collapses the pane to its original size.</td>
</tr>
<tr>
<td>Classes Pane button</td>
<td><img src="image8.png" alt="Classes Pane" /></td>
<td>Lists all classes in the project browser database.</td>
</tr>
<tr>
<td>Class Declaration button</td>
<td><img src="image9.png" alt="Class Declaration" /></td>
<td>Opens a window that shows declarations for all classes in the project.</td>
</tr>
</tbody>
</table>
Using Class Browser Windows

Class Browser window

Table 14.1 Class Browser Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open File button</td>
<td></td>
<td>Opens the current source file in a new editor window.</td>
</tr>
<tr>
<td>VCS list pop-up</td>
<td></td>
<td>With a version control system enabled, choose the version-control command to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>execute on the displayed source file.</td>
</tr>
</tbody>
</table>

Table 14.2 BrowserAccess Filters

<table>
<thead>
<tr>
<th>Filter</th>
<th>Icon</th>
<th>Show items with this access:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Public</td>
</tr>
<tr>
<td>View as implementor</td>
<td>![Icon]</td>
<td>*</td>
</tr>
<tr>
<td>View as subclass</td>
<td>![Icon]</td>
<td>*</td>
</tr>
<tr>
<td>View as user</td>
<td>![Icon]</td>
<td>*</td>
</tr>
<tr>
<td>Show public</td>
<td>![Icon]</td>
<td>*</td>
</tr>
<tr>
<td>Show protected</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>Show private</td>
<td>![Icon]</td>
<td></td>
</tr>
</tbody>
</table>

Viewing Class Data from Browser Contents Window

To view class data for a project in the Browser Contents window, follow these steps:
1. Open the **Browser Contents** window, as explained in Table 14.3.

Table 14.3 Opening Browser Contents Window

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Browser Contents.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Browser Contents.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; Browser Contents.</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; Browser Contents.</td>
</tr>
</tbody>
</table>

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 Contextual Menu for Selected Class

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the selected class.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the selected class.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Click and hold on the selected class.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on the selected class.</td>
</tr>
</tbody>
</table>

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

![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.
Using Class Browser Windows

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.

Classes Pane

Use the Classes pane to perform these tasks:

- Create a new class
- Toggle viewing of classes
- Sort classes

Figure 14.1 shows the Classes pane. Table 14.5 explains the items in the pane.
Using Class Browser Windows

Classes Pane

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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Item</td>
<td>![icon]</td>
<td>Click to create a new class using the New Class Wizard.</td>
</tr>
<tr>
<td>Sort Alphabetical</td>
<td>![icon]</td>
<td>Click to sort the Classes list in alphabetical order.</td>
</tr>
<tr>
<td>Sort Hierarchical</td>
<td>![icon]</td>
<td>Click to sort the Classes list in hierarchical order.</td>
</tr>
</tbody>
</table>
Using Class Browser Windows

Member Functions 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

---

**Table 14.6** Member Function, Data Member Identifier Icons

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Icon</th>
<th>The member is…</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>🍌</td>
<td>a static member</td>
</tr>
<tr>
<td>virtual</td>
<td>🍌</td>
<td>a virtual function that can be overridden, or an override of an inherited function</td>
</tr>
<tr>
<td>pure virtual or abstract</td>
<td>🍌</td>
<td>a member function that must be overridden in a subclass to create instances of that subclass</td>
</tr>
</tbody>
</table>
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 in the **Member Functions** pane.
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 shows the Source pane. Table 14.7 explains the items in the pane.

For information on editing source code, see Editing Source Code.

Table 14.7 Source Pane Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open File</td>
<td>![File Icon]</td>
<td>Click to open the current source file in a new editor window.</td>
</tr>
<tr>
<td>VCS menu</td>
<td>![Menu Icon]</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 shows the status area. Table 14.8 explains items in the status area.

Table 14.8 Status Area Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Classes Pane</td>
<td>![Pane Icon]</td>
<td>Click to display the Classes pane in the Class Browser window.</td>
</tr>
<tr>
<td>Hide Classes Pane</td>
<td>![Pane Icon]</td>
<td>Click to hide the Classes pane in the Class Browser window.</td>
</tr>
</tbody>
</table>
## Using Class Browser Windows

### Status Area

**Table 14.8 Status Area Items (continued)**

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Declaration</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to show the declaration of the current class.</td>
</tr>
<tr>
<td>Access Filter Display</td>
<td></td>
<td>Displays the access state of the current class.</td>
</tr>
</tbody>
</table>
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 consists of these sections:

- Multiple-Class Hierarchy Window
- Single-Class Hierarchy Window
- Browser Contents window
- Symbols Window

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 shows the Multi-Class Hierarchy window. Table 15.1 explains the items in the window.
Using Other Browser Windows

Multiple-Class Hierarchy Window

Figure 15.1 Multi-Class Hierarchy Window

![Multi-Class Hierarchy Window](image)

Table 15.1 Multi-Class Hierarchy Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchy Control</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to expand or collapse the subclasses displayed for a specific class.</td>
</tr>
<tr>
<td>Ancestor menu</td>
<td><img src="image" alt="Icon" /></td>
<td>Click and hold on class or subclass box to display a menu. Select a class from menu to display that class.</td>
</tr>
<tr>
<td>Line button</td>
<td><img src="image" alt="Icon" /></td>
<td>Click to toggle the lines that connect classes between diagonal and straight lines.</td>
</tr>
</tbody>
</table>

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.
Using Other Browser Windows
Multiple-Class Hierarchy Window

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>
<thead>
<tr>
<th>Table 15.2 Opening Hierarchy Window</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>On this host...</strong></td>
</tr>
<tr>
<td>Windows</td>
</tr>
<tr>
<td>Macintosh</td>
</tr>
<tr>
<td>Solaris</td>
</tr>
<tr>
<td>Linux</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Table 15.3 Host Platform Graphics-File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host</strong></td>
</tr>
<tr>
<td>Windows</td>
</tr>
<tr>
<td>Macintosh</td>
</tr>
<tr>
<td>Solaris</td>
</tr>
<tr>
<td>Linux</td>
</tr>
</tbody>
</table>

1. Open the Class Hierarchy window.
2. Choose File > Save a Copy As.
3. Save the image to a file.
4. Open the image file in a graphics-processing application.
5. Print the image file.
Using Other Browser Windows

Single-Class Hierarchy Window

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.
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.

*Figure 15.3* shows the Browser Contents window. *Table 15.4* explains the items in the window.
Using Other Browser Windows

Browser Contents window

Figure 15.3 Browser Contents Window

Table 15.4 Browser Contents Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols list box</td>
<td><img src="icon.png" alt="Icon" /></td>
<td>Select the type of symbol to display in the Symbols list.</td>
</tr>
<tr>
<td>Symbols list</td>
<td></td>
<td>Double-click a symbol name to display the source file in a new editor window that defines the symbol.</td>
</tr>
</tbody>
</table>

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
Using Other Browser Windows
Symbols Window

- 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 Browser Contents Window

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Browser Contents</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Browser Contents</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; Browser Contents</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; Browser Contents</td>
</tr>
</tbody>
</table>

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.

Figure 15.4 shows the Symbols window. Table 15.5 explains the items in the window.
Using Other Browser Windows

Symbols Window

Figure 15.4 Symbols Window

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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols toolbar</td>
<td>Provides one-click access to common browser commands and class-filtering commands.</td>
</tr>
<tr>
<td>Symbols pane</td>
<td>Displays a list of all symbols with multiple declarations.</td>
</tr>
<tr>
<td>Source pane</td>
<td>Displays the source code for the currently selected item.</td>
</tr>
</tbody>
</table>
1. Open a contextual menu, as explained in Table 15.7.

Table 15.7 Opening Symbols Window

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the symbol name.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the symbol name.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Click and hold on the symbol name.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on the symbol name.</td>
</tr>
</tbody>
</table>

2. Select **Find all implementations of** 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** 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:

- New Class Wizard
- The New Member Function Wizard
- The New Data Member Wizard

**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.

New Class Wizard

Figure 16.1 shows the New Class wizard Name and Location page. Use this page 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

New Class Wizard

Figure 16.1  New Class Wizard Name, Location

Using the New Class Wizard

To use the New Class Wizard, follow these steps:

1. Open the Class Browser window, as Table 16.1 explains.

Table 16.1  Opening Class Browser Window

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Class Browser</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Class Browser</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; New Class Browser</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; New Class Browser</td>
</tr>
</tbody>
</table>

2. Select Browser > New Class.

IDE 5.7 User’s Guide
Using Browser Wizards

New Class Wizard

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**.

4. Click **Next**. The Base Classes and Methods page (**Figure 16.2**) appears.
Using Browser Wizards
New Class Wizard

Figure 16.2 New Class Wizard Base Class, 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**. The Include Files page (Figure 16.3) appears.
Using Browser Wizards

New Class Wizard

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**. The Targets page (Figure 16.4) appears.
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

Figure 16.5 shows the **New Member Function** wizard Member Function Declaration page. Use this page 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.
Using the New Member Function Wizard

To use the New Member Function wizard, follow these steps:

1. Open the Class Browser window, as Table 16.2 explains.

Table 16.2 Opening Class Browser Window

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Class Browser</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Class Browser</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; New Class Browser</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; New Class Browser</td>
</tr>
</tbody>
</table>

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. The File Locations page (Figure 16.6) appears.

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

Figure 16.7 shows the New Data Member wizard Declaration page. Use this page 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.

Using the New Data Member Wizard

To use the New Data Member wizard, follow these steps:

1. Open the Class Browser window, as Table 16.3 explains.

Table 16.3 Opening Class Browser Window

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Class Browser</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Class Browser</td>
</tr>
</tbody>
</table>
Using Browser Wizards
The New Data Member Wizard

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.  
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.
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
- Profiler
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 consists of these sections:

- About the Debugger
- About Symbolics Files
- Thread Window
- Common Debugging Actions
- Symbol Hint
- Contextual Menus
- Multi-Core Debugging
- External Builds Support

About the 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:

- 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
Working with the Debugger

About Symbolics Files

• 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 symbols 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

The Thread Window is a tool within the IDE used for debugging. It displays the call stack, variables, and source code of the program.

#### Stack

- **Stack Frame 1**: Stack frame details are shown here.
  - Frame: `mainCRTStartup`
  - Address: `0x7E14C7`
- **Stack Frame 2**: Stack frame details are shown here.
  - Frame: `main`
  - Address: `0x7E1494`

#### Variables: Ali, Value, Location

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cout</code></td>
<td>0x0041422C</td>
<td>global</td>
</tr>
<tr>
<td><code>basic_lo</code></td>
<td>0x00414230</td>
<td>0x00414230</td>
</tr>
<tr>
<td><code>ios_base</code></td>
<td>0x00414230</td>
<td>0x00414230</td>
</tr>
<tr>
<td><code>hcsir</code></td>
<td>0x00001002</td>
<td>0x00414254</td>
</tr>
<tr>
<td><code>N1</code></td>
<td><code>&quot;0&quot;</code></td>
<td>0x00414268</td>
</tr>
</tbody>
</table>

#### Source Code

```cpp
#include <iostream>

int main()
{
    using namespace std;
    cout << "Hello World, this is CodeWarrior!" << endl;
    return 0;
}
```

This code snippet demonstrates a simple program that prints a message to the console.
## Working with the Debugger
### Thread Window

### Table 17.1 Thread Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| Debug / Run / Resume button   | ![Icon](image1) | Click to perform these tasks:  
- Continue execution up to the next breakpoint, watchpoint, or eventpoint  
- Run the program until it exits  
- Continue execution of a currently stopped program |
| Stop button                   | ![Icon](image2) | Click to stop (pause) program execution.                                                                                                                                                                   |
| Kill button                   | ![Icon](image3) | Click to terminate program execution and close the Thread window.                                                                                                                                       |
| Step Over button              | ![Icon](image4) | Click to execute the current line, including any routines, and proceed to the next statement.                                                                                                                |
| Step Into button              | ![Icon](image5) | Click to execute the current line, following execution inside a routine.                                                                                                                                   |
| Step Out button               | ![Icon](image6) | Click to continue execution to the end of the current routine, then follow execution to the routine’s caller.                                                                                              |
| Breakpoints button            | ![Icon](image7) | Click to open the Breakpoints window.                                                                                                                                                                    |
| Expressions button            | ![Icon](image8) | Click to open the Expressions window.                                                                                                                                                                   |
| Symbolics button              | ![Icon](image9) | Click to open the Symbolics window.                                                                                                                                                                    |
| Pane Expand box               | ![Icon](image10) | Click to enlarge the pane to fill the window.                                                                                                                                                            |
| Pane Collapse box             | ![Icon](image11) | Click to reduce an expanded pane to its original size.                                                                                                                                                 |
## Working with the Debugger

### Thread Window

Table 17.1 Thread Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pane resize bar</td>
<td><img src="image1.png" alt="Icon" /></td>
<td>Drag to resize the panes on either side of the bar.</td>
</tr>
<tr>
<td>Stack pane</td>
<td></td>
<td>Shows the current routine calling chain, with the most current routine name at the bottom</td>
</tr>
<tr>
<td>Variables pane</td>
<td></td>
<td>Shows local and global variables that the current routine uses.</td>
</tr>
<tr>
<td>Variables Pane Listing button</td>
<td><img src="image2.png" alt="Icon" /></td>
<td>Click this icon to switch among these display states:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- All—show all local and global variables in the code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Auto—show only the local variables of the routine pointed to by the current-statement arrow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- None—show no variables. Use this display state to improve stepping performance for slow remote connections</td>
</tr>
<tr>
<td>Source pane disclosure triangle</td>
<td><img src="image3.png" alt="Icon" /></td>
<td>Click to show or hide the Source pane.</td>
</tr>
<tr>
<td>Source pane</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Source File button</td>
<td><img src="image4.png" alt="Icon" /></td>
<td>Click to edit the contents of the Source pane in an editor window.</td>
</tr>
<tr>
<td>Current-statement arrow</td>
<td><img src="image5.png" alt="Icon" /></td>
<td>Points to statement that debugger will execute next.</td>
</tr>
<tr>
<td>Dash</td>
<td><img src="image6.png" alt="Icon" /></td>
<td>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.</td>
</tr>
</tbody>
</table>
Working with the Debugger
Common Debugging Actions

Table 17.1 Thread Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions list box</td>
<td>![Functions list box icon]</td>
<td>Click to show a list of functions declared in the file. Select a function to highlight it in the Source pane.</td>
</tr>
<tr>
<td>Line and Column button</td>
<td>![Line 20 Col 1]</td>
<td>Shows the current line and column number of the text-insertion cursor. Click to specify a line to show in the Source pane.</td>
</tr>
<tr>
<td>Source list box</td>
<td>![Source]</td>
<td>Click to specify how to display source code in the Source pane:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Source—programming-language statements appear exclusively in the pane</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Assembler—assembly-language instructions appear exclusively in the pane</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Mixed—each programming-language statement shows its corresponding assembly-language instructions</td>
</tr>
</tbody>
</table>

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.

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
Working with the Debugger

Common Debugging Actions

- 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.
Working with the Debugger

Symbol Hint

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. Figure 17.2 shows such a symbol hint.

Select the Show variable values in source code option in the Display Settings preference panel to use the symbol hint.

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.
Working with the Debugger

Contextual Menus

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

Contextual menus provide shortcuts to frequently used menu commands. The available menu commands change, based on the context of the selected item.

Sample uses of contextual menus for debugging 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.
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 Plugins

Data Viewers are plugins that include an user interface to show a custom view of data. These plugins are often platform specific. Data editors are data viewers that also let you modify and write data.

The IDE will keep a registry of plugins that can view particular types. The plugins 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 performs these 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.

Figure 17.4 shows the External Build Target preference panel.
Use this panel to enter 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
Working with the Debugger

External Builds Support

Window. A build output window, such as Figure 17.5, 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.
- 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

Figure 17.6 and Figure 17.7 show the two pages of the External Build wizard. This wizard prompts you for project-creation information 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.
Working with the Debugger

External Builds Support

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**

![External Build Wizard Page 1](image)
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
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 consists of these sections:

- Breakpoints
- Eventpoints
- Watchpoints
- Special 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:
Manipulating Program Execution

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  Breakpoint States

<table>
<thead>
<tr>
<th>State</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>●</td>
<td>Indicates that the breakpoint is currently enabled. The debugger halts program execution at an enabled breakpoint. Click the icon to disable the breakpoint.</td>
</tr>
<tr>
<td>Disabled</td>
<td>○</td>
<td>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.</td>
</tr>
</tbody>
</table>

Breakpoints Window

Use the Breakpoints window to set breakpoints. Figure 18.1 shows this window. Table 18.2 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.
Manipulating Program Execution

Breakpoints

Figure 18.1 Breakpoints Window

Table 18.2 Breakpoints Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Breakpoint Template</td>
<td>![icon]</td>
<td>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.</td>
</tr>
<tr>
<td>Create Breakpoint Group</td>
<td>![icon]</td>
<td>Click to create a new group in the Groups page of the Breakpoints window. Clicking this button is equivalent to clicking Breakpoints &gt; Create Breakpoint Group.</td>
</tr>
</tbody>
</table>

Click an icon in this column to disable or enable a group. Click an icon in this column to disable or enable the Condition associated with the item. Shows projects that the item affects (when it affects more than one project).
### Manipulating Program Execution

#### Breakpoints

<table>
<thead>
<tr>
<th>Table 18.2 Breakpoints Window Items (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Item</strong></td>
</tr>
<tr>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td><strong>Set Default Breakpoint Template</strong></td>
</tr>
<tr>
<td>![Icon]</td>
</tr>
<tr>
<td>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 &gt; Set Default Breakpoint Template with the Breakpoints window frontmost.</td>
</tr>
<tr>
<td><strong>Rename Breakpoint</strong></td>
</tr>
<tr>
<td>![Icon]</td>
</tr>
<tr>
<td>Click to rename the selected item in the Breakpoints window. Clicking this button is equivalent to clicking Breakpoints &gt; Rename Breakpoint with the Breakpoints window frontmost.</td>
</tr>
<tr>
<td><strong>Breakpoint Properties</strong></td>
</tr>
<tr>
<td>![Icon]</td>
</tr>
<tr>
<td>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 &gt; Breakpoint Properties with the Breakpoints window frontmost.</td>
</tr>
<tr>
<td><strong>Groups tab</strong></td>
</tr>
<tr>
<td>![Icon]</td>
</tr>
<tr>
<td>Click to display the Groups page. This page lets you work with breakpoints, eventpoints, watchpoints, and internal breakpoints.</td>
</tr>
<tr>
<td><strong>Instances tab</strong></td>
</tr>
<tr>
<td>![Icon]</td>
</tr>
<tr>
<td>Click to display the Instances page. This page lets you set breakpoints, eventpoints, and watchpoints on a per-thread or per-process basis.</td>
</tr>
<tr>
<td><strong>Templates tab</strong></td>
</tr>
<tr>
<td>![Icon]</td>
</tr>
<tr>
<td>Click to display the Templates page. This page lets you define breakpoint templates and specify a default breakpoint template.</td>
</tr>
</tbody>
</table>
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.
Manipulating Program Execution

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

![Breakpoints column](image)

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakpoint Type</td>
<td>The type of item, such as Auto Breakpoint.</td>
</tr>
<tr>
<td>Serial number</td>
<td>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 plugins use to identify the item.</td>
</tr>
<tr>
<td>Condition</td>
<td>The conditional expression associated with the item. This conditional expression must evaluate to true in order for the item to perform its specified action.</td>
</tr>
<tr>
<td>Hit Count</td>
<td>Displays the number of times that program execution arrived at the breakpoint before the program stopped.</td>
</tr>
<tr>
<td>File-Info</td>
<td>The path to the file that contains the item.</td>
</tr>
<tr>
<td>Name</td>
<td>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.</td>
</tr>
<tr>
<td>Original Process</td>
<td>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.</td>
</tr>
<tr>
<td>Original-Target</td>
<td>The path to the build target that contains the item.</td>
</tr>
<tr>
<td>Times Hit</td>
<td>The number of times that this item affected program execution.</td>
</tr>
<tr>
<td>Times Left</td>
<td>The number of times remaining for this item to affect program execution.</td>
</tr>
<tr>
<td>Thread</td>
<td>The thread in which the item takes effect.</td>
</tr>
<tr>
<td>Hardware</td>
<td>The hardware on which to use the item. For example, set this property to <strong>Prefer Hardware</strong> to specify that the breakpoint is a hardware breakpoint.</td>
</tr>
</tbody>
</table>
Manipulating Program Execution

Breakpoints

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.

- 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.
Manipulating Program Execution

Breakpoints

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.
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

The CodeWarrior 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.
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.
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 18.4 Eventpoints

<table>
<thead>
<tr>
<th>Eventpoint</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Point</td>
<td>🗒️</td>
<td>Logs or speaks a string or expression and records messages to the Log window</td>
</tr>
<tr>
<td>Pause Point</td>
<td>🔔</td>
<td>Pauses execution long enough to refresh debugger data</td>
</tr>
<tr>
<td>Script Point</td>
<td>⚙️</td>
<td>Runs a script, application, or other item</td>
</tr>
<tr>
<td>Skip Point</td>
<td>⏯️</td>
<td>Skips execution of a line of source code</td>
</tr>
<tr>
<td>Sound Point (Windows OS)</td>
<td>🔊</td>
<td>Plays a sound</td>
</tr>
<tr>
<td>Trace Collection Off</td>
<td>⏸️</td>
<td>Stops collecting trace data for the Trace window</td>
</tr>
<tr>
<td>Trace Collection On</td>
<td>🎶</td>
<td>Starts collecting trace data for the Trace window</td>
</tr>
</tbody>
</table>

### Table 18.5 Eventpoint States

<table>
<thead>
<tr>
<th>State</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>🗒️</td>
<td>Indicates that the eventpoint is currently enabled. The debugger performs the specified task at an enabled eventpoint. Click the icon to disable the eventpoint.</td>
</tr>
<tr>
<td>Disabled</td>
<td>⏸️</td>
<td>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.</td>
</tr>
</tbody>
</table>
Manipulating Program Execution
Eventpoints

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 (Figure 18.3) appears.

Figure 18.3 Log Point Settings Window

3. Enter the text of your log message in the Message text box.
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
   - 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*. 
Manipulating Program Execution
Eventpoints

---

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(Figure 18.4) appears.

Figure 18.4 Script Point Settings Window

---

IDE 5.7 User's Guide
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:
Manipulating Program Execution

Eventpoints

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 (Figure 18.5) appears.

Figure 18.5  Sound Point Settings Window

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.
Manipulating Program Execution
Eventpoints

**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 recreating 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). 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:
Manipulating Program Execution

Watchpoints

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.
   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.
Manipulating Program Execution

Watchpoints

Table 18.6 Watchpoint States

<table>
<thead>
<tr>
<th>State</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td></td>
<td>Indicates that the watchpoint is currently enabled. The debugger halts program execution at an enabled watchpoint. Click the icon to disable the watchpoint.</td>
</tr>
<tr>
<td>Disabled</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
Manipulating Program Execution

Watchpoints

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:
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.
Manipulating Program Execution
Watchpoints

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.

   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.
Manipulating Program Execution

Special Breakpoints

3. In the Groups or Instances pages of the Breakpoints window, find the watchpoint that you want to associate with a conditional expression.

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 watchpoint.

**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 plugin 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.

![Active icon](image1.png)

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.

![Inactive icon](image2.png)

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 consists of these sections:

- Global Variables Window
- Variable Window
- Expressions Window

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 shows the Global Variables window. Table 19.1 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>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.</td>
</tr>
<tr>
<td>Variables</td>
<td>Lists variables according to the file selected in the File pane. Double-click a variable to display it in a separate Variable window.</td>
</tr>
</tbody>
</table>

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.
Working with Variables

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.

Alternatively, use a contextual menu to open a variable window, as Table 19.2 explains.
Working with Variables

Variable Window

Table 19.2 Opening a Variable Window by Using a Contextual Menu

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the variable and select View Variable.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the variable and select View Variable.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Click and hold on the variable, then select View Variable.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on the variable, then select View Variable.</td>
</tr>
</tbody>
</table>

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 that Listing 19.1 defines.

Listing 19.1 Sample Structure Definition

```c
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 Listing 19.2 shows.

Listing 19.2 Sample Variable Format Definition

```xml
<variableformats>
  <variableformat>
    <osname>osWin32</osname>
    <runtimename>runtimeWin32</runtimename>
    <typename>Rect</typename>
    <expression>
      "T: " + ^var.top + 
      "L: " + ^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 Format XML Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>variableformats</td>
<td>A group of variable format records.</td>
</tr>
<tr>
<td>variableformat</td>
<td>An individual variable format record.</td>
</tr>
<tr>
<td>osname</td>
<td>The operating system that defines the scope of this record.</td>
</tr>
<tr>
<td>runtimename</td>
<td>The runtime that defines the scope of this record.</td>
</tr>
<tr>
<td>typename</td>
<td>The name of the Type that this record will format.</td>
</tr>
<tr>
<td>expression</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Expressions Window

The Expressions window (**Figure 19.3**) helps you monitor and manipulate these kinds of items:

- global and local variables
- structure members
Working with Variables

Expressions Window

- array elements

Table 19.4 explains items of this window.

Figure 19.3 Expressions Window

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IpCmdLine</td>
<td>0x00140F6F</td>
</tr>
<tr>
<td>*IpCmdLine</td>
<td>0x00</td>
</tr>
<tr>
<td>hPrevInstance</td>
<td>0x00000000</td>
</tr>
<tr>
<td>hInstance</td>
<td>0x00000000</td>
</tr>
<tr>
<td>nCmdShow</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 19.4 Expressions Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression column</td>
<td>Lists expressions and expression hierarchies. Click the hierarchical controls to expand or collapse the expression view.</td>
</tr>
<tr>
<td>Value column</td>
<td>Shows the current value of each corresponding expression. Double-click a value to change it.</td>
</tr>
</tbody>
</table>

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.

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:
Working with Variables
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 Memory

This chapter explains how to work with memory in a CodeWarrior™ IDE debugging session. The following windows show various types of information about memory:

- 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 consists of these sections:

- Memory Window
- Array Window
- Registers Window
- Register Details Window (Windows OS)

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

**NOTE** 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 explains the items in the window.
Working with Memory

Memory Window

Figure 20.1 Memory Window

Table 20.1 Memory Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td></td>
<td>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>.</td>
</tr>
<tr>
<td>View</td>
<td><img src="image" alt="View: Raw data" /></td>
<td>Select the data format in which to view memory contents.</td>
</tr>
<tr>
<td>Memory Space (for processors that support multiple memory spaces)</td>
<td></td>
<td>Choose the memory space in which to view selected variables or source code.</td>
</tr>
<tr>
<td>Previous Memory Block</td>
<td><img src="image" alt="Previous" /></td>
<td>Click to view the preceding block of memory.</td>
</tr>
<tr>
<td>Next Memory Block</td>
<td><img src="image" alt="Next" /></td>
<td>Click to view the succeeding block of memory.</td>
</tr>
<tr>
<td>Address</td>
<td></td>
<td>Displays a contiguous range of memory addresses, beginning with the address entered in the Display field.</td>
</tr>
</tbody>
</table>
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.
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.

Table 20.1 Memory Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td></td>
<td>Displays a hexadecimal representation of the memory addresses shown in the Address pane.</td>
</tr>
<tr>
<td>Ascii</td>
<td></td>
<td>Displays an ASCII representation of the memory addresses shown in the Address pane.</td>
</tr>
<tr>
<td>Word Size</td>
<td>Word Size: 32</td>
<td>Select the bit size of displayed words.</td>
</tr>
<tr>
<td>Page (for processors that support multiple pages)</td>
<td>Page: X Memory</td>
<td>Select the memory-space page in which to view source code.</td>
</tr>
</tbody>
</table>
Working with Memory

Memory Window

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

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the item and select View Memory.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the item and select View Memory.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Click and hold on the item, then select View Memory.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on the item, then select View Memory.</td>
</tr>
</tbody>
</table>

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.

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 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>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x1001EB9C</td>
<td>0x1001EED0</td>
</tr>
<tr>
<td>1</td>
<td>0x1001EBF4</td>
<td>0x1001EED0</td>
</tr>
<tr>
<td>2</td>
<td>0x1001EC88</td>
<td>0x1001EED0</td>
</tr>
<tr>
<td>3</td>
<td>0x1001ECC3</td>
<td>0x1001EED0</td>
</tr>
<tr>
<td>4</td>
<td>0x1001EC7C</td>
<td>0x1001EED0</td>
</tr>
<tr>
<td>5</td>
<td>0x1001ED04</td>
<td>0x1001EED0</td>
</tr>
</tbody>
</table>

Table 20.3 Array Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical control</td>
<td>![Hierarchical control icon]</td>
<td>Click to collapse the view of the information pane.</td>
</tr>
<tr>
<td>Bind To</td>
<td></td>
<td>Select the base address of the array: Address, Variable, or Register.</td>
</tr>
</tbody>
</table>
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 Table 20.4 explains.

Table 20.4 Opening an Array Window by Using a Contextual Menu

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the array and select View Array.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the array and select View Array.</td>
</tr>
</tbody>
</table>
Working with Memory

Registers Window

Table 20.4  Opening an Array Window by Using a Contextual Menu (continued)

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>Click and hold on the array, then select View Array.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on the array, then select View Array.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Windows PC</td>
<td></td>
</tr>
<tr>
<td>C++ Console Debug.exe</td>
<td></td>
</tr>
<tr>
<td>Thread 0x458</td>
<td></td>
</tr>
<tr>
<td>General Registers</td>
<td></td>
</tr>
<tr>
<td>EAX</td>
<td>0x00416010</td>
</tr>
<tr>
<td>ECX</td>
<td>0x0041801C</td>
</tr>
<tr>
<td>EDX</td>
<td>0x00403AC0</td>
</tr>
<tr>
<td>EBX</td>
<td>0x7FFDF000</td>
</tr>
<tr>
<td>ESP</td>
<td>0x0012FF90</td>
</tr>
<tr>
<td>EBP</td>
<td>0x0012FFC0</td>
</tr>
<tr>
<td>ESI</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
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.

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>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Registers</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Registers Window</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; Registers Window</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; Registers Window</td>
</tr>
</tbody>
</table>

Viewing Registers
View registers to inspect and modify their contents.
Working with Memory

Registers Window

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 Table 20.6 explains.
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.

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 shows the Register Details window. Table 20.7 explains items in the window.

Table 20.6 Changing Data Format by Using a Contextual Menu

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the register value and select View as format.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the register value and select View as format.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Click and hold on the register value and select View as format.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on the register value and select View as format.</td>
</tr>
</tbody>
</table>
Working with Memory
Register Details Window (Windows OS)

Figure 20.4  Register Details Window

Table 20.7  Register Details Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description File text box</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Register Name</td>
<td></td>
<td>Shows the name of the register depicted in the window.</td>
</tr>
<tr>
<td>Address text box</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Format list box</td>
<td></td>
<td>Specify the data format for bit values in the Register Display:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Character</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Decimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Unsigned Decimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Hexadecimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Default—have the IDE determine the best format</td>
</tr>
</tbody>
</table>
## Table 20.7 Register Details Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register Display</strong></td>
<td><img src="image" alt="Register Display Icon" /></td>
<td>Shows a depiction of the register that you specify in the Description File text box, including individual register bits and their values.</td>
</tr>
<tr>
<td>Bitfield Name list box</td>
<td><img src="image" alt="Bitfield Name List Box Icon" /></td>
<td>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.</td>
</tr>
<tr>
<td>Bit Value text box</td>
<td><img src="image" alt="Bit Value Text Box Icon" /></td>
<td>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.</td>
</tr>
<tr>
<td>Bit Value Modifier list box</td>
<td><img src="image" alt="Bit Value Modifier List Box Icon" /></td>
<td>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.</td>
</tr>
<tr>
<td>Description</td>
<td><img src="image" alt="Description Icon" /></td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

---

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revert button</td>
<td></td>
<td>Click to change a modified value in the Register Display to its original value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you clicked the Write button to write a new value to the register, you cannot revert that value.</td>
</tr>
<tr>
<td>Read button</td>
<td></td>
<td>Click to have the Register Display reflect current bit values from the register itself.</td>
</tr>
<tr>
<td>Write button</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Reset Value button</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Text View list box</td>
<td></td>
<td>Use to specify information that appears in the Description portion of the window.</td>
</tr>
</tbody>
</table>
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
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
- System Browser Window
- Log Window

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 list. Symbolics information will be generated during the next build or debugging session.

To view the Symbolics window, start a CodeWarrior debug session (Project > Debug from CodeWarior menu bar), then select View > Symbolics.

Figure 21.1 shows the Symbolics window. Table 21.1 explains items in the window.
Working with Debugger Data
Symbolics Window

Figure 21.1 Symbolics Window

Table 21.1 Symbolics Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger toolbar</td>
<td>![Debugger toolbar icon]</td>
<td>Contains buttons that represent common debugging commands, such as stepping through code.</td>
</tr>
<tr>
<td>Executables pane</td>
<td>![Executables pane icon]</td>
<td>Lists recently used executable files that contain symbolics information.</td>
</tr>
<tr>
<td>Files pane</td>
<td>![Files pane icon]</td>
<td>Lists source files in build target being debugged, for selected executable file.</td>
</tr>
<tr>
<td>Functions pane</td>
<td>![Functions pane icon]</td>
<td>Lists functions declared in the file selected in the Files pane.</td>
</tr>
<tr>
<td>Source pane</td>
<td>![Source pane icon]</td>
<td>Displays source code in the file selected in the Files pane.</td>
</tr>
</tbody>
</table>
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.

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** explains items in the window.

**NOTE** If the System Browser window does not show processes for a specific machine, you must 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

Table 21.2 System Browser Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach to Process</td>
<td>![Icon]</td>
<td>Click to have the debugger control the selected process.</td>
</tr>
<tr>
<td>Stack Crawl window</td>
<td>![Icon]</td>
<td>Click to open a Thread window for the selected process.</td>
</tr>
<tr>
<td>Refresh</td>
<td>![Icon]</td>
<td>This icon indicates that information for selected item is periodically updated. Click this icon to toggle between refresh and no refresh.</td>
</tr>
<tr>
<td>No Refresh</td>
<td>![Icon]</td>
<td>This icon indicates that information is not updated for the selected item.</td>
</tr>
<tr>
<td>Expand</td>
<td>![Icon]</td>
<td>Click to expand a process and list related tasks.</td>
</tr>
</tbody>
</table>

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.
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:

<table>
<thead>
<tr>
<th>Menu Bar Layout</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; System</td>
</tr>
<tr>
<td>Macintosh (also applies to Linux/Solaris)</td>
<td>Select Window &gt; System Windows</td>
</tr>
</tbody>
</table>
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 Hardware Tools

This chapter explains the CodeWarrior™ IDE hardware tools. Use these tools for board bring-up, test, and analysis.

NOTE  Not all products support all the IDE features this chapter describes, such as the Flash programmer window, hardware diagnostic window, and logic analyzer. Some screen captures in this chapter were taken on a Windows PC; their actual appearance varies slightly on other host platforms.

This chapter consists of these sections:

- Flash Programmer Window
- Hardware Diagnostics Window
- Working with a Logic Analyzer
- Trace Window
- Cache Window
- Profile Window
- Command Window

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 shows the Flash Programmer window. Table 22.1 explains the items in the window.

To open the Flash Programmer window, click Tools > Flash Programmer.

The Flash Programmer window contains these panels:

- Target Configuration
- Flash Configuration
- Program / Verify
Working with Hardware Tools

Flash Programmer Window

- Erase / Blank Check
- Checksum

Figure 22.1 Flash Programmer Window

Table 22.1 Flash Programmer Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Programmer pane</td>
<td>Shows a list of panel names. Click a panel name to display that panel.</td>
</tr>
<tr>
<td>Show Log</td>
<td>Click to display a text file that logs flash programmer actions. Check the Enable Logging checkbox in the Options group to enable this button.</td>
</tr>
<tr>
<td>Load Settings</td>
<td>Click to restore previously saved settings for the current panel.</td>
</tr>
<tr>
<td>Save Settings</td>
<td>Click to save settings for the current panel to a file.</td>
</tr>
</tbody>
</table>
Target Configuration

The Target Configuration panel configures general flash programmer settings. Figure 22.2 shows the Target Configuration panel. Table 22.2 explains items in the panel.

Table 22.1 Flash Programmer Window Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Click to save changes to all panels and close the window.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Click to discard changes to all panels and close the window.</td>
</tr>
</tbody>
</table>

Target Configuration Panel

Figure 22.2 Target Configuration Panel
### Table 22.2 Target Configuration Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Project</td>
<td>Shows the current default project in the IDE.</td>
</tr>
<tr>
<td>Default Target</td>
<td>Shows the default build target. Clear the <strong>Use Custom Settings</strong> checkbox to have the IDE use connection settings from the build target for connecting to the hardware.</td>
</tr>
<tr>
<td>Use Custom Settings checkbox.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Connection information includes the information that you specify in the <strong>Target Processor</strong> list box, the <strong>Connection</strong> list box, and the <strong>Use Target Initialization</strong> text box.</td>
</tr>
<tr>
<td>Target Processor text/list box</td>
<td>Use to specify the hardware processor.</td>
</tr>
<tr>
<td>Connection list box</td>
<td>Use to specify the method that the IDE uses to connect to the hardware.</td>
</tr>
<tr>
<td>Use Target Initialization checkbox and text box</td>
<td>Check to specify an initialization file for the hardware connection. Enter the initialization file path in the text box, or click the <strong>Browse</strong> button to open a dialog box that you can use to specify the initialization file path.</td>
</tr>
<tr>
<td></td>
<td>Clear if you do not want to use an initialization file for the hardware connection.</td>
</tr>
<tr>
<td>Target Memory Buffer Address text box</td>
<td>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).</td>
</tr>
<tr>
<td></td>
<td>The flash programmer should not modify any memory location other than the target memory buffer and flash memory.</td>
</tr>
<tr>
<td></td>
<td>For example, the flash programmer uses the target memory buffer to download and execute the flash device driver.</td>
</tr>
</tbody>
</table>
Working with Hardware Tools
Flash Programmer Window

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 explains the items in the panel.

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Memory Buffer Size text box</td>
<td>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. The flash programmer should not modify any memory location other than the target memory buffer and flash memory.</td>
</tr>
<tr>
<td>Enable Logging checkbox</td>
<td>Check to have the IDE generate detailed status information during flash operations. Checking this checkbox enables the Show Log button. Clear to disable logging of detailed status information during flash operations. Clearing this checkbox disables the Show Log button. Click the Show Log button to view the status information.</td>
</tr>
<tr>
<td>Verify Target Memory Writes checkbox</td>
<td>Check to have the IDE verify all write operations to the hardware RAM by reading the result of each write operation. Clear to have the IDE perform write operations without verifying them.</td>
</tr>
</tbody>
</table>
### Working with Hardware Tools

#### Flash Programmer Window

*Figure 22.3 Flash Configuration Panel*

![Flash Configuration Panel](image)

*Table 22.3 Flash Configuration Panel Items*

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Memory Base Address text box</td>
<td>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.</td>
</tr>
<tr>
<td>Device pane</td>
<td>Shows an alphabetical list of supported flash device types. Select a device type from this pane. Your selection determines the contents of the <strong>Organization</strong> and <strong>Sector Address Map</strong> panes.</td>
</tr>
</tbody>
</table>
Program / Verify

The **Program / Verify** panel lets you program an image into flash memory and verify the programming operation. Figure 22.4 shows the Program / Verify panel. Table 22.4 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Selected File</td>
<td>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. Clear to have the IDE program flash memory with the file that the default build target determines. 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.</td>
</tr>
<tr>
<td>File Type</td>
<td>Select the file type. Options are: Auto Detect, Binary/Raw Format, Elf Format, or Motorola S-Record Format.</td>
</tr>
</tbody>
</table>
Working with Hardware Tools

Flash Programmer Window

Table 22.4 Program / Verify Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrict Address Range</td>
<td>Check to use the <strong>Start</strong> and <strong>End</strong> 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. Clear to have the IDE determine the address range in which to program flash data.</td>
</tr>
<tr>
<td>Start text box</td>
<td>Enter the starting address of the range that you want the flash programmer to use for programming flash data. Check the <strong>Restrict Address Range</strong> checkbox to enable this text box.</td>
</tr>
<tr>
<td>End text box</td>
<td>Enter the ending address of the range that you want the flash programmer to use for programming flash data. Check the <strong>Restrict Address Range</strong> checkbox to enable this text box.</td>
</tr>
<tr>
<td>Apply Address Offset</td>
<td>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. 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.</td>
</tr>
<tr>
<td>Offset text box</td>
<td>Enter the offset to add to the starting address that the file specifies. The flash programmer begins programming flash data at the resulting address. Check the <strong>Apply Address Offset</strong> checkbox to enable this text box.</td>
</tr>
<tr>
<td>Flash Base Address</td>
<td>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 <strong>Flash Memory Base Address</strong> text box of the <strong>Flash Configuration</strong> panel.</td>
</tr>
<tr>
<td>Flash Base + Offset</td>
<td>Shows the resulting address of adding the offset value that you specify in the <strong>Offset</strong> text box to the <strong>Flash Base Address</strong> value. The flash programmer begins programming flash data at this resulting address.</td>
</tr>
</tbody>
</table>
Working with Hardware Tools

Flash Programmer Window

Table 22.4  Program / Verify Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Shows flash programmer progress information. Click the Details button to show more thorough progress information.</td>
</tr>
<tr>
<td>Program button</td>
<td>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.</td>
</tr>
<tr>
<td>Verify button</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 explains items in the panel.
Figure 22.5 Erase / Blank Check Panel

Table 22.5 Erase / Blank Check Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Sectors checkbox and list</td>
<td>Check to apply the erase or blank check operation to the entire flash memory.</td>
</tr>
<tr>
<td></td>
<td>Clear to specify sectors that you want to erase or check for blanks. Select sectors in the list below the checkbox.</td>
</tr>
<tr>
<td>Erase Sectors Individually</td>
<td>Check to have the flash programmer ignore chip erase commands and erase each individual sector instead.</td>
</tr>
<tr>
<td></td>
<td>Clear to have the flash programmer obey chip erase commands and erase all sectors at once.</td>
</tr>
<tr>
<td></td>
<td>Check the All Sectors checkbox to enable this checkbox.</td>
</tr>
<tr>
<td>Status</td>
<td>Shows flash programmer progress information. Click the Details button to show more thorough progress information.</td>
</tr>
</tbody>
</table>
Checksum

The **Checksum** panel lets you calculate checksum values. Figure 22.6 shows the Checksum panel. Table 22.6 explains items in the panel.

### Figure 22.6 Checksum Panel

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erase button</td>
<td>Click to have the flash programmer erase the sectors that you specified. The <strong>Status</strong> reflects flash programmer progress.</td>
</tr>
</tbody>
</table>
| Blank Check button | Click to have the flash programmer perform these tasks:  
  - 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. |
### Table 22.6 Checksum Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File on Target</td>
<td>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.</td>
</tr>
<tr>
<td>File on Host</td>
<td>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.</td>
</tr>
<tr>
<td>Memory Range on Target</td>
<td>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.</td>
</tr>
<tr>
<td>Entire Flash</td>
<td>Select to have the flash programmer read the entire contents of flash memory. The flash programmer uses this data for the checksum operation.</td>
</tr>
<tr>
<td>Start text box</td>
<td>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.</td>
</tr>
<tr>
<td>Size text box</td>
<td>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.</td>
</tr>
<tr>
<td>Status</td>
<td>Shows flash programmer progress information. Click the Details button to show more thorough progress information.</td>
</tr>
<tr>
<td>Calculate Checksum</td>
<td>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.</td>
</tr>
</tbody>
</table>
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** shows the Hardware Diagnostics window. **Table 22.7** 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**

**Figure 22.7**  Hardware Diagnostics window

![Hardware Diagnostics window](image)
Table 22.7 Hardware Diagnostics Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Diagnostics pane</td>
<td>Shows a list of panel names. Click a panel name to display that panel in the Hardware Diagnostics window.</td>
</tr>
<tr>
<td>Load Settings</td>
<td>Click to restore previously saved settings for the current panel.</td>
</tr>
<tr>
<td>Save Settings</td>
<td>Click to save settings for the current panel to a file.</td>
</tr>
<tr>
<td>OK button</td>
<td>Click to save changes to all panels and close the window.</td>
</tr>
<tr>
<td>Cancel button</td>
<td>Click to discard changes to all panels and close the window.</td>
</tr>
</tbody>
</table>

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
Working with Hardware Tools

Hardware Diagnostics Window

Table 22.8 Configuration Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Project</td>
<td>Shows the current default project in the IDE.</td>
</tr>
<tr>
<td>Default Target</td>
<td>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.</td>
</tr>
<tr>
<td>Use Custom Settings checkbox.</td>
<td>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.</td>
</tr>
<tr>
<td>Target Processor text/list box</td>
<td>Use to specify the hardware processor.</td>
</tr>
<tr>
<td>Connection list box</td>
<td>Use to specify the method that the IDE uses to connect to the hardware.</td>
</tr>
<tr>
<td>Use Target Initialization checkbox and text box</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 shows the Memory Read / Write panel. Table 22.9 explains items in the panel.
Working with Hardware Tools

Hardware Diagnostics Window

Figure 22.9 Memory Read / Write Panel

Table 22.9 Memory Read / Write Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Select to have the hardware diagnostic tools perform read tests.</td>
</tr>
<tr>
<td>Write</td>
<td>Select to have the hardware diagnostic tools perform write tests.</td>
</tr>
<tr>
<td>Byte</td>
<td>Select to have the hardware diagnostic tools perform byte-size operations.</td>
</tr>
<tr>
<td>Word</td>
<td>Select to have the hardware diagnostic tools perform word-size operations.</td>
</tr>
<tr>
<td>Long Word</td>
<td>Select to have the hardware diagnostic tools perform long-word-size operations.</td>
</tr>
<tr>
<td>Target Address</td>
<td>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).</td>
</tr>
</tbody>
</table>
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 shows the Scope Loop panel. Table 22.10 explains items in the panel.

After the first 1000 operations, the Status shows the estimated time between operations.

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value to Write</td>
<td>Specify the value that the hardware diagnostic tools write during testing.</td>
</tr>
<tr>
<td></td>
<td>Select the Write option to enable this text box.</td>
</tr>
<tr>
<td>Status</td>
<td>Shows hardware diagnostic progress information. Click the Details button to show more progress information.</td>
</tr>
<tr>
<td>Access Target button</td>
<td>Click to have the hardware diagnostic tools perform specified tests.</td>
</tr>
<tr>
<td></td>
<td>The Status shows test results.</td>
</tr>
</tbody>
</table>

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.
Working with Hardware Tools

Hardware Diagnostics Window

Figure 22.10 Scope Loop Panel

Table 22.10 Scope Loop Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Select to have the hardware diagnostic tools perform read tests.</td>
</tr>
<tr>
<td>Write</td>
<td>Select to perform write tests.</td>
</tr>
<tr>
<td>Byte</td>
<td>Select to have the hardware diagnostic tools perform byte-size operations.</td>
</tr>
<tr>
<td>Word</td>
<td>Select to perform word-size operations.</td>
</tr>
<tr>
<td>Long Word</td>
<td>Select to perform long-word-size operations.</td>
</tr>
<tr>
<td>Target Address</td>
<td>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).</td>
</tr>
<tr>
<td>Value to Write</td>
<td>Specify the value that the hardware diagnostic tools write during testing. Select the Write option to enable this text box.</td>
</tr>
</tbody>
</table>
Working with Hardware Tools
Hardware Diagnostics Window

Table 22.10  Scope Loop Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed slider</td>
<td>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.</td>
</tr>
<tr>
<td>Status</td>
<td>Shows hardware diagnostic progress information. Click the Details button to show more thorough progress information.</td>
</tr>
<tr>
<td>Begin Scope Loop button</td>
<td>Click to have the hardware diagnostic tools perform your specified tests. The Status shows test results.</td>
</tr>
</tbody>
</table>

Memory Tests

The Memory Tests panel lets you perform three different tests on the hardware:

- Walking Ones
- Address
- Bus Noise

Figure 22.11 shows the Memory Tests panel. Table 22.11 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.
Figure 22.11 Memory Tests Panel

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walking 1’s</td>
<td>Check to have the hardware diagnostic tools perform the Walking Ones test.</td>
</tr>
<tr>
<td></td>
<td>Clear to have the diagnostic tools skip the Walking Ones test.</td>
</tr>
<tr>
<td>Address</td>
<td>Check to have the hardware diagnostic tools perform the Address test.</td>
</tr>
<tr>
<td></td>
<td>Clear to have the diagnostic tools skip the Address test.</td>
</tr>
<tr>
<td>Bus Noise</td>
<td>Check to have the hardware diagnostic tools perform the Bus Noise test.</td>
</tr>
<tr>
<td></td>
<td>Clear to have the diagnostic tools skip the Bus Noise test.</td>
</tr>
<tr>
<td>Start:</td>
<td>Enter the starting address of the range that you want to test.</td>
</tr>
<tr>
<td>End:</td>
<td>Enter the ending address of the range that you want to test.</td>
</tr>
<tr>
<td>Byte</td>
<td>Select to have the hardware diagnostic tools perform byte-size operations.</td>
</tr>
</tbody>
</table>
Table 22.11 Memory Tests Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>Select to have the hardware diagnostic tools perform word-size operations.</td>
</tr>
<tr>
<td>Long Word</td>
<td>Select to have the hardware diagnostic tools perform long-word-size operations.</td>
</tr>
<tr>
<td>Passes</td>
<td>Enter the number of times that you want to repeat the specified tests.</td>
</tr>
<tr>
<td>Show Log</td>
<td>Click to display a text file that logs memory test actions.</td>
</tr>
<tr>
<td>Use Target CPU</td>
<td>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. Clear to have the hardware diagnostic tools execute the test code through the remote connection interface. 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.</td>
</tr>
<tr>
<td>Target Scratch Memory Start</td>
<td>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).</td>
</tr>
<tr>
<td>Target Scratch Memory End</td>
<td>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).</td>
</tr>
<tr>
<td>Status</td>
<td>Shows memory test progress information. Click the Details button to show more thorough progress information.</td>
</tr>
<tr>
<td>Begin Test</td>
<td>Click to have the hardware diagnostic tools perform the memory tests that you specified. The Status reflects memory test progress.</td>
</tr>
</tbody>
</table>

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.
- **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 $+\text{number}$ and $-\text{number}$ 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 shows the window that appears when you click the Edit Connection button. Table 22.12 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.
Figure 22.13 Editing a Logic Analyzer Connection

Table 22.12 Logic Analyzer Connection Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name that you want to assign to this collection of options.</td>
</tr>
<tr>
<td>Debugger</td>
<td>Use to specify the debugger to use with the logic analyzer.</td>
</tr>
<tr>
<td>Connection Type</td>
<td>Use to specify the connection method to the logic analyzer.</td>
</tr>
<tr>
<td>Analyzer Type</td>
<td>Use to specify the type of logic analyzer.</td>
</tr>
<tr>
<td>Host Name</td>
<td>Enter the Internet Protocol (IP) address of the logic analyzer.</td>
</tr>
<tr>
<td>Analyzer Configuration File</td>
<td>Enter the name of the configuration file that the logic analyzer requires.</td>
</tr>
<tr>
<td>Analyzer Slot</td>
<td>Enter the slot name that identifies the logic analyzer location.</td>
</tr>
</tbody>
</table>
Working with Hardware Tools

Working with a 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace Support File</td>
<td>Enter the name of the file that the logic analyzer requires to support the collection of trace data.</td>
</tr>
<tr>
<td>Analyzer Can Cause Target Breakpoint</td>
<td>Check to allow the logic analyzer to cause a hardware breakpoint. Clear to prevent the logic analyzer from causing a hardware breakpoint.</td>
</tr>
<tr>
<td>Target Breakpoint Can Cause Analyzer Trigger</td>
<td>Check to allow a hardware breakpoint to trigger the logic analyzer. Clear to prevent a hardware breakpoint from triggering the logic analyzer.</td>
</tr>
</tbody>
</table>
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.
Working with Hardware Tools

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.

**Figure 22.15 Cache Window**

Profile Window

Use the Profile window (Figure 22.16) 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.

**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.
Working with Hardware Tools

Command Window

Figure 22.16 Profile Window

<table>
<thead>
<tr>
<th>Function</th>
<th>Count</th>
<th>Time</th>
<th>%</th>
<th>% of Children</th>
<th>% of Average</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Stack Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>1</td>
<td>54</td>
<td>3.4</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td>argc</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>debug_break</td>
<td>1</td>
<td>0.7</td>
<td>0.1</td>
<td>0.7</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>debug_thread</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>debug_expression</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>debug_source</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>debug_thread_end</td>
<td>1</td>
<td>0.7</td>
<td>0.1</td>
<td>0.7</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>emnu</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>func</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>func2</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>func3</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>func4</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>func5</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>func6</td>
<td>1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 shows the Command window.

Figure 22.17 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.
Working with Hardware Tools

Command Window
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 consists of these sections:

- Overview
- Using the Profiler
- Configuring
- Viewing Results
- Troubleshooting
- Reference

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.

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

```c
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 would add this line of code to the source file that includes your `main()` function:

```c
#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 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.

<table>
<thead>
<tr>
<th>#pragma profile on</th>
<th>enables calls to the profiler in functions that are declared following the pragma</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma profile off</td>
<td>disables calls to the profiler in functions that are declared following the pragma</td>
</tr>
<tr>
<td>#pragma profile reset</td>
<td>sets the profile setting to the value selected in the preferences panel</td>
</tr>
</tbody>
</table>

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++.
Profiler
Configuring

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 23.1  ProfilerInit() Prototypes

| C/C++     | long ProfilerInit(        |
|           |                     ProfilerCollectionMethod method, |
|           |                     ProfilerTimeBase timeBase, |
|           |                     short numFunctions, short |
|           |                     stackDepth); |

Table 23.2  ProfilerInit() Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>collect detailed or summary data</td>
</tr>
<tr>
<td>timeBase</td>
<td>time scale to use in measurements</td>
</tr>
<tr>
<td>numFunctions</td>
<td>maximum number of routines to profile</td>
</tr>
<tr>
<td>stackDepth</td>
<td>approximate maximum depth of deepest calling tree</td>
</tr>
</tbody>
</table>

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
Profiler
Configuring

supported on all target platforms. Refer to the Targeting Manual for your product to
determine which time bases 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:

```c
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().
Profiler

Viewing Results

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.

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 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 to 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 explains the ways you may view profile data. You will look at:

- What It Does—the principle features of the profiler
Profiler
Viewing Results

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.

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 timebases (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

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
<td>Routine name. (The profiler unmangles C++ function names.)</td>
</tr>
<tr>
<td>Count</td>
<td>Number of times this routine was called.</td>
</tr>
<tr>
<td>Time</td>
<td>Time spent in this routine, not counting time in routines that this routine calls.</td>
</tr>
<tr>
<td>%</td>
<td>Percent of total time for the Time column.</td>
</tr>
<tr>
<td>+Children</td>
<td>Time spent in this routine and all the routines it calls.</td>
</tr>
<tr>
<td>%</td>
<td>Percent of total time for the +Children column.</td>
</tr>
<tr>
<td>Average</td>
<td>Average time for each routine invocation: Time divided by the number of times the routine was called.</td>
</tr>
<tr>
<td>Maximum</td>
<td>Longest time for an invocation of the routine.</td>
</tr>
<tr>
<td>Minimum</td>
<td>Shortest time for an invocation of the routine.</td>
</tr>
</tbody>
</table>

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.

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 Figure 23.4 shows.
Profiler
Viewing Results

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.
### 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.

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma profile on</td>
<td>Enables calls to the profiler in functions that are declared following the pragma.</td>
</tr>
<tr>
<td>#pragma profile off</td>
<td>Disables calls to the profiler in functions that are declared following the pragma.</td>
</tr>
<tr>
<td>#pragma profile reset</td>
<td>Sets the profile setting to the value selected in the preferences panel.</td>
</tr>
</tbody>
</table>

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:

```c
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().

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.

```c
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.

```c
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.

```c
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 consists of 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 plugin compiler architecture. This architecture provides these features:

- Modularity—the IDE associates a specific compiler plugin with a particular programming language or environment. For example, a compiler plugin exists for C++ source code, and another compiler plugin exists for Java source code.

- Flexibility—as new programming languages develop, the IDE can use new compiler plugins.

The IDE associates common filename extensions with various plugin 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.
**Compiling Projects**

### 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).
- 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.
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 consists of 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.
- 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.
Linkers
Linking Projects

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
- Register Details Window XML Specification
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 consists of these sections:

- Customizing IDE Commands
- Customize Toolbars
- Customize Key Bindings

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 shows the Customize IDE Commands window. Table 26.1 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 Buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Group</td>
<td>Click to add a new command group to the Commands list.</td>
</tr>
<tr>
<td>New Command</td>
<td>Click to add a new command setting within a group.</td>
</tr>
<tr>
<td>Factory Settings</td>
<td>Click to restore default options for the current Customize IDE Commands (Commands and Toolbar Items) lists.</td>
</tr>
<tr>
<td>Revert</td>
<td>Click to restore the most recently saved options for the current Customize IDE Commands (Commands and Toolbar Items) lists.</td>
</tr>
<tr>
<td>Export</td>
<td>Click to save a file that contains commands and key bindings to use later in the Customize IDE Commands lists.</td>
</tr>
</tbody>
</table>
Customizing the IDE

Customizing IDE Commands

Table 26.1 Customize IDE Commands Window Buttons (continued)

<table>
<thead>
<tr>
<th>Button</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import</td>
<td>Click to open a file that contains commands and key bindings to use in the current Customize IDE Commands lists.</td>
</tr>
<tr>
<td>Save</td>
<td>Click to save customizations to the Customize IDE Commands list.</td>
</tr>
</tbody>
</table>

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 (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, as Figure 26.2 shows. 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.
Customizing the IDE

Customizing IDE Commands

Defining Command Actions (Windows)

Figure 26.2 shows Command Action fields, which let you associate an action with the new menu command:

Figure 26.2 Command Action Fields

- **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 shows a list of pre-defined argument variables; Figure 26.4 shows a list of pre-defined directory variables.
Table 26.2 explains pre-defined variables for command-line arguments.

**Table 26.2 Pre-Defined Variables in Command Definitions**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Command-line output</th>
</tr>
</thead>
<tbody>
<tr>
<td>%sourceFilePath</td>
<td>sourceFilePath is the frontmost editor window's full path.</td>
</tr>
<tr>
<td>%sourceFileDir</td>
<td>sourceFileDir is the frontmost editor window's directory.</td>
</tr>
<tr>
<td>%sourceFileName</td>
<td>sourceFileName is the frontmost editor window's filename.</td>
</tr>
<tr>
<td>%sourceLineNumber</td>
<td>sourceLineNumber is the line number of the insertion point in the front window.</td>
</tr>
<tr>
<td>%sourceSelection</td>
<td>sourceSelection is the path to a temporary file containing the currently selected text.</td>
</tr>
<tr>
<td>%sourceSelUpdate</td>
<td>sourceSelUpdate is like sourceSelection, except the IDE waits for the command to finish and updates the selected text with the contents of the file.</td>
</tr>
</tbody>
</table>
### Table 26.2 Pre-Defined Variables in Command Definitions (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Command-line output</th>
</tr>
</thead>
<tbody>
<tr>
<td>%projectFilePath</td>
<td>projectFilePath is the full path of the front project window.</td>
</tr>
<tr>
<td>%projectFileDir</td>
<td>projectFileDir is the directory of the front project window.</td>
</tr>
<tr>
<td>%projectFileName</td>
<td>projectFileName is the filename of the front project window.</td>
</tr>
<tr>
<td>%projectSelectedFiles</td>
<td>projectSelectedFiles passes the selected filenames in the project window.</td>
</tr>
<tr>
<td>%targetFilePath</td>
<td>targetFilePath is the full path of the output file of the front project.</td>
</tr>
<tr>
<td>%targetFileDir</td>
<td>targetFileDir is the directory of the output file of the front project.</td>
</tr>
<tr>
<td>%targetFileName</td>
<td>targetFileName is the filename of the output file of the front project.</td>
</tr>
<tr>
<td>%currentTargetName</td>
<td>currentTargetName passes the name of the current target of the frontmost window.</td>
</tr>
<tr>
<td>%symFilePath</td>
<td>symFilePath is the full path to the symbolics file of the front project (can be the same as targetFile, such as CodeView).</td>
</tr>
<tr>
<td>%symFileDir</td>
<td>symFileDir is the full directory to the symbolics file of the front project (can be the same as targetFile, such as CodeView)</td>
</tr>
<tr>
<td>%symFileName</td>
<td>symFileName is the full filename to the symbolics file of the front project (can be the same as targetFile, such as CodeView)</td>
</tr>
</tbody>
</table>

### Using a Pre-defined Variable

To use a pre-defined variable, follow these steps:

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.
Customizing the IDE

Customize Toolbars

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 use the Toolbar Items page of the Customize IDE Commands window to put the commands you use often into your IDE toolbars. Figure 26.5 shows this page.

The IDE toolbars contain a series of elements, each of which 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.

Figure 26.5  Toolbar Items Page

This section explains these topics:
Kinds of Toolbars

The CodeWarrior IDE uses two toolbar types:

- **Main toolbar (Windows OS)**—This toolbar (Figure 26.6), also known as the floating toolbar, is always available.
- **Window toolbars**—These toolbars appear in particular windows, such as the Project window toolbar (Figure 26.7) 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)](image)

![Figure 26.7 Project Window Toolbar](image)

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
Customizing the IDE
Customize Toolbars

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

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 Table 26.3 explains.

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the button.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the button.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Control-click the button.</td>
</tr>
<tr>
<td>Linux</td>
<td>Ctrl-click the button.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this to clear the main toolbar...</th>
<th>Do this to clear the window toolbar...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Toolbars &gt; Clear Main Toolbar.</td>
<td>Select View &gt; Toolbars &gt; Clear Window Toolbar.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Toolbar &gt; Clear Floating Toolbar.</td>
<td>Select Window &gt; Toolbar &gt; Clear Window Toolbar.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; Toolbar &gt; Clear Floating Toolbar.</td>
<td>Select Window &gt; Toolbar &gt; Clear Window Toolbar.</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; Toolbar &gt; Clear Floating Toolbar.</td>
<td>Select Window &gt; Toolbar &gt; Clear Window Toolbar.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this to reset the main toolbar...</th>
<th>Do this to reset the window toolbar...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Select View &gt; Toolbars &gt; Reset Main Toolbar.</td>
<td>Select View &gt; Toolbars &gt; Reset Window Toolbar.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Select Window &gt; Toolbar &gt; Reset Floating Toolbar.</td>
<td>Select Window &gt; Toolbar &gt; Reset Window Toolbar.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Select Window &gt; Toolbar &gt; Reset Floating Toolbar.</td>
<td>Select Window &gt; Toolbar &gt; Reset Window Toolbar.</td>
</tr>
<tr>
<td>Linux</td>
<td>Select Window &gt; Toolbar &gt; Reset Floating Toolbar.</td>
<td>Select Window &gt; Toolbar &gt; Reset Window Toolbar.</td>
</tr>
</tbody>
</table>

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.
Customize Key Bindings

As Figure 26.8 shows, 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
- Importing commands and key bindings
- Quote key prefix

### Table 26.6 Resetting a Toolbar by Using a Contextual Menu

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this to reset the main toolbar...</th>
<th>Do this to reset the window toolbar...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Right-click the toolbar and select <strong>Reset Toolbar</strong>.</td>
<td>Right-click the toolbar and select <strong>Reset Toolbar</strong>.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Control-click the toolbar and select <strong>Reset Toolbar</strong>.</td>
<td>Control-click the toolbar and select <strong>Reset Toolbar</strong>.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Click and hold on the toolbar, then select <strong>Reset Toolbar</strong>.</td>
<td>Click and hold on the toolbar, then select <strong>Reset Toolbar</strong>.</td>
</tr>
<tr>
<td>Linux</td>
<td>Click and hold on the toolbar, then select <strong>Reset Toolbar</strong>.</td>
<td>Click and hold on the toolbar, then select <strong>Reset Toolbar</strong>.</td>
</tr>
</tbody>
</table>
Customizing the IDE
Customize Key Bindings

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. The Edit Key Binding dialog box (Figure 26.9) appears.
Customizing the IDE

Customize Key Bindings

Figure 26.9 Edit Key Binding Dialog Box

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.

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.
Customizing the IDE

Customize 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. 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.
Customizing the IDE

Customize Key Bindings
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 consists of these sections:

- IDE Preferences Window
- General Panels
- Editor Panels
- Debugger Panels

IDE Preferences Window

The IDE Preferences window (Figure 27.1) lists global IDE options. Table 27.1 explains the items of this window. 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 Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE Preference Panels list</td>
<td>Lists preference panels, organized by group. Click the hierarchical control next to a group name to show or hide individual preference panels.</td>
</tr>
<tr>
<td>Preference panel</td>
<td>Shows options for the selected item in the IDE Preference Panels list.</td>
</tr>
<tr>
<td>Factory Settings</td>
<td>Click to restore the default options for the current preference panel.</td>
</tr>
<tr>
<td>Revert Panel</td>
<td>Click to restore the most recently saved options for the current preference panel.</td>
</tr>
<tr>
<td>Export Panel</td>
<td>Click to save an XML file that contains options for the current preference panel.</td>
</tr>
<tr>
<td>Import Panel</td>
<td>Click to open an XML file that contains options for the current preference panel.</td>
</tr>
</tbody>
</table>
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
- Concurrent Compiles
- IDE Extras
- Help Preferences
- Plugin Settings
- Shielded Folders
- Source Trees

Build Settings

The Build Settings preference panel (Figure 27.2) 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

Table 27.2 explains the items of this panel.
Working with IDE Preferences

General Panels

Figure 27.2  Build Settings Preference Panel

<table>
<thead>
<tr>
<th>Settings</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build before running</td>
<td>Select to have the IDE play an alert sound after completing a Bring Up To Date or Make command.</td>
</tr>
<tr>
<td>Success (Macintosh, Solaris, and Linux)</td>
<td>Choose the sound the IDE plays after successfully completing a Bring Up To Date or Make command.</td>
</tr>
<tr>
<td>Failure (Macintosh, Solaris, and Linux)</td>
<td>Choose the sound the IDE plays after failing to complete a Bring Up To Date or Make command.</td>
</tr>
<tr>
<td>Compiler thread stack (K)</td>
<td>Enter the kilobytes of memory to allocate to the file cache used for #include files during a project build.</td>
</tr>
</tbody>
</table>

Table 27.2 Build Settings Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play sound after ‘Bring Up To Date’ &amp; ‘Make’ (Macintosh, Solaris, and Linux)</td>
<td>Select to have the IDE play an alert sound after completing a Bring Up To Date or Make command.</td>
</tr>
<tr>
<td>Success (Macintosh, Solaris, and Linux)</td>
<td>Choose the sound the IDE plays after successfully completing a Bring Up To Date or Make command.</td>
</tr>
<tr>
<td>Failure (Macintosh, Solaris, and Linux)</td>
<td>Choose the sound the IDE plays after failing to complete a Bring Up To Date or Make command.</td>
</tr>
<tr>
<td>Build before running</td>
<td>Choose to always build the project before running it, never build the project before running it, or ask for the desired action.</td>
</tr>
<tr>
<td>Save open files before build</td>
<td>Select to automatically save the contents of all editor windows before starting a build.</td>
</tr>
<tr>
<td>Show message after building up-to-date project</td>
<td>Select to have the IDE display a message after successfully building a project.</td>
</tr>
</tbody>
</table>
Concurrent Compiles

The Concurrent Compiles preference panel (Figure 27.3) controls execution of simultaneous IDE compilation processes. The IDE lists this panel in the IDE Preference Panels list when the active compiler supports concurrency. Table 27.3 explains the items of this panel.

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.

Table 27.3 Concurrent Compiles Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler thread stack</td>
<td>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.</td>
</tr>
<tr>
<td>(Windows and Macintosh)</td>
<td></td>
</tr>
<tr>
<td>Use Local Project Data Storage</td>
<td>Select to specify a location to save project data if the project is on a read-only volume. Click <strong>Choose</strong> to select the location.</td>
</tr>
</tbody>
</table>
Working with IDE Preferences

General Panels

Table 27.3 Concurrent Compiles Preference Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recommended</td>
<td>Select to allow the number of concurrent compiles suggested by the IDE.</td>
</tr>
<tr>
<td>User Specified</td>
<td>Select to stipulate the number of concurrent compiles.</td>
</tr>
</tbody>
</table>

IDE Extras

The IDE Extras preference panel (Figure 27.4) 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

Table 27.4 explains the items of this panel.

Figure 27.4 IDE Extras Preference Panel
Table 27.4 IDE Extras Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Menu bar layout</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Projects</strong></td>
<td>Enter the number of recently opened projects for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.</td>
</tr>
<tr>
<td><strong>Documents</strong></td>
<td>Enter the number of recently opened documents for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.</td>
</tr>
<tr>
<td><strong>Symbolics</strong></td>
<td>Enter the number of recently opened symbolics files for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.</td>
</tr>
<tr>
<td><strong>Workspaces</strong></td>
<td>Enter the number of recently opened workspaces for the IDE to display in the Open Recent submenu. Enter zero to disable this feature.</td>
</tr>
<tr>
<td><strong>Context popup delay</strong> (Macintosh, Solaris, and Linux)</td>
<td>Enter the number of ticks to wait before displaying contextual menus. A tick is 1/60 of a second.</td>
</tr>
<tr>
<td><strong>Use Third Party Editor</strong> (Windows)</td>
<td>Select to use a third-party text editor to edit source files.</td>
</tr>
<tr>
<td><strong>Launch Editor</strong> (Windows)</td>
<td>Enter a command-line expression that runs the desired third-party text editor.</td>
</tr>
<tr>
<td><strong>Launch Editor w/ Line #</strong> (Windows)</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Use Multiple Document Interface</strong> (Windows)</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Zoom windows to full screen</strong> (Macintosh, Solaris, and Linux)</td>
<td>Select to have zoomed windows fill the entire screen. Clear to have zoomed windows in a default size.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use Script menu</strong> (Macintosh, Solaris, and Linux)</td>
<td>Select to display the Scripts menu in the menu bar. Clear to remove the Scripts menu from the menu bar.</td>
</tr>
<tr>
<td><strong>Use External Editor</strong> (Macintosh, Solaris, and Linux)</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Use ToolServer menu</strong> (Classic Macintosh)</td>
<td>Select to display the ToolServer menu in the menu bar. Clear to remove the ToolServer menu from the menu bar.</td>
</tr>
<tr>
<td><strong>Enable automatic Toolbar help</strong> (Classic Macintosh)</td>
<td>Select to display Balloon Help after resting the cursor over a toolbar icon. Clear to prevent Balloon Help from appearing.</td>
</tr>
<tr>
<td><strong>Use default workspace</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Find Reference using</strong> (Macintosh)</td>
<td>Choose an online browser application to view reference information and definitions.</td>
</tr>
<tr>
<td><strong>Show Code and Data Sizes</strong> (Windows)</td>
<td>Displays or hides Code and Data columns in project manager.</td>
</tr>
</tbody>
</table>
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 (Figure 27.5), available on the Solaris and Linux IDE hosts, specifies the browser used for viewing IDE online help. Table 27.5 explains the items of this panel.

**Figure 27.5 Help Preferences Panel**

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser Path</td>
<td>Enter a path to the browser to use for viewing IDE online help. Alternatively, use the Choose... button.</td>
</tr>
<tr>
<td>Choose...</td>
<td>Click to select the path to the browser to use for viewing IDE online help.</td>
</tr>
</tbody>
</table>

**Plugin Settings**

The Plugin Settings preference panel (Figure 27.6) contains options for troubleshooting third-party IDE plugins. Table 27.6 explains the items of this panel.
Shielded Folders

The Shielded Folder preference panel (Figure 27.7) 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. Table 27.7 explains the items of this panel; Table 27.8 explains the default regular expressions in this 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.
Working with IDE Preferences

General Panels

Figure 27.7 Shielded Folders Preference Panel

Table 27.7 Shielded Folders Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shielded folder list</td>
<td></td>
<td>Lists folders that match the specified regular expression. The IDE skips these folders during project operations, find-and-compare operations, or both.</td>
</tr>
<tr>
<td>Regular Expression</td>
<td></td>
<td>Enter the regular expression used to shield folders from selected operations.</td>
</tr>
<tr>
<td>Project operations</td>
<td></td>
<td>Select to have the IDE skip folders during project operations. A bullet appears in the corresponding column of the shielded folder list.</td>
</tr>
<tr>
<td>Find and compare operations</td>
<td></td>
<td>Select to have the IDE skip folders during find-and-compare operations. A bullet appears in the corresponding column of the shielded folder list.</td>
</tr>
<tr>
<td>Add</td>
<td></td>
<td>Click to add the current Regular Expression field entry to the shielded folder list.</td>
</tr>
</tbody>
</table>
Source Trees

Use the Source Trees panel (Figure 27.8) 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. Table 27.9 explains the items of this panel.
Figure 27.8 Source Trees Panel

![Source Trees Panel Image]

Table 27.9 Source Trees Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Tree list</td>
<td>Contains the <strong>Name</strong> and <strong>Path</strong> of currently defined source trees.</td>
</tr>
<tr>
<td>Name</td>
<td>Enter a name for a new source tree or modify the name of a selected source tree.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Choose the source-tree path type.</td>
</tr>
<tr>
<td>Choose</td>
<td>Click to select or modify a source-tree path.</td>
</tr>
<tr>
<td>Add</td>
<td>Click to add a new source-tree path to the Source Tree list.</td>
</tr>
<tr>
<td>Change</td>
<td>Click to modify the selected source-tree name or path.</td>
</tr>
<tr>
<td>Remove</td>
<td>Click to delete the selected source tree from the Source Tree list.</td>
</tr>
</tbody>
</table>
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.
Working with IDE Preferences

Editor Panels

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
- Code Formatting
- Editor Settings
- Font & Tabs
- Text Colors

Code Completion

The Code Completion preference panel (Figure 27.9) provides options for customizing the IDE code-completion behavior, including:

- automatic invocation and indexing
Working with IDE Preferences
Editor Panels

- window positioning and appearance delay
- case sensitivity

Table 27.10 explains the items of this panel.

Figure 27.9 Code Completion Preference Panel

Table 27.10 Code Completion Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Invocation</td>
<td>Select to automatically open the Code Completion window to complete programming-language symbols. Clear to manually open the window.</td>
</tr>
<tr>
<td>Window follows insertion point</td>
<td>Select to have the Code Completion window follow the insertion point as you edit text. Clear to leave the window in place.</td>
</tr>
<tr>
<td>Display deprecated items</td>
<td>Select to have the Code Completion window display obsolete items in gray text. Clear to have the window hide obsolete items.</td>
</tr>
<tr>
<td>Case sensitive</td>
<td>Select to have the IDE consider case when completing code. Clear to have the IDE ignore case.</td>
</tr>
<tr>
<td>Code Completion Delay (ticks)</td>
<td>Enter the number of ticks to wait before opening the Code Completion window. A tick is 1/60 of a second.</td>
</tr>
</tbody>
</table>
Code Formatting

The Code Formatting preference panel (Figure 27.10) provides options for customizing editor code-formatting behavior, including:

- indenting
- syntax placement
- brace handling

Table 27.11 explains the items of this panel.

Figure 27.10 Code Formatting Preference Panel

Table 27.11 Code Formatting Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Automatic Code Formatting</td>
<td>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.</td>
</tr>
<tr>
<td>Language Settings</td>
<td>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.</td>
</tr>
</tbody>
</table>

Sample

```
if (a > b)
{
    c = c;
}
else if (a < b)
{
    a = f;
}
switch (g)
{
    case 1:
```
# Table 27.11 Code Formatting Preference Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format braces</td>
<td>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.</td>
</tr>
<tr>
<td>Place opening brace on separate line</td>
<td>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.</td>
</tr>
<tr>
<td>Indent braces</td>
<td>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.</td>
</tr>
<tr>
<td>Place &quot;else&quot; on same line as closing brace</td>
<td>Check to have the editor place else and else if text on the same line as the closing brace of the if or else if statement. Clear to prevent the editor from placing else and else if text on the same line as the closing brace of the if or else if statement.</td>
</tr>
<tr>
<td>Indent code within braces</td>
<td>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.</td>
</tr>
</tbody>
</table>
Editor Settings

The Editor Settings preference panel (Figure 27.11) provides options for customizing the editor, including:

- fonts, window locations, and insertion-point positions
- contextual menus
- additional editor-window features

Table 27.12 explains the items of this panel.
Working with IDE Preferences

Editor Panels

Figure 27.11 Editor Settings Preference Panel

![Editor Settings Preference Panel](image)

Table 27.12 Editor Settings Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Font preferences</strong></td>
<td>Select to retain font settings for each source file. Clear to apply default font settings each time the IDE displays the source file.</td>
</tr>
<tr>
<td><strong>Selection position</strong></td>
<td>Select to retain the text-insertion position in each source file.</td>
</tr>
<tr>
<td><strong>Window position and size</strong></td>
<td>Select to retain the location and dimensions of each editor window.</td>
</tr>
<tr>
<td><strong>Edit Commands</strong></td>
<td>Select to add <strong>Edit</strong> menu commands to contextual menus.</td>
</tr>
<tr>
<td><strong>Browser Commands</strong></td>
<td>Select to add <strong>Browser</strong> menu commands to contextual menus. Also select in order to use the <strong>Insert Template Commands</strong> option.</td>
</tr>
<tr>
<td><strong>Insert Template Commands</strong></td>
<td>Select to add the <strong>Insert Template</strong> submenu to contextual menus. The submenu displays source-defined function templates.</td>
</tr>
</tbody>
</table>
### Table 27.12 Editor Settings Preference Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Commands</strong></td>
<td>Select to add Project menu commands to contextual menus.</td>
</tr>
<tr>
<td><strong>VCS Commands</strong></td>
<td>Select to add VCS (Version Control System) menu commands to contextual menus.</td>
</tr>
<tr>
<td><strong>Debugger Commands</strong></td>
<td>Select to add Debug menu commands to contextual menus.</td>
</tr>
<tr>
<td><strong>Balance while typing</strong></td>
<td>Select to flash the matching (, [, or ( after typing ), ], or ) in an editor window.</td>
</tr>
<tr>
<td><strong>Use multiple undo</strong></td>
<td>Select to allow multiple undo and redo operations while editing text.</td>
</tr>
<tr>
<td><strong>Relaxed C popup parsing</strong></td>
<td>Select to allow the C parser to recognize some non-standard function formats and avoid skipping or misinterpreting some definition styles.</td>
</tr>
<tr>
<td><strong>Drag and drop editing</strong></td>
<td>Select to allow drag-and-drop text editing.</td>
</tr>
<tr>
<td><strong>Left margin click selects line</strong></td>
<td>Select to allow selection of an entire line of text by clicking in the left margin of the editor window.</td>
</tr>
<tr>
<td><strong>Sort function popup</strong></td>
<td>Select to sort function names by alphabetical order in menus. Clear to sort function names by order of appearance in the source file.</td>
</tr>
<tr>
<td><strong>Enable Virtual Space</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Balance Flash Delay</strong></td>
<td>Enter the number of ticks to flash a balancing punctuation character. A tick is 1/60 of a second.</td>
</tr>
<tr>
<td><strong>Default file format</strong></td>
<td>Choose the default end-of-line format used to save files.</td>
</tr>
</tbody>
</table>

### Font & Tabs

The Font & Tabs preference panel (Figure 27.12) provides options for customizing settings used by the editor, including:

- font and font size used in editor windows
- auto indentation and tab size
Working with IDE Preferences

Editor Panels

- tabs on selections and replacing tabs with spaces

Table 27.13 explains the items of this panel.

Figure 27.12  Font & Tabs Preference Panel

Table 27.13  Font & Tabs Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font</td>
<td>Choose the typeface displayed in editor windows.</td>
</tr>
<tr>
<td>Size</td>
<td>Choose the font size displayed in editor windows.</td>
</tr>
<tr>
<td>Script (Windows)</td>
<td>Choose the IDE script system. The script system maps keyboard keys to characters of an alphabet.</td>
</tr>
<tr>
<td>Tab indents selection</td>
<td>Select to indent each line of selected text after pressing Tab. Clear to replace selected text with a tab character after pressing Tab.</td>
</tr>
<tr>
<td>Tab Size</td>
<td>Enter the number of spaces to substitute in place of a tab character. This number applies to the Tab Inserts Spaces option.</td>
</tr>
</tbody>
</table>
Working with IDE Preferences
Editor Panels

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 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.
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.

Table 27.13  Font & Tabs Preference Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Indent</td>
<td>Select to automatically apply the indentation level from the previous line of text to each new line created by pressing Enter or Return.</td>
</tr>
<tr>
<td>Tab Inserts Spaces</td>
<td>Select to insert spaces instead of a tab character after pressing Tab. The Tab Size option determines the number of inserted spaces.</td>
</tr>
</tbody>
</table>
Text Colors

The Text Colors preference panel (Figure 27.13) 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

Table 27.14 explains the items of this panel.

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 Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Foreground</strong></td>
<td>Click the color swatch to display a dialog box. Use the dialog box to set the foreground color used in editor windows for text.</td>
</tr>
<tr>
<td><strong>Background</strong></td>
<td>Click the color swatch to set the background color used in editor windows.</td>
</tr>
<tr>
<td><strong>Activate Syntax Coloring</strong></td>
<td>Select to apply custom colors to comments, keywords, strings, and custom keywords in text. Clear to use the <strong>Foreground</strong> color for all text.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>Click the color swatch to set the color used for source-code comments.</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>Click the color swatch to set the color used for source-code language keywords.</td>
</tr>
<tr>
<td><strong>Strings</strong></td>
<td>Click the color swatch to set the color used for source-code string literals.</td>
</tr>
<tr>
<td><strong>Set 1, Set 2, Set 3, Set 4</strong></td>
<td>Click a color swatch to set the color used for the corresponding custom-keyword set.</td>
</tr>
<tr>
<td><strong>Edit</strong></td>
<td>Click to add, modify, or remove keywords from the corresponding custom-keyword set.</td>
</tr>
<tr>
<td><strong>Activate Browser Coloring</strong></td>
<td>Select to apply custom colors to browser symbols in text. Clear to use the <strong>Foreground</strong> color for all text.</td>
</tr>
<tr>
<td><strong>Classes</strong></td>
<td>Click the color swatch to set the color used for source-code classes.</td>
</tr>
<tr>
<td><strong>Constants</strong></td>
<td>Click the color swatch to set the color used for source-code constants.</td>
</tr>
<tr>
<td>** Enums**</td>
<td>Click the color swatch to set the color used for source-code enumerations.</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>Click the color swatch to set the color used for source-code functions.</td>
</tr>
<tr>
<td><strong>Globals</strong></td>
<td>Click the color swatch to set the color used for source-code global variables.</td>
</tr>
</tbody>
</table>
Working with IDE Preferences

Editor Panels

Table 27.14  Text Colors Preference Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macros</td>
<td>Click the color swatch to set the color used for source-code macros.</td>
</tr>
<tr>
<td>Templates</td>
<td>Click the color swatch to set the color used for source-code templates.</td>
</tr>
<tr>
<td>TypeDefs</td>
<td>Click the color swatch to set the color used for source-code type definitions.</td>
</tr>
<tr>
<td>Other</td>
<td>Click the color swatch to set the color used for other symbols not specified in the <strong>Activate Browser Coloring</strong> section.</td>
</tr>
</tbody>
</table>

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**.
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**
- **Window Settings**
- **Global Settings**
- **Remote Connections**

---

**Display Settings**

The **Display Settings** preference panel (Figure 27.14) provides options for customizing various aspects of the IDE Debugger, including:

- assignment of colors to changed variables and watchpoints
- viewing variable types
- displaying local variables
- using decimal values
- sorting functions
• using dynamic objects

Table 27.15 explains the items in this panel.

Figure 27.14 Display Settings Preference Panel

Table 27.15 Display Settings Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable values change</td>
<td>Click the color swatch to set the color that indicates a changed variable value.</td>
</tr>
<tr>
<td>Watchpoint indicator</td>
<td>Click the color swatch to set the color that indicates a changed watchpoint value.</td>
</tr>
<tr>
<td>Show variable types</td>
<td>Select to always show the type associated with each variable.</td>
</tr>
<tr>
<td>Show variable location</td>
<td>Select to display the Location column in the Variables pane of the Thread window.</td>
</tr>
<tr>
<td>Show all locals</td>
<td>Select to show all local variables. Clear to have the debugger show only variables near the program counter.</td>
</tr>
</tbody>
</table>
## Working with IDE Preferences

### Debugger Panels

The **Window Settings** preference panel (Figure 27.15) provides options for customizing how the debugger displays windows during debugging sessions, including non-debugging and project windows. **Table 27.16** explains the items of this panel.

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Smart Variable Formatting</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Show hidden locals</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Show values as decimal</strong></td>
<td>Select to always show decimal values instead of hexadecimal values.</td>
</tr>
<tr>
<td><strong>Sort functions by method name, in symbolics window</strong></td>
<td>Select to sort functions of the form className::methodName in the Symbolics window by methodName. Clear to sort by className.</td>
</tr>
<tr>
<td><strong>Attempt to show the dynamic runtime type of objects</strong></td>
<td>Select to attempt to display the runtime type of the specified language objects. Clear to display the static type.</td>
</tr>
<tr>
<td><strong>Show variable values in source code</strong></td>
<td>Select to show variable values in contextual menus in the source code.</td>
</tr>
<tr>
<td><strong>Default size for unbounded arrays</strong></td>
<td>Enter the default number of unbounded array elements to display in a View Array window.</td>
</tr>
<tr>
<td><strong>Scientific Notation for (1-9) or more 0s after “0.”</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

### Window Settings

**Table 27.15** Display Settings Preference Panel Items (continued)
Working with IDE Preferences

Debugger Panels

Figure 27.15  Window Settings Preference Panel

<table>
<thead>
<tr>
<th>Window Settings</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>When Debugging Starts</td>
<td></td>
</tr>
<tr>
<td>Do nothing</td>
<td>Select to leave all windows in place when starting a debugging session.</td>
</tr>
<tr>
<td>Hide non-debugging windows</td>
<td>Select to hide, but not close, all non-debugging windows when starting a debugging session.</td>
</tr>
<tr>
<td>Do nothing to project windows</td>
<td>Select to prevent the IDE from hiding project windows when starting a debugging session.</td>
</tr>
<tr>
<td>Minimize non-debugging windows</td>
<td>Select to minimize all non-debugging windows when starting a debugging session.</td>
</tr>
<tr>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td>Collapse non-debugging windows</td>
<td>Select to collapse all non-debugging windows when starting a debugging session.</td>
</tr>
<tr>
<td>(Macintosh, Solaris, and Linux)</td>
<td></td>
</tr>
<tr>
<td>Hide non-debugging windows</td>
<td>Select to hide, but not close, all non-debugging windows when starting a debugging session.</td>
</tr>
<tr>
<td>Close non-debugging windows</td>
<td>Select to close all non-debugging windows, except for the active project window, when starting a debugging session.</td>
</tr>
<tr>
<td>Use Debugging Monitor</td>
<td>Select to use a second monitor during debugging sessions.</td>
</tr>
<tr>
<td>(Classic Macintosh)</td>
<td></td>
</tr>
</tbody>
</table>

Table 27.16  Window Settings Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do nothing</td>
<td>Select to leave all windows in place when starting a debugging session.</td>
</tr>
<tr>
<td>Minimize non-debugging windows</td>
<td>Select to minimize all non-debugging windows when starting a debugging session.</td>
</tr>
<tr>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td>Collapse non-debugging windows</td>
<td>Select to collapse all non-debugging windows when starting a debugging session.</td>
</tr>
<tr>
<td>(Macintosh, Solaris, and Linux)</td>
<td></td>
</tr>
<tr>
<td>Hide non-debugging windows</td>
<td>Select to hide, but not close, all non-debugging windows when starting a debugging session.</td>
</tr>
<tr>
<td>Close non-debugging windows</td>
<td>Select to close all non-debugging windows, except for the active project window, when starting a debugging session.</td>
</tr>
<tr>
<td>Do nothing to project windows</td>
<td>Select to prevent the IDE from hiding project windows when starting a debugging session.</td>
</tr>
<tr>
<td>Use Debugging Monitor</td>
<td>Select to use a second monitor during debugging sessions.</td>
</tr>
<tr>
<td>(Classic Macintosh)</td>
<td></td>
</tr>
<tr>
<td>Monitor for debugging</td>
<td>Choose the monitor to display debugging windows. The coordinates in parentheses identify the selected monitor in the QuickDraw® coordinate space.</td>
</tr>
<tr>
<td>(Classic Macintosh)</td>
<td></td>
</tr>
</tbody>
</table>
The Global Settings preference panel (Figure 27.16) 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

Table 27.17 explains the items of this panel.
Working with IDE Preferences
Debugger Panels

Figure 27.16 Global Settings Preference Panel

<table>
<thead>
<tr>
<th>Global Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Edited Files Between Debug Sessions</td>
</tr>
<tr>
<td>Maintain files in cache for [ ] days</td>
</tr>
</tbody>
</table>

Other Settings
- Confirm invalid file modification dates when debugging
- Automatically launch applications when SYM file opened
- Confirm "Kill Process" when closing or quitting
- Select stack crawl window when task is stopped
- Don't step into runtime support code
- Auto Target Libraries

Table 27.17 Global Settings Preference Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Edited Files Between Debug Sessions</td>
<td>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.</td>
</tr>
<tr>
<td>Maintain files in cache</td>
<td>Enter the number of days that the IDE maintains its file cache.</td>
</tr>
<tr>
<td>Purge Cache</td>
<td>Click to delete the file cache maintained by the IDE, freeing memory and disk space.</td>
</tr>
<tr>
<td>Confirm invalid file modification dates when debugging</td>
<td>Select to have the IDE display a warning message when debugging a project with mismatched file modification dates.</td>
</tr>
<tr>
<td>Automatically launch applications when SYM file opens</td>
<td>Select to automatically launch the application program associated with an open symbols file.</td>
</tr>
<tr>
<td>Confirm &quot;Kill Process&quot; when closing or quitting</td>
<td>Select to prompt for confirmation before killing processes upon quitting a debugging session.</td>
</tr>
<tr>
<td>Select stack crawl window when task is stopped</td>
<td>Select to bring forward the Stack Crawl window (also known as the Thread window) after the debugger stops tasks.</td>
</tr>
</tbody>
</table>
Working with IDE Preferences
Debugger Panels

Remote Connections

The **Remote Connections** preference panel (Figure 27.17) configures general network settings for remote-debugging connections between the host computer and other computers. **Table 27.18** explains the items of this panel.

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.

**Table 27.17**  Global Settings Preference Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Don’t step into runtime support code</strong></td>
<td>Select to have the IDE not step into Main Standard Library (MSL) runtime support code and instead directly step into your own code.</td>
</tr>
<tr>
<td><strong>Auto Target Libraries</strong></td>
<td>Select to have the IDE attempt to debug dynamically linked libraries (DLLs) loaded by the target application.</td>
</tr>
</tbody>
</table>

**Figure 27.17**  Remote Connections Preference Panel

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abatron Serial</td>
<td>Serial</td>
</tr>
<tr>
<td>Abatron TCP/IP</td>
<td>TCP/IP</td>
</tr>
<tr>
<td>Agilent EDM</td>
<td>TCP/IP</td>
</tr>
<tr>
<td>Agilent JTAG</td>
<td>TCP/IP</td>
</tr>
<tr>
<td>CCS EPPC</td>
<td>CCS Remote Connection</td>
</tr>
<tr>
<td>MSI EDM Raven</td>
<td>Parallel</td>
</tr>
<tr>
<td>MSI COP Raven</td>
<td>Parallel</td>
</tr>
<tr>
<td>MSI Wiggler</td>
<td>Parallel</td>
</tr>
<tr>
<td>MetroTRK</td>
<td>Serial</td>
</tr>
<tr>
<td>P&amp;E EDM</td>
<td>Parallel</td>
</tr>
<tr>
<td>PowerTAP JTAG</td>
<td>TCP/IP</td>
</tr>
<tr>
<td>PowerTAP PRO DPI</td>
<td>TCP/IP</td>
</tr>
<tr>
<td>WireTAP 6xx</td>
<td>Parallel</td>
</tr>
<tr>
<td>WireTAP JTAG</td>
<td>Parallel</td>
</tr>
</tbody>
</table>
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 symbols 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 symbols 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.
Working with IDE Preferences
Debugger Panels

4. Click **Remove**.
   The IDE updates the **Remote Connections** list.
5. Click **OK**, **Apply**, or **Save**.
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 consists of these sections:

- Target Settings Window
- Target Panels
- Code Generation Panels
- Editor Panels
- Debugger Panels

Target Settings Window

The Target Settings window (Figure 28.1) lists settings for the current project’s build targets. These target settings supersede global preferences defined in the IDE Preferences window. Table 28.1 explains the items of this 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
- **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
Working with Target Settings

**Target Settings Window**

- Command-Line Extras (Linux/Solaris)—configure environmental variables for user applications and define custom tool commands (if necessary)

**Figure 28.1 Target Settings Window**

![Target Settings Window](image)

**Table 28.1 Target Settings Window Items**

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Settings Panels list</td>
<td>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.</td>
</tr>
<tr>
<td>Settings panel</td>
<td>Shows options for the selected item in the <strong>Target Settings Panels</strong> list.</td>
</tr>
<tr>
<td>Factory Settings</td>
<td>Click to restore the default options for the current settings panel.</td>
</tr>
<tr>
<td>Revert Panel</td>
<td>Click to restore the most recently saved options for the current settings panel.</td>
</tr>
</tbody>
</table>
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.

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
Working with Target Settings
Target Panels

Target Settings
The Target Settings panel (Figure 28.2) 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

Table 28.2 explains the items of this panel.

Figure 28.2 Target Settings Panel

Table 28.2 Target Settings Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Name</td>
<td>Enter a name (26 or fewer characters) for the selected build target as it will appear in the project window.</td>
</tr>
<tr>
<td>Linker</td>
<td>Select the linker to use on the current build target.</td>
</tr>
<tr>
<td>Pre-linker</td>
<td>Select the pre-linker to use on the current build target.</td>
</tr>
<tr>
<td>Post-linker</td>
<td>Select the post-linker to use on the current build target.</td>
</tr>
</tbody>
</table>

Table 28.2 Target Settings Panel Items
Table 28.2 Target Settings Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output Directory</strong></td>
<td>Shows the location where the IDE creates the output binary file. Click <strong>Choose</strong> to change this location.</td>
</tr>
<tr>
<td><strong>Choose</strong></td>
<td>Click to select the directory in which the IDE saves the output binary file.</td>
</tr>
<tr>
<td><strong>Clear</strong></td>
<td>Click to delete the current <strong>Output Directory</strong> path.</td>
</tr>
<tr>
<td><strong>Save project entries using relative paths</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Access Paths**

The **Access Paths** settings panel (Figure 28.3) defines the search paths for locating and accessing a build target’s system files and header files. Table 28.3 explains the items of this panel.

**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 Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Always Search User Paths</strong></td>
<td>Select to treat <code>#include &lt;...&gt;</code> statements the same as <code>#include &quot;...&quot;</code> statements.</td>
</tr>
<tr>
<td><strong>Source relative includes</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>User Paths</strong></td>
<td>The <strong>User Paths</strong> list shows currently defined user-level access paths searched by <code>#include &quot;...&quot;</code> statements.</td>
</tr>
<tr>
<td><strong>System Paths</strong></td>
<td>The <strong>System Paths</strong> list shows currently defined system-level access paths searched by <code>#include &lt;...&gt;</code> statements.</td>
</tr>
</tbody>
</table>
Working with Target Settings
Target Panels

Table 28.3 Access Paths Settings Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interpret DOS and Unix Paths</strong>&lt;br&gt;(Macintosh)</td>
<td>Select to treat <code>/</code> and <code>\</code> as subfolder separator characters. Clear to treat <code>/</code> and <code>\</code> as ordinary text.</td>
</tr>
<tr>
<td><strong>Require Framework Style Includes</strong>&lt;br&gt;(Mac OS X)</td>
<td>Select to require <code>#include</code> statements of the form <code>LibraryName/HeaderFile.h</code>. Clear to allow statements of the form <code>HeaderFile.h</code>.</td>
</tr>
<tr>
<td><strong>Add Default</strong></td>
<td>Click to restore the default user- and system-level access paths.</td>
</tr>
<tr>
<td><strong>Host Flags</strong> list pop-up</td>
<td>Choose the host platforms that can use the selected access path.</td>
</tr>
<tr>
<td><strong>Add</strong></td>
<td>Click to add a user- or system-level access path.</td>
</tr>
<tr>
<td><strong>Change</strong></td>
<td>Click to modify the selected user- or system-level access path.</td>
</tr>
<tr>
<td><strong>Remove</strong></td>
<td>Click to remove the selected user- or system-level access path.</td>
</tr>
</tbody>
</table>

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, System Paths List Columns

<table>
<thead>
<tr>
<th>Name</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search status</td>
<td>✓</td>
<td>A checkmark icon indicates an active access path that the IDE searches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No checkmark icon indicates an inactive access path that the IDE does not search.</td>
</tr>
<tr>
<td>Recursive search</td>
<td>🔍</td>
<td>A folder icon indicates that the IDE recursively searches subdirectories of the access path.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No folder icon indicates that the IDE does not recursively search the access path.</td>
</tr>
</tbody>
</table>
**Working with Target Settings**

**Target Panels**

Table 28.4 User Paths, System Paths List Columns (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework (Mac OS X development)</td>
<td>![ ]</td>
<td>An f icon indicates that the access path points to a framework. Framework paths are implicitly recursive.</td>
</tr>
<tr>
<td></td>
<td>![ ]</td>
<td>No f icon indicates that the access path does not point to a framework.</td>
</tr>
<tr>
<td>Access path</td>
<td></td>
<td>Shows the full access path to the selected directory. Access paths have these types:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Absolute—the complete path, from the root level of the hard drive to the directory, including all intermediate directories</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Project—the path from the project file relative to the designated directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CodeWarrior—the path from the CodeWarrior IDE relative to the designated directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System—the path from the operating system's base directory relative to the designated directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Source tree—the path from a user-defined source tree relative to the designated directory</td>
</tr>
</tbody>
</table>

**Build Extras**

The **Build Extras** settings panel (Figure 28.4) contains options that define how the CodeWarrior IDE builds a project. Table 28.5 explains the items of this panel.
Working with Target Settings
Target Panels

Figure 28.4 Build Extras Settings Panel

Table 28.5 Build Extras Settings Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| 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 Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Browser Data From</td>
<td>Choose whether the IDE generates browser data for the project, and the method by which the IDE generates that data.</td>
</tr>
<tr>
<td>Dump internal browse information after compile</td>
<td>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.</td>
</tr>
<tr>
<td>Prefix file</td>
<td>Enter the path to your project’s prefix file. This option appears after selecting Language Parser from the Generate Browser Data From pop-up menu.</td>
</tr>
<tr>
<td>Macro file</td>
<td>Enter the path to your project’s macro file. This option appears after selecting Language Parser from the Generate Browser Data From pop-up menu.</td>
</tr>
<tr>
<td>Use External Debugger</td>
<td>Select to use an external debugger instead of the CodeWarrior debugger.</td>
</tr>
<tr>
<td>Application</td>
<td>Click Browse to select the external debugger application. Alternatively, enter the path to the external debugger.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Enter any program arguments to pass to the external debugger when the IDE transfers control.</td>
</tr>
<tr>
<td>Initial directory</td>
<td>Click Browse to select an initial directory for the external debugger. Alternatively, enter the path to the initial directory.</td>
</tr>
</tbody>
</table>

Runtime Settings

The Runtime Settings panel (Figure 28.5) specifies a debugging application for non-executable files. Dynamic linked libraries (DLLs), shared libraries, and code resources are sample non-executable files. Table 28.6 explains the items of this panel.
Figure 28.5 Runtime Settings Panel

Table 28.6 Runtime Settings Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host Application for Libraries &amp; Code Resources</strong></td>
<td>Click <strong>Choose</strong> to select the program for debugging non-executable files. Alternatively, enter the path to the application program. Click <strong>Clear</strong> to delete the current field entry.</td>
</tr>
<tr>
<td><strong>Working Directory</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>Enter a command line of program arguments to pass to the host application when the IDE transfers control.</td>
</tr>
<tr>
<td><strong>Environment Settings</strong></td>
<td>Lists the environment variables that have been added to the build target.</td>
</tr>
<tr>
<td><strong>Add</strong></td>
<td>Click to add the current <strong>Variable</strong> and <strong>Value</strong> pair to the Environment Settings list.</td>
</tr>
<tr>
<td><strong>Change</strong></td>
<td>Click to replace the selected entry in the Environment Settings list with the current <strong>Variable</strong> and <strong>Value</strong> pair.</td>
</tr>
</tbody>
</table>
Working with Target Settings

Target Panels

Table 28.6 Runtime Settings Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
<td>Click to delete the selected environment variable from the Environment Settings list.</td>
</tr>
<tr>
<td>Variable</td>
<td>Enter a name for the environment variable. This name pairs with the information in the Value field.</td>
</tr>
<tr>
<td>Value</td>
<td>Enter a value for the environment variable. This value pairs with the information in the Variable field.</td>
</tr>
</tbody>
</table>

File Mappings

The File Mappings settings panel (Figure 28.6) associates filename extensions with a CodeWarrior plugin compiler. These associations determine whether the IDE recognizes a source file by its filename extension or file type. Use this settings panel to add, change, and remove file mappings. Table 28.7 explains the items of this panel.

Figure 28.6 File Mappings Settings Panel

Table 28.6 Runtime Settings Panel Items (continued)

<table>
<thead>
<tr>
<th>File Type</th>
<th>Extension</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>.c</td>
<td>MW/ C/C++ x86</td>
</tr>
<tr>
<td>TEXT</td>
<td>.c++</td>
<td>MW/ C/C++ x86</td>
</tr>
<tr>
<td>TEXT</td>
<td>.cc</td>
<td>MW/ C/C++ x86</td>
</tr>
<tr>
<td>TEXT</td>
<td>.cpp</td>
<td>MW/ C/C++ x86</td>
</tr>
<tr>
<td>TEXT</td>
<td>.def</td>
<td>None</td>
</tr>
<tr>
<td>TEXT</td>
<td>.h</td>
<td>MW/ C/C++ x86</td>
</tr>
<tr>
<td>TEXT</td>
<td>.hpp</td>
<td>MW/ C/C++ x86</td>
</tr>
</tbody>
</table>
### Table 28.7  File Mappings Settings Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Mappings list</td>
<td></td>
<td>Displays a list of currently defined mappings between filename extensions and plugin compilers.</td>
</tr>
<tr>
<td><strong>File Type</strong></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Extension</strong></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Resource File flag</td>
<td>![bullet]</td>
<td>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.</td>
</tr>
<tr>
<td>Launcheable flag</td>
<td>![bullet]</td>
<td>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.</td>
</tr>
<tr>
<td>Precompiled File flag</td>
<td>![bullet]</td>
<td>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.</td>
</tr>
<tr>
<td>Ignored By Make flag</td>
<td>![x]</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td></td>
<td>Choose from this list the plugin compiler to associate with the selected file mapping. This compiler selection also appears in the corresponding column of the File Mappings list.</td>
</tr>
<tr>
<td>Flags</td>
<td>![checkmark]</td>
<td>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.</td>
</tr>
</tbody>
</table>
Working with Target Settings

Code Generation Panels

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 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 (Figure 28.7) configures how the compiler optimizes object code. All optimization routines rearrange object code without affecting its logical execution sequence. Table 28.8 explains the items of this panel.

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 CodeWarrior compilers. This panel is not appropriate for the Linux-hosted IDE, which uses gcc.

**Figure 28.7 Global Optimizations Settings Panel**

The *Details* field lists individual optimization routines applied at the selected optimization level. *Table 28.9* explains these optimizations and their availability at certain optimization levels.

**Table 28.8 Global Optimizations Settings Panel Items**

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster Execution Speed</td>
<td>Select to favor optimization routines that increase the execution speed of the final object code, at the expense of larger code size.</td>
</tr>
<tr>
<td>Smaller Code Size</td>
<td>Select to favor optimization routines that reduce the size of the final object code, at the expense of slower execution speed.</td>
</tr>
<tr>
<td>Optimization Level slider</td>
<td>Move to the desired optimization level. The IDE applies more optimization routines at higher optimization levels. The <em>Details</em> field lists the active optimization routines.</td>
</tr>
</tbody>
</table>
Table 28.9 Optimization Routines

<table>
<thead>
<tr>
<th>Optimization Routine</th>
<th>Explanation</th>
<th>Optimization Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Register Allocation or Global Register Allocation Only for Temporary Values</td>
<td>Stores working values of heavily used variables in registers instead of memory.</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>Dead Code Elimination</td>
<td>Removes statements never logically executed or referred to by other statements.</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>Branch Optimizations</td>
<td>Merges and restructures portions of the intermediate code translation in order to reduce branch instructions.</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>Arithmetic Operations</td>
<td>Replaces intensive computational instructions with faster equivalent instructions that produce the same result.</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>Expression Simplification</td>
<td>Replaces complex arithmetic expressions with simplified equivalent expressions.</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>Common Subexpression Elimination</td>
<td>Replaces redundant expressions with a single expression.</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>Copy Propagation or Copy and Expression Propagation</td>
<td>Replaces multiple occurrences of one variable with a single occurrence.</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>Peephole Optimization</td>
<td>Applies local optimization routines to small sections of code.</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>Dead Store Elimination</td>
<td>Removes assignments to a variable that goes unused before being reassigned again.</td>
<td>3, 4</td>
</tr>
<tr>
<td>Live Range Splitting</td>
<td>Reduces variable lifetimes to achieve optimal allocation. Shorter variable lifetimes reduce register spilling.</td>
<td>3, 4</td>
</tr>
</tbody>
</table>
Working with Target Settings
Editor Panels

Table 28.9 Optimization Routines (continued)

<table>
<thead>
<tr>
<th>Optimization Routine</th>
<th>Explanation</th>
<th>Optimization Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop-Invariant Code Motion</td>
<td>Moves static computations outside of a loop</td>
<td>3, 4</td>
</tr>
<tr>
<td>Strength Reduction</td>
<td>Inside loops, replaces multiplication instructions with addition instructions.</td>
<td>3, 4</td>
</tr>
<tr>
<td>Loop Transformations</td>
<td>Reorganizes loop object code in order to reduce setup and completion-test overhead.</td>
<td>3, 4</td>
</tr>
<tr>
<td>Loop Unrolling or Loop Unrolling (Opt for Speed Only)</td>
<td>Duplicates code inside a loop in order to spread branch and completion-test overhead over more operations.</td>
<td>3, 4</td>
</tr>
<tr>
<td>Vectorization</td>
<td>For processors that support vector optimizations, translates computations with code-loop arrays into equivalent vector instructions.</td>
<td>3, 4</td>
</tr>
<tr>
<td>Lifetime Based Register Allocation or Register Coloring</td>
<td>In a particular routine, uses the same processor register to store different variables, as long as no statement uses those variables simultaneously.</td>
<td>3, 4</td>
</tr>
<tr>
<td>Instruction Scheduling</td>
<td>Rearranges the instruction sequence to reduce conflicts among registers and processor resources.</td>
<td>3, 4</td>
</tr>
<tr>
<td>Repeated</td>
<td>Iterates the optimization routines listed between {* and *}.</td>
<td>4</td>
</tr>
</tbody>
</table>

Editor Panels

The Editor group of the Target Settings Panels provides a single core panel for configuring custom keywords within a project.
Working with Target Settings

Editor Panels

Custom Keywords

The Custom Keywords settings panel (Figure 28.8) 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. Table 28.10 explains the items of this panel.

Figure 28.8  Custom Keywords Settings Panel

<table>
<thead>
<tr>
<th>Custom Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword set 1</td>
</tr>
<tr>
<td>Keyword set 2</td>
</tr>
<tr>
<td>Keyword set 3</td>
</tr>
<tr>
<td>Keyword set 4</td>
</tr>
</tbody>
</table>

Table 28.10  Custom Keywords Settings Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword set 1, Keyword set 2,</td>
<td>Click a color swatch to set the color used for the corresponding custom-keyword set.</td>
</tr>
<tr>
<td>Keyword set 3, Keyword set 4</td>
<td></td>
</tr>
<tr>
<td>Edit</td>
<td>Click to add, modify, or remove keywords from the corresponding custom-keyword set.</td>
</tr>
</tbody>
</table>

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.

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 (Figure 28.9) configures additional executable files for the IDE to debug together with the current build target. Table 28.11 explains the items of this panel.
**Working with Target Settings**

**Debugger Panels**

---

**Figure 28.9  Other Executables Settings Panel**

![Other Executables Settings Panel](image)

**Table 28.11  Other Executables Settings Panel Items**

<table>
<thead>
<tr>
<th>Item</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File list</td>
<td></td>
<td>Lists executable files that the IDE can debug together with the current build target.</td>
</tr>
<tr>
<td>Debug column</td>
<td>🌿</td>
<td>Click in this column to toggle debugging of the corresponding executable file.</td>
</tr>
<tr>
<td>Add</td>
<td></td>
<td>Click to select an executable file to add to the File list.</td>
</tr>
<tr>
<td>Change</td>
<td></td>
<td>Click to change the selected entry in the File list.</td>
</tr>
<tr>
<td>Remove</td>
<td></td>
<td>Click to remove the selected entry in the File list.</td>
</tr>
</tbody>
</table>

---

**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 (Figure 28.10) configures activity logs, data-update intervals, and other debugger-related options. **Table 28.12** explains the items of this panel.

**Figure 28.10  Debugger Settings Panel**

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Location of Relocated Libraries and Code Resources</strong></td>
<td>Enter the path to code resources or relocated libraries required for debugging the project. Alternatively, click <strong>Choose</strong> to select the required files.</td>
</tr>
<tr>
<td><strong>Stop on application launch</strong></td>
<td>Select to halt program execution at the beginning of a debugging session. Select the desired stop point: <strong>Program entry point</strong>, <strong>Default language entry point</strong>, or <strong>User specified</strong>.</td>
</tr>
<tr>
<td><strong>Program entry point</strong></td>
<td>Select to halt program execution upon entering the program.</td>
</tr>
</tbody>
</table>
Remote Debugging

The Remote Debugging settings panel (Figure 28.11) 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. Table 28.13 explains the items of this panel.

Table 28.12 Debugger Settings Panel Items (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default language entry point</td>
<td>Select to halt program execution upon entering a default point defined by the programming language.</td>
</tr>
<tr>
<td>User specified</td>
<td>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.</td>
</tr>
<tr>
<td>Auto-target Libraries</td>
<td>Select to debug dynamically linked libraries (DLLs) loaded by the target application, at the expense of slower performance.</td>
</tr>
<tr>
<td>Cache symbolics between runs</td>
<td>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.</td>
</tr>
<tr>
<td>Log System Messages</td>
<td>Select to log all system messages to a Log window.</td>
</tr>
<tr>
<td>Stop at Watchpoints</td>
<td>Select to halt program execution at every watchpoint. Clear to halt program execution at watchpoints with changed values.</td>
</tr>
<tr>
<td>Update data every n seconds</td>
<td>Enter the number of seconds n to wait before updating the data displayed in debugging-session windows.</td>
</tr>
</tbody>
</table>
Working with Target Settings
Debugger Panels

Figure 28.11 Remote Debugging Settings Panel

Table 28.13 Remote Debugging Settings Panel Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable remote debugging</td>
<td>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.</td>
</tr>
<tr>
<td>Remote download path</td>
<td>Enter the path to the directory in which to store downloaded files.</td>
</tr>
<tr>
<td>Launch remote host application</td>
<td>Select to launch an application on the remote computer to serve as a host application. Enter the path to the remote application.</td>
</tr>
</tbody>
</table>
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.

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 “…”).
Preference and Target Settings Options

- 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

## 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

<table>
<thead>
<tr>
<th>On this host...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Press Alt while the IDE opens the symbolics file.</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Press Option while the IDE opens the symbolics file.</td>
</tr>
<tr>
<td>Solaris</td>
<td>Press Alt while the IDE opens the symbolics file.</td>
</tr>
<tr>
<td>Linux</td>
<td>Press Alt while the IDE opens the symbolics file.</td>
</tr>
</tbody>
</table>

## 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).
Preference and Target Settings Options

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.
Preference and Target Settings Options

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
- Purge Cache

Cache Subprojects
Use this option to improve multi-project updating and linking.
- 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.
Preference and Target Settings Options

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

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

**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>
Preference and Target Settings Options

- 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 plugins, 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 Main 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 plugin compiler or linker provides for the IDE. Use this option to help develop plugins 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.

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.
Preference and Target Settings Options

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.

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 lists default filename extensions.
### Table 29.2 Default Filename Extensions

<table>
<thead>
<tr>
<th>Type</th>
<th>Extension</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum CodeWarrior Installation</td>
<td>.iSYM</td>
<td>CodeWarrior Intel® Symbols</td>
</tr>
<tr>
<td></td>
<td>.mch</td>
<td>CodeWarrior Precompiled Header</td>
</tr>
<tr>
<td></td>
<td>.mcp</td>
<td>CodeWarrior Project File</td>
</tr>
<tr>
<td></td>
<td>.SYM</td>
<td>CodeWarrior Mac OS 68K Debug Symbols</td>
</tr>
<tr>
<td></td>
<td>.xSYM</td>
<td>CodeWarrior Mac OS PPC Debug Symbols</td>
</tr>
<tr>
<td></td>
<td>.dbg</td>
<td>CodeWarrior Debug Preferences</td>
</tr>
<tr>
<td></td>
<td>.exp</td>
<td>Exported Symbol File</td>
</tr>
<tr>
<td></td>
<td>.iMAP</td>
<td>CodeWarrior Link Map</td>
</tr>
<tr>
<td></td>
<td>.MAP</td>
<td>CodeWarrior Link Map</td>
</tr>
<tr>
<td>Assembly</td>
<td>.a</td>
<td>Assembly Source File (Windows and Macintosh)</td>
</tr>
<tr>
<td></td>
<td>.asm</td>
<td>Assembly Source File</td>
</tr>
<tr>
<td></td>
<td>.dump</td>
<td>CodeWarrior Disassembled File</td>
</tr>
<tr>
<td>C and C++</td>
<td>.c++</td>
<td>C++ Source File</td>
</tr>
<tr>
<td></td>
<td>.cc</td>
<td>C++ Source File</td>
</tr>
<tr>
<td></td>
<td>.hh</td>
<td>C++ Header File</td>
</tr>
<tr>
<td></td>
<td>.hpp</td>
<td>C++ Header File</td>
</tr>
<tr>
<td></td>
<td>.i</td>
<td>C Inline Source File</td>
</tr>
<tr>
<td></td>
<td>.icc</td>
<td>C++ Inline Source File</td>
</tr>
<tr>
<td></td>
<td>.m</td>
<td>Object C Source File</td>
</tr>
<tr>
<td></td>
<td>.mm</td>
<td>Object C++ Source File</td>
</tr>
</tbody>
</table>
### Preference and Target Settings Options

#### Table 29.2 Default Filename Extensions (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Extension</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default C and C++</td>
<td>.c</td>
<td>C Source File</td>
</tr>
<tr>
<td></td>
<td>.cp</td>
<td>C++ Source File</td>
</tr>
<tr>
<td></td>
<td>.cpp</td>
<td>C++ Source File</td>
</tr>
<tr>
<td></td>
<td>.h</td>
<td>C and C++ Header File</td>
</tr>
<tr>
<td>Default Java</td>
<td>.class</td>
<td>Java Class File</td>
</tr>
<tr>
<td></td>
<td>.jar</td>
<td>Java Archive File</td>
</tr>
<tr>
<td></td>
<td>.jav</td>
<td>Java Source File</td>
</tr>
<tr>
<td></td>
<td>.java</td>
<td>Java Source File</td>
</tr>
<tr>
<td>Java</td>
<td>.JMAP</td>
<td>Java Import Mapping Dump</td>
</tr>
<tr>
<td></td>
<td>.jpb</td>
<td>Java Constructor File</td>
</tr>
<tr>
<td></td>
<td>.mf</td>
<td>Java Manifest File</td>
</tr>
<tr>
<td>Library</td>
<td>.a</td>
<td>(Static) Archive Library (Solaris and Linux)</td>
</tr>
<tr>
<td></td>
<td>.lib</td>
<td>Library File</td>
</tr>
<tr>
<td></td>
<td>.o</td>
<td>Object File (Windows and Macintosh)</td>
</tr>
<tr>
<td></td>
<td>.o</td>
<td>Object (Relocatable) Library or Kernel Module</td>
</tr>
<tr>
<td></td>
<td>.obj</td>
<td>Object File (Solaris and Linux)</td>
</tr>
<tr>
<td></td>
<td>.pch</td>
<td>Precompiled Header Source File</td>
</tr>
<tr>
<td></td>
<td>.pch++</td>
<td>Precompiled Header Source File</td>
</tr>
<tr>
<td></td>
<td>.so</td>
<td>Shared Library (Linux)</td>
</tr>
<tr>
<td>Script</td>
<td>.sh</td>
<td>Shell Script (Linux)</td>
</tr>
<tr>
<td></td>
<td>.psh</td>
<td>Precompile Shell Script (Linux)</td>
</tr>
<tr>
<td></td>
<td>.pl</td>
<td>Perl Script (Linux)</td>
</tr>
</tbody>
</table>

426 IDE 5.7 User’s Guide
Preference and Target Settings Options

Table 29.2  Default Filename Extensions (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Extension</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS X</td>
<td>.dylib</td>
<td>Mach-O Dynamic Library</td>
</tr>
<tr>
<td></td>
<td>.a</td>
<td>Mach-O Static Library</td>
</tr>
<tr>
<td></td>
<td>.o</td>
<td>Mach-O Object File</td>
</tr>
<tr>
<td></td>
<td>.plist</td>
<td>Property List</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Extension</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.dylib</td>
<td>Mach-O Dynamic Library</td>
</tr>
<tr>
<td></td>
<td>.a</td>
<td>Mach-O Static Library</td>
</tr>
<tr>
<td></td>
<td>.o</td>
<td>Mach-O Object File</td>
</tr>
<tr>
<td></td>
<td>.plist</td>
<td>Property List</td>
</tr>
</tbody>
</table>
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.
Preference and Target Settings Options

G-I

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:
Preference and Target Settings Options

- 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 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.
Preference and Target Settings Options

**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.

**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.
Preference and Target Settings Options

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:

```bash
runemacs %file
```

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:

```bash
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 plugins in development. This information is useful for diagnosing plugin behavior or for viewing information about the properties of installed plugins.

Choose one of these levels of plugin diagnostic information:

- **None** (default)—The IDE does not activate plugin diagnostics or produce output.
Preference and Target Settings Options

- **Errors Only**—The IDE reports problems encountered while loading plugins. These problems appear in a new text file after the IDE starts up.
- **All Info**—The IDE reports information for each installed plugin, such as problems with plugin loading, optional plugin information, and plugin properties. This information appears in a new text file after the IDE starts up. The text file also contains a complete list of installed plugins 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 plugins. The text file also provides suggestions for correcting general plugin errors.

**Linker**

Use the **Linker** option menu to select the linker to use with the project. The choices available are always dependent on the plugin 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.

**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.
Preference and Target Settings Options

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](#)

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.
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.

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 and unsuccessful builds using the Success and Failure pop-up options, respectively.

See also:
  - Failure
  - Success

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.
Preference and Target Settings Options

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.
Preference and Target Settings Options

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.


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:
Preference and Target Settings Options

- **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.

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.

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.
Preference and Target Settings Options

- **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 **Listing 29.1** function:

**Listing 29.1 Hidden Locals Example**

```c
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;
        
    }
}
```
When debugging this function, the earlier CodeWarrior debugger would show five variables at all times:

\[
\begin{align*}
\text{loop1} & : \text{int} \\
\text{varA} & : \text{char} \\
\text{varA} & : \text{char} \\
\text{varB} & : \text{int} \\
\text{varB} & : \text{float}
\end{align*}
\]

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:

\[
\begin{align*}
\text{varA} & : \text{char} \\
\text{varB} & : \text{int}
\end{align*}
\]

and the display will look like this when inside the for loop:

\[
\begin{align*}
\text{loop1} & : \text{int} \\
\text{varA} & : \text{char} \\
\text{varB} & : \text{float}
\end{align*}
\]

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, check the “Show hidden locals” option; the variable list when inside the for loop looks like this:

\[
\begin{align*}
\text{loop1} & : \text{int} \\
\text{varA} & : \text{char} \quad // \text{displayed as dimmed} \\
\text{varA} & : \text{char} \\
\text{varB} & : \text{int} \quad // \text{displayed as dimmed} \\
\text{varB} & : \text{float}
\end{align*}
\]

**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”
Preference and Target Settings Options

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.
Preference and Target Settings Options

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:

```c
struct Rect {
    short top;
    short left;
    short bottom;
    short right;
};
```

then normally a variable of that type would look like this in the debugger:

```plaintext
myRect 0x000DCEA8
```

If the Variable Formatter is given a format that looks like this:

```xml
<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:

```plaintext
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 “Variable Formats” 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 “cpuix86”.

- `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` - Specifies 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.
Preference and Target Settings Options

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.

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.
Preference and Target Settings Options

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.

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 overlapped 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.
Preference and Target Settings Options

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.
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.
Preference and Target Settings Options

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
- Launch Editor w/ Line #

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.

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.
Preference and Target Settings Options

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.

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.
Preference and Target Settings Options

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.
Register Details Window XML Specification

XML consists 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.

Attributes can include numeric values, which you may enter in a variety of formats (Table 30.1). 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’
Register Details Window XML Specification

Every XML file created for use with the Register Details window must conform to the specification shown in Listing 30.1. The specification defines the following types of elements:

- **REGISTER**
- **BITFIELD**
- **BFVALUE**

Descriptions of these elements follow the specification.

### Listing 30.1 Register Details Window XML Specification

```xml
<!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>```

### Table 30.1 Numeric Attribute Value Formats

<table>
<thead>
<tr>
<th>Value Format</th>
<th>Sample Values</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>123 and -334</td>
<td></td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>0x12 and 0X12</td>
<td>Preceded by 0x or 0X</td>
</tr>
<tr>
<td>Octal</td>
<td>012</td>
<td>Preceded by 0</td>
</tr>
<tr>
<td>Binary</td>
<td>0b11001 and 0B11</td>
<td>Preceded by 0b or 0B</td>
</tr>
<tr>
<td>Character</td>
<td>'c' and 'XYZ'</td>
<td>Enclosed in ''</td>
</tr>
</tbody>
</table>
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>
```

**Attributes**

**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.

You must enter a NAME attribute in the form

```xml
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

\[ MSB:LSB \]

\[ BitNumber \]

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 30.1 on page 454 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.

<table>
<thead>
<tr>
<th>BITFIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

**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"
   BITRANGE      =   "(MSB:LSB|Number)"
   FORMAT        =   "BitfieldFormat"
   ACCESS        =   "AccessFormat"
   CONDITION     =   "Expression"
   DESCRIPTION   =   "BitfieldDescription">
</BITFIELD>
```

**Attributes**

**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` 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.

**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 30.2). 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`. 
Register Details Window XML Specification

**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"
```

---

<table>
<thead>
<tr>
<th>Expression</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CONDITION = &quot;$$&amp;0x80&quot;</code></td>
<td>The current register value ANDed with 0x80</td>
</tr>
<tr>
<td><code>CONDITION = &quot;!($$&amp;0x80)&quot;</code></td>
<td>The inversion of the current register value ANDed with 0x80</td>
</tr>
<tr>
<td><code>CONDITION = &quot;$MSR&amp;0x8000&quot;</code></td>
<td>Another register value (the MSR register) ANDed with 0x8000</td>
</tr>
<tr>
<td><code>CONDITION = &quot;foo&amp;0x10 &gt;= 0&quot;</code></td>
<td>An expression using a variable named foo</td>
</tr>
</tbody>
</table>
Register Details Window XML Specification

Accessing the XML Files from CodeWarrior

**Attributes**

**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
- 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:
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 30.2 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 30.2 Sample Base XML File

```xml
<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>
```

The IDE looks up system registers by name. Memory-mapped registers can have any name, since that name is used for display purposes only.
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 30.5.

### 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 30.2, 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, your first revision of the
base file might appear as shown in Listing 30.3.

#### Listing 30.3 First Revision of Base XML File

```xml
<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>

</REGISTER>
```
<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"
  NAME="V"
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, your second revision of the base file might resemble Listing 30.4.

Listing 30.4 Second Revision of XML Base File

```xml
<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."
    <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." />}
```
Register Details Window XML Specification
A Sample XML File

```xml
<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." />
  <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 of previous sections, Listing 30.5 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.

Listing 30.5 Sophisticated XML File

<REGISTER NAME="BR2"
  BITRANGE="0:31"
  DESCRIPTION="Memory Controller Base Register 2.">

  <BITFIELD
    BITRANGE="0:16"
    NAME="BA"
Register Details Window XML Specification

A Sample XML File

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 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 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 if this bit is set."/>
Register Details Window XML Specification

References

```xml
</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 these references:

Menus

This section contains these chapters:

- IDE Menus
- Menu Commands
IDE Menus

This chapter is 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**
- **Macintosh Menu Layout**

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. **Table 31.1** explains the commands of this menu.

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Creates new projects using the New Project wizard or project stationery files.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens source and project files for editing and project modification operations.</td>
</tr>
</tbody>
</table>
### Table 31.1 File Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find and Open File</td>
<td>Opens the file specified in the Find and Open File dialog or from the selected text in the active window.</td>
</tr>
<tr>
<td></td>
<td>When using the Windows menu layout on a Macintosh host, hold down the Option key to change this command to Find.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the active window.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves the active file using the editor window's filename.</td>
</tr>
<tr>
<td></td>
<td>When using the Windows menu layout on a Macintosh host, hold down the Option key to change this command to Save All.</td>
</tr>
<tr>
<td>Save All</td>
<td>Saves all open editor windows.</td>
</tr>
<tr>
<td></td>
<td>When using the Windows menu layout on a Macintosh host, hold down the Option key to substitute this command for the Save command.</td>
</tr>
<tr>
<td>Save As</td>
<td>Saves a copy of the active file under a new name and closes the original file.</td>
</tr>
<tr>
<td>Save A Copy As</td>
<td>Saves a copy of the active file without closing the file.</td>
</tr>
<tr>
<td>Revert</td>
<td>Discards all changes made to the active file since the last save operation.</td>
</tr>
<tr>
<td>Open Workspace</td>
<td>Opens a workspace that you previously saved.</td>
</tr>
<tr>
<td>Close Workspace</td>
<td>Closes the current workspace. (You cannot close the default workspace.)</td>
</tr>
<tr>
<td>Save Workspace</td>
<td>Saves the current state of onscreen windows, recent items, and debugging.</td>
</tr>
<tr>
<td>Save Workspace As</td>
<td>Saves an existing workspace under a different name.</td>
</tr>
<tr>
<td>Import Components</td>
<td>Imports the components from another catalog into the current catalog.</td>
</tr>
<tr>
<td>Close Catalog</td>
<td>Closes the current catalog and its associated Catalog Components window and Component Palette.</td>
</tr>
<tr>
<td>Import Project</td>
<td>Imports a project file previously saved in extensible markup language format (XML) and converts it into project file format.</td>
</tr>
</tbody>
</table>
IDE Menus
Windows Menu Layout

Table 31.2  Edit Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Undo</strong></td>
<td>Undoes the last cut, paste, clear, or typing operation. If you cannot undo the action, this command changes to Can’t Undo.</td>
</tr>
<tr>
<td><strong>Redo</strong></td>
<td>Redoes the action of the last Undo operation. If you cannot redo the action, this command changes to Can’t Redo.</td>
</tr>
<tr>
<td><strong>Cut</strong></td>
<td>Removes the selected text and places a copy of it on the Clipboard.</td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td>Copies the selected text and places a copy of it on the Clipboard.</td>
</tr>
</tbody>
</table>
### IDE Menus

*Windows Menu Layout*

#### Table 31.2 Edit Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Paste</strong></td>
<td>Places the contents of the Clipboard at current insertion point or replaces the selected text.</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Select All</strong></td>
<td>Selects all text in current editor window or text box for cut, copy, paste, clear, or typing operations.</td>
</tr>
<tr>
<td><strong>Balance</strong></td>
<td>Selects text between the nearest set of parenthesis, braces, or brackets.</td>
</tr>
<tr>
<td><strong>Shift Left</strong></td>
<td>Moves selected text one tab stop to the left.</td>
</tr>
<tr>
<td><strong>Shift Right</strong></td>
<td>Moves selected text one tab stop to the right.</td>
</tr>
<tr>
<td><strong>Get Previous Completion</strong></td>
<td>Shortcut for selecting the previous item that appears in the Code Completion window.</td>
</tr>
<tr>
<td><strong>Get Next Completion</strong></td>
<td>Shortcut for selecting the next item that appears in the Code Completion window.</td>
</tr>
<tr>
<td><strong>Complete Code</strong></td>
<td>Opens the Code Completion window.</td>
</tr>
<tr>
<td><strong>Preferences</strong></td>
<td>Opens the IDE Preferences window where you can set general IDE, editor, debugger, and layout options.</td>
</tr>
<tr>
<td><strong>Target Settings</strong></td>
<td>Opens the project’s Target Settings window where you can set target, language, code generation, linker, editor, and debugger options. (the name changes, based on the name of the active build target)</td>
</tr>
<tr>
<td><strong>Version Control Settings</strong></td>
<td>Opens the VCS Settings window to enable activation of a version control system and its relevant settings.</td>
</tr>
<tr>
<td><strong>Commands &amp; Key Bindings</strong></td>
<td>Opens the Customize IDE Commands window where you can create, modify, remove menus, menu commands, and key bindings.</td>
</tr>
</tbody>
</table>
View Menu

The **View** menu contains commands for viewing toolbars, the class browser, the Message window, and debugging windows. **Table 31.3** explains the command of this menu.

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolbars</td>
<td>Use the Toolbars menu to show, hide, reset, and clear window and main toolbars.</td>
</tr>
<tr>
<td>Project Inspector</td>
<td>Opens or brings to the front a Project Inspector window.</td>
</tr>
<tr>
<td>Browser Contents</td>
<td>Opens or brings to the front a Browser Contents window.</td>
</tr>
<tr>
<td>Class Browser</td>
<td>Opens or brings to the front a New Class Browser window.</td>
</tr>
<tr>
<td>Class Hierarchy or Class Hierarchy Window</td>
<td>Opens or brings to the front a Class Hierarchy window.</td>
</tr>
<tr>
<td>Build Progress or Build Progress Window</td>
<td>Opens the Build Progress window.</td>
</tr>
<tr>
<td>Errors &amp; Warnings or Errors &amp; Warnings Window</td>
<td>Opens or brings to the front an Errors &amp; Warnings window.</td>
</tr>
<tr>
<td>Symbolics or Symbolics Window</td>
<td>Opens the Symbolics window.</td>
</tr>
<tr>
<td>Processes or Processes Window</td>
<td>Opens or brings to the front a Processes window.</td>
</tr>
<tr>
<td>Breakpoints or Breakpoints Window</td>
<td>Opens or brings to the front the Breakpoints window. Use this window to view, create, modify, and remove breakpoints.</td>
</tr>
<tr>
<td>Registers or Register Window</td>
<td>Opens or brings to the front a Register window.</td>
</tr>
<tr>
<td>Expressions or Expressions Window</td>
<td>Opens or brings to the front an Expressions window. Use to view, create, modify, and remove expressions.</td>
</tr>
<tr>
<td>Global Variables or Global Variables Window</td>
<td>Opens or brings to the front a Global Variables window.</td>
</tr>
</tbody>
</table>
IDE Menus
Windows Menu Layout

Search Menu

The Search menu contains commands for finding text, replacing text, comparing files, and navigating code. Table 31.4 explains the commands of this menu.

Table 31.4 Search Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td>Opens the Find and Replace window for performing searches in the active editor window.</td>
</tr>
<tr>
<td>Replace Selection</td>
<td>Opens the Find and Replace window for replacing text in the active editor window.</td>
</tr>
<tr>
<td>Find in Files</td>
<td>Opens the Find in Files window for performing searches in the active editor window.</td>
</tr>
<tr>
<td>Find Next</td>
<td>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.</td>
</tr>
<tr>
<td>Find In Next File</td>
<td>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.</td>
</tr>
<tr>
<td>Enter Find String</td>
<td>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.</td>
</tr>
<tr>
<td>Find Selection</td>
<td>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.</td>
</tr>
<tr>
<td>Replace Selection</td>
<td>Replaces the replace string in the Replace text box with the selected text.</td>
</tr>
</tbody>
</table>
The Project menu contains commands for manipulating files, handling libraries, compiling projects, building projects, and linking projects. Table 31.5 explains the commands of this menu.
### IDE Menus

#### Windows Menu Layout

**Table 31.5  Project Menu Commands**

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Window</td>
<td>Adds the active window to the project.</td>
</tr>
<tr>
<td>Add Files</td>
<td>Opens a dialog box that you can use to add multiple files to the active project.</td>
</tr>
<tr>
<td>Create Group</td>
<td>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.</td>
</tr>
<tr>
<td>Create Target</td>
<td>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.</td>
</tr>
<tr>
<td>Create Overlay or Create Segment</td>
<td>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.</td>
</tr>
<tr>
<td>Create Design</td>
<td>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.</td>
</tr>
<tr>
<td>Export Project as GNU Makefile</td>
<td>Exports the current project to a GNU makefile.</td>
</tr>
<tr>
<td>Check Syntax</td>
<td>Checks the active editor window or selected files in the project window for compilation errors.</td>
</tr>
<tr>
<td>Preprocess</td>
<td>Preprocesses the active editor window or selected files in the project window and displays results in a new editor window.</td>
</tr>
<tr>
<td>Precompile</td>
<td>Precompiles the active editor window or selected files in the project window and stores results in a new header file.</td>
</tr>
<tr>
<td>Compile</td>
<td>Compiles the active editor window or selected files in the project window.</td>
</tr>
<tr>
<td>Disassemble</td>
<td>Disassembles the active editor window or selected files in the project window and displays results in a new editor window.</td>
</tr>
<tr>
<td>Bring Up To Date</td>
<td>Compiles all marked or modified files in the current build target of the active project.</td>
</tr>
</tbody>
</table>
### Table 31.5 Project Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
<td>Compiles and links all marked or modified files in the current build target of the active project, saving the executable file.</td>
</tr>
<tr>
<td>Stop Build</td>
<td>Stops the current compile and linking operation and cancels the remainder of the build process.</td>
</tr>
<tr>
<td>Remove Object Code</td>
<td>Removes object code from one or more build targets in the project.</td>
</tr>
<tr>
<td></td>
<td>When using the Windows menu layout on a Macintosh host, hold down the Shift key to change this command to Remove Object Code &amp; Compact.</td>
</tr>
<tr>
<td>Re-search for Files</td>
<td>Resets the cached locations of source files using the project access paths, and stores them for faster builds and project operations.</td>
</tr>
<tr>
<td>Reset Project Entry Paths</td>
<td>Resets the location of all source files in the active project using the project access paths.</td>
</tr>
<tr>
<td>Synchronize Modification Dates</td>
<td>Updates the modification dates of all source files in the active project.</td>
</tr>
<tr>
<td>Debug</td>
<td>Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.</td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>Resume</td>
<td>Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.</td>
</tr>
<tr>
<td>Run</td>
<td>Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.</td>
</tr>
<tr>
<td>Set Default Project</td>
<td>Uses the Set Default Project menu to choose the default project when more than one project is open in the IDE.</td>
</tr>
<tr>
<td>Set Default Target</td>
<td>Uses the Set Default Target menu to choose the default build target when more than one build target is present in the project file.</td>
</tr>
</tbody>
</table>
Debug Menu

The Debug menu contains commands for managing program execution. Table 31.6 explains the commands of this menu.

Table 31.6 Debug Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break</td>
<td>Pauses execution of the program in a debugging session to enable examination of register and variable contents</td>
</tr>
<tr>
<td>Kill</td>
<td>Terminates the current debugging session returning control to the IDE.</td>
</tr>
<tr>
<td>Restart</td>
<td>Terminates the current debugging session, then restarts the program from the beginning.</td>
</tr>
<tr>
<td>Step Over</td>
<td>Executes each source line in the program, treating routine calls as a single statement and stopping the program at the next line of code.</td>
</tr>
<tr>
<td>Step Into</td>
<td>Executes each source line in the program, following any subroutine calls.</td>
</tr>
<tr>
<td>Step Out</td>
<td>Executes each source line in the subroutine and stops the program when the routine returns to its caller.</td>
</tr>
<tr>
<td>Run to Cursor</td>
<td>Sets a temporary breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Change Program Counter</td>
<td>Opens the Change Program Counter dialog box that you can use to move the current statement arrow to an address or symbol.</td>
</tr>
<tr>
<td>Set Breakpoint or Clear Breakpoint</td>
<td>Sets a breakpoint on the source line containing the insertion point. Clears the breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Set Eventpoint</td>
<td>Sets an eventpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Clear Eventpoint</td>
<td>Clears the breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Set/Clear Breakpoint</td>
<td>Opens the Set/Clear Breakpoint dialog box that you can use for setting or clearing breakpoints by address or symbol.</td>
</tr>
</tbody>
</table>
The Data menu contains commands that control how the CodeWarrior debugger displays data values. This menu appears only during a debugging session. Table 31.7 explains the commands of this menu.
### Table 31.7 Data Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Types</strong></td>
<td>Toggles the appearance of the data type on local and global variables displayed in Variable panes and Variable windows.</td>
</tr>
<tr>
<td><strong>Refresh All Data</strong></td>
<td>Updates data displays.</td>
</tr>
<tr>
<td><strong>New Expression</strong></td>
<td>Creates a new expression entry in the Expressions window.</td>
</tr>
<tr>
<td><strong>Copy to Expression</strong></td>
<td>Copies the selected variable to the Expressions window.</td>
</tr>
<tr>
<td><strong>View As</strong></td>
<td>Displays the View As dialog where the data type of the selected variable can be specified.</td>
</tr>
<tr>
<td><strong>View Variable</strong></td>
<td>Displays the selected variable in a new Variables window.</td>
</tr>
<tr>
<td><strong>View Array</strong></td>
<td>Displays the selected array variable in a new Arrays window.</td>
</tr>
<tr>
<td><strong>View Memory</strong></td>
<td>Displays the selected variable in a new Memory window.</td>
</tr>
<tr>
<td><strong>View Memory As</strong></td>
<td>Displays the View As dialog where the data type of the selected variable can be specified, then shown in a new Memory window.</td>
</tr>
<tr>
<td><strong>Cycle View</strong></td>
<td>Toggles the data view among <strong>View Source</strong>, <strong>View Disassembly</strong>, <strong>View Mixed</strong>, and <strong>View Raw Data</strong>.</td>
</tr>
<tr>
<td><strong>View Source</strong></td>
<td>View data as source code.</td>
</tr>
<tr>
<td><strong>View Disassembly</strong></td>
<td>View data as language disassembly.</td>
</tr>
<tr>
<td><strong>View Mixed</strong></td>
<td>View data as source code and its disassembly.</td>
</tr>
<tr>
<td><strong>View Raw Data</strong></td>
<td>View data without applied formatting.</td>
</tr>
<tr>
<td><strong>View As Default</strong></td>
<td>Views the selected variable in the default value format.</td>
</tr>
<tr>
<td><strong>View As Binary</strong></td>
<td>Views the selected variable as a binary value.</td>
</tr>
<tr>
<td><strong>View As Signed Decimal</strong></td>
<td>Views the selected variable as a signed decimal value.</td>
</tr>
<tr>
<td><strong>View As Unsigned Decimal</strong></td>
<td>Views the selected variable as an unsigned decimal value.</td>
</tr>
<tr>
<td><strong>View As Hexadecimal</strong></td>
<td>Views the selected variable as a hexadecimal value.</td>
</tr>
</tbody>
</table>
IDE Menus
Windows Menu Layout

Table 31.8  Window Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>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.</td>
</tr>
<tr>
<td>Close All</td>
<td>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.</td>
</tr>
<tr>
<td>Cascade</td>
<td>Arranges all editor windows so that only the title bar is visible.</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>Tiles all editor windows horizontally on the screen so none overlap.</td>
</tr>
</tbody>
</table>

Table 31.7  Data Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>View As Character</td>
<td>Views the selected variable as a character value.</td>
</tr>
<tr>
<td>View As C String</td>
<td>Views the selected variable as a C string.</td>
</tr>
<tr>
<td>View As Pascal String</td>
<td>Views the selected variable as a Pascal string.</td>
</tr>
<tr>
<td>View As Unicode String</td>
<td>Views the selected variable as a Unicode string.</td>
</tr>
<tr>
<td>View As Floating Point</td>
<td>Views the selected variable as a floating point value.</td>
</tr>
<tr>
<td>View As Enumeration</td>
<td>Views the selected variable as an enumerated value.</td>
</tr>
<tr>
<td>View As Fixed</td>
<td>Views the selected variable as a 32-bit fixed value.</td>
</tr>
</tbody>
</table>

Window Menu

The Window menu contains commands that manipulate IDE windows. Table 31.8 explains the commands of this menu.

The Window menu also lists the names of all open file and project windows. A checkmark appears beside the active window; an underline indicates a modified and unsaved file.
IDE Menus

Macintosh Menu Layout

Table 31.8 Window Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile Vertically</td>
<td>Tiles all editor windows vertically on the screen so none overlap.</td>
</tr>
<tr>
<td>Save Default Window</td>
<td>Saves the active browser windows settings and applies it to other browser windows as they are opened.</td>
</tr>
</tbody>
</table>

Help Menu

The Help menu contains commands for accessing the IDE’s online help. Table 31.9 explains the commands of this menu.

Table 31.9 Help Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeWarrior Help</td>
<td>Launches a help viewer to display the online help. Click on a link to view a specific IDE topic.</td>
</tr>
<tr>
<td>Index</td>
<td>Launches a help viewer to display a glossary of common terms used in the CodeWarrior help and manuals.</td>
</tr>
<tr>
<td>Search</td>
<td>Launches a help viewer to a page for searching the CodeWarrior help and manuals.</td>
</tr>
<tr>
<td>Freescale CodeWarrior Website</td>
<td>Launches a browser and automatically points you to the web site.</td>
</tr>
<tr>
<td>About Freescale CodeWarrior</td>
<td>Displays the CodeWarrior IDE version and build number information.</td>
</tr>
</tbody>
</table>

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 Freescale 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. Table 31.10 explains the commands of this menu.

If you use the Macintosh menu layout on a Windows host, this menu does not appear.

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Freescale CodeWarrior</td>
<td>Displays the CodeWarrior IDE version and build number information.</td>
</tr>
<tr>
<td>Preferences</td>
<td>Opens the IDE Preferences window where you can set general IDE, editor, debugger, and layout options.</td>
</tr>
<tr>
<td>Quit or Quit CodeWarrior</td>
<td>Quits the CodeWarrior IDE.</td>
</tr>
</tbody>
</table>

**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 31.11 explains the commands of this menu.

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Text File</td>
<td>Creates a new text file and displays it in a new editor window.</td>
</tr>
<tr>
<td>New</td>
<td>Creates new projects using the New Project wizard or project stationery files.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens source and project files for editing and project modification operations.</td>
</tr>
<tr>
<td>Open Recent</td>
<td>Displays a submenu of recently opened files and projects that can be chosen to open in the IDE.</td>
</tr>
</tbody>
</table>
Table 31.11 File Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find and Open File</strong></td>
<td>Opens the file specified in the Find and Open File dialog or from the selected text in the active window.</td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Save</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Save As</strong></td>
<td>Saves a copy of the active file under a new name and closes the original file.</td>
</tr>
<tr>
<td><strong>Save A Copy As</strong></td>
<td>Saves a copy of the active file without closing the file.</td>
</tr>
<tr>
<td><strong>Revert</strong></td>
<td>Discards all changes made to the active file since the last save operation.</td>
</tr>
<tr>
<td><strong>Open Workspace</strong></td>
<td>Opens a workspace that you previously saved.</td>
</tr>
<tr>
<td><strong>Close Workspace</strong></td>
<td>Closes the current workspace. (You cannot close the default workspace.)</td>
</tr>
<tr>
<td><strong>Save Workspace</strong></td>
<td>Saves the current state of onscreen windows, recent items, and debugging.</td>
</tr>
<tr>
<td><strong>Save Workspace As</strong></td>
<td>Saves an existing workspace under a different name.</td>
</tr>
<tr>
<td><strong>Import Components</strong></td>
<td>Imports the components from another catalog into the current catalog.</td>
</tr>
<tr>
<td><strong>Close Catalog</strong></td>
<td>Closes the current catalog and its associated Catalog Components window and Component Palette.</td>
</tr>
<tr>
<td><strong>Import Project</strong></td>
<td>Imports a project file previously saved in extensible markup language format (XML) and converts it into project file format.</td>
</tr>
<tr>
<td><strong>Export Project</strong></td>
<td>Exports the active project file to disk in XML format.</td>
</tr>
<tr>
<td><strong>Page Setup</strong></td>
<td>Displays the Page Setup dialog for setting paper size, orientation, and other printer options.</td>
</tr>
</tbody>
</table>
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 31.12 explains the commands of this menu.

Table 31.12  Edit Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| **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. |
### Table 31.12 Edit Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Select All</strong></td>
<td>Selects all text in the current editor window or text box for cut, copy, paste, clear, or typing operations.</td>
</tr>
<tr>
<td><strong>Balance</strong></td>
<td>Selects text between the nearest set of parenthesis, braces, or brackets.</td>
</tr>
<tr>
<td><strong>Shift Left</strong></td>
<td>Moves selected text one tab stop to the left.</td>
</tr>
<tr>
<td><strong>Shift Right</strong></td>
<td>Moves selected text one tab stop to the right.</td>
</tr>
<tr>
<td><strong>Get Previous Completion</strong></td>
<td>Shortcut for selecting the previous item that appears in the Code Completion window.</td>
</tr>
<tr>
<td><strong>Get Next Completion</strong></td>
<td>Shortcut for selecting the next item that appears in the Code Completion window.</td>
</tr>
<tr>
<td><strong>Complete Code</strong></td>
<td>Opens the Code Completion window.</td>
</tr>
<tr>
<td><strong>Insert Reference Template</strong></td>
<td>Inserts a routine template corresponding to the selected Mac OS Toolbox call in the active window.</td>
</tr>
<tr>
<td></td>
<td>When using the Macintosh menu layout on a Windows host, this command does not appear.</td>
</tr>
<tr>
<td><strong>Preferences</strong></td>
<td>Opens the IDE Preferences window where you can set general IDE, editor, debugger, and layout options.</td>
</tr>
<tr>
<td><strong>Target Settings</strong></td>
<td>Opens the project’s Target Settings window where you can set target, language, linker, editor, and debugger options.</td>
</tr>
<tr>
<td>(name changes, based on name of active build target)</td>
<td></td>
</tr>
<tr>
<td><strong>Version Control Settings</strong></td>
<td>Opens the VCS Settings window to enable activation of a version control system and its relevant settings.</td>
</tr>
<tr>
<td><strong>Commands &amp; Key Bindings</strong></td>
<td>Opens the Customize IDE Commands window where you can create, modify, remove menus, menu commands, and key bindings.</td>
</tr>
</tbody>
</table>

### Search Menu

The **Search** menu contains commands for finding text, replacing text, comparing files, navigating code, and finding routine definitions. **Table 31.13** explains the commands of this menu.
Table 31.13 Search Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find and Replace</td>
<td>Opens the Find and Replace window for performing find and replace operations on the active editor window.</td>
</tr>
<tr>
<td>Find in Files</td>
<td>Opens the Find in Files window for performing searches in the active editor window.</td>
</tr>
<tr>
<td>Find Next</td>
<td>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.</td>
</tr>
<tr>
<td>Find In Next File</td>
<td>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.</td>
</tr>
<tr>
<td>Enter Find String</td>
<td>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.</td>
</tr>
<tr>
<td>Find Selection</td>
<td>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.</td>
</tr>
<tr>
<td>Replace Selection</td>
<td>Replaces the replace string in the Replace text box with the selected text.</td>
</tr>
<tr>
<td>Replace and Find Next</td>
<td>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.</td>
</tr>
<tr>
<td>Replace All</td>
<td>Finds all matches of the Find text box string and replaces them with the Replace text box string.</td>
</tr>
</tbody>
</table>
### IDE Menus

**Macintosh Menu Layout**

#### Table 31.13 Search Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| **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**. |
| **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 31.14 explains the commands of this menu.
Table 31.14  Project Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add Window</strong></td>
<td>Adds the active window to the project.</td>
</tr>
<tr>
<td>(name changes, based on</td>
<td></td>
</tr>
<tr>
<td>name of selected item)</td>
<td></td>
</tr>
<tr>
<td><strong>Add Files</strong></td>
<td>Opens a dialog that you can use to add multiple files to the active project.</td>
</tr>
<tr>
<td><strong>Create Group</strong></td>
<td>Displays the Create Group dialog where you can add a new file group to the</td>
</tr>
<tr>
<td>or</td>
<td>active project immediately after selected file or group.</td>
</tr>
<tr>
<td><strong>Create Target</strong></td>
<td>Displays the Create Target dialog where you can add a new build target to</td>
</tr>
<tr>
<td>or</td>
<td>the active project immediately after selected build target.</td>
</tr>
<tr>
<td>**Create Overlay or Create</td>
<td>Displays the Create Overlay dialog where you can add a new memory overlay</td>
</tr>
<tr>
<td>Segment**</td>
<td>to the active project immediately after the selected overlay.</td>
</tr>
<tr>
<td>or</td>
<td>Displays the Create Segment dialog where you can add a new segment to the</td>
</tr>
<tr>
<td>**Create Overlay or Create</td>
<td>active project immediately after the selected segment.</td>
</tr>
<tr>
<td>Segment**</td>
<td></td>
</tr>
<tr>
<td><strong>Create Design</strong></td>
<td>Opens the Create New Design dialog box that you can use to add a design to</td>
</tr>
<tr>
<td></td>
<td>the active project. The new design appears in the <strong>Design</strong> tab of the</td>
</tr>
<tr>
<td></td>
<td>project window.</td>
</tr>
<tr>
<td><strong>Check Syntax</strong></td>
<td>Checks the active editor window or selected files in the project window for</td>
</tr>
<tr>
<td></td>
<td>compilation errors.</td>
</tr>
<tr>
<td><strong>Preprocess</strong></td>
<td>Preprocesses the active editor window or selected files in the project</td>
</tr>
<tr>
<td></td>
<td>window and displays results in a new editor window.</td>
</tr>
<tr>
<td><strong>Precompile</strong></td>
<td>Precompiles active editor window or selected files in project window and</td>
</tr>
<tr>
<td></td>
<td>stores results in a new header file.</td>
</tr>
<tr>
<td><strong>Compile</strong></td>
<td>Compiles the active editor window or selected files in project window.</td>
</tr>
<tr>
<td><strong>Disassemble</strong></td>
<td>Disassembles the active editor window or selected files in the project</td>
</tr>
<tr>
<td></td>
<td>window and displays the results in a new editor window.</td>
</tr>
<tr>
<td><strong>Bring Up To Date</strong></td>
<td>Compiles all marked or modified files in the current build target of the</td>
</tr>
<tr>
<td></td>
<td>active project.</td>
</tr>
</tbody>
</table>
### IDE Menus

#### Macintosh Menu Layout

Table 31.14  Project Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Make</strong></td>
<td>Compiles and links all marked or modified files in the current build target of the active project; saving the executable file.</td>
</tr>
<tr>
<td><strong>Stop Build</strong></td>
<td>Stops the current compile and link operation and cancels the remainder of the build process.</td>
</tr>
<tr>
<td><strong>Remove Object Code</strong></td>
<td>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 <strong>Remove Object Code &amp; Compact</strong>.</td>
</tr>
<tr>
<td><strong>Re-search for Files</strong></td>
<td>Resets the cached locations of source files using the project access paths, and storing them for faster builds and project operations.</td>
</tr>
<tr>
<td><strong>Reset Project Entry Paths</strong></td>
<td>Resets the location of all source files in the active project using the project access paths.</td>
</tr>
<tr>
<td><strong>Synchronize Modification Dates</strong></td>
<td>Updates the modification dates of all source files in the active project.</td>
</tr>
<tr>
<td><strong>Debug</strong> or <strong>Resume</strong></td>
<td>Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.</td>
</tr>
<tr>
<td><strong>Run</strong></td>
<td>Compiles and links all marked or modified files in the current build target of the active window, then runs the built executable file.</td>
</tr>
<tr>
<td><strong>Set Default Project</strong></td>
<td>Uses the Set Default Project menu to choose the default project when more than one project is open in the IDE.</td>
</tr>
<tr>
<td><strong>Set Default Target</strong></td>
<td>Uses the Set Default Target menu to choose the default build target when more than one build target is present in the project file.</td>
</tr>
</tbody>
</table>
Debug Menu

The Debug menu contains commands for managing program execution. Table 31.15 explains the commands of this menu.

Table 31.15  Debug Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kill</td>
<td>Terminates the current debugging session returning control to the IDE.</td>
</tr>
<tr>
<td>Restart</td>
<td>Terminates the current debugging session, then restarts the program from the beginning.</td>
</tr>
<tr>
<td>Step Over</td>
<td>Executes each source line in the program, treating routine calls as a single statement and stopping the program at the next line of code.</td>
</tr>
<tr>
<td>Step Into</td>
<td>Executes each source line in the program, following any subroutine calls.</td>
</tr>
<tr>
<td>Step Out</td>
<td>Executes each source line in the subroutine and stops the program when the routine returns to its caller.</td>
</tr>
<tr>
<td>Stop</td>
<td>Pauses execution of the program in a debugging session to enable examination of register and variable contents.</td>
</tr>
<tr>
<td>Set Breakpoint</td>
<td>Sets a breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>or Clear Breakpoint</td>
<td>Clears the breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Set Eventpoint</td>
<td>Sets an eventpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Clear Eventpoint</td>
<td>Clears the breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Set/Clear Breakpoint</td>
<td>Displays the Set/Clear Breakpoint dialog for setting or clearing breakpoints by address or symbol.</td>
</tr>
<tr>
<td>Enable Breakpoint</td>
<td>Activates the disabled breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>or Disable Breakpoint</td>
<td>De-activates the breakpoint on the source line containing the insertion point.</td>
</tr>
</tbody>
</table>
### Table 31.15  Debug Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear All Breakpoints</td>
<td>Clears all breakpoints currently set in the default build target of the active project.</td>
</tr>
<tr>
<td>Show Breakpoints or Hide Breakpoints</td>
<td>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.</td>
</tr>
<tr>
<td>Set Watchpoint or Clear Watchpoint</td>
<td>Sets a watchpoint on the source line containing the insertion point. Clears the watchpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Enable Watchpoint or Disable Watchpoint</td>
<td>Activates the disabled watchpoint on the source line containing the insertion point. De-activates the watchpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Clear All Watchpoints</td>
<td>Clears all watchpoints currently set in the default build target of the active project.</td>
</tr>
<tr>
<td>Run to Cursor</td>
<td>Sets a temporary breakpoint on the source line containing the insertion point.</td>
</tr>
<tr>
<td>Change Program Counter</td>
<td>Displays the Change Program Counter dialog where you can move the current statement arrow to an address or symbol.</td>
</tr>
<tr>
<td>Break on C++ Exception</td>
<td>Configures the debugger to break at <code>__throw()</code> each time a C++ exception occurs.</td>
</tr>
<tr>
<td>Break on Java Exceptions</td>
<td>Use this menu to select which Java exceptions the debugger should break on.</td>
</tr>
<tr>
<td>Switch to Monitor</td>
<td>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.</td>
</tr>
</tbody>
</table>
Data Menu

The **Data** menu contains commands that control how the CodeWarrior debugger displays data values. This menu only appears during a debugging session. **Table 31.16** explains the commands of this menu.

**Table 31.16 Data Menu Commands**

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Types</td>
<td>Toggles the appearance of the data type on local and global variables displayed in Variable panes and Variable windows.</td>
</tr>
<tr>
<td>Refresh All Data</td>
<td>Updates data displays.</td>
</tr>
<tr>
<td>New Expression</td>
<td>Creates a new expression entry in the Expressions window.</td>
</tr>
<tr>
<td>Copy to Expression</td>
<td>Copies the selected variable to the Expressions window.</td>
</tr>
<tr>
<td>View As</td>
<td>Displays the View As dialog where the data type of the selected variable can be specified.</td>
</tr>
<tr>
<td>View Variable</td>
<td>Displays the selected variable in a new Variables window.</td>
</tr>
<tr>
<td>View Array</td>
<td>Displays the selected array variable in a new Arrays window.</td>
</tr>
<tr>
<td>View Memory</td>
<td>Displays the selected variable in a new Memory window.</td>
</tr>
<tr>
<td>View Memory As</td>
<td>Displays the View As dialog where the data type of the selected variable can be specified, then shown in a new Memory window.</td>
</tr>
<tr>
<td>Cycle View</td>
<td>Toggles the data view among <strong>View Source</strong>, <strong>View Disassembly</strong>, <strong>View Mixed</strong>, and <strong>View Raw Data</strong>.</td>
</tr>
<tr>
<td>View Source</td>
<td>View data as source code.</td>
</tr>
<tr>
<td>View Disassembly</td>
<td>View data as disassembled.</td>
</tr>
<tr>
<td>View Mixed</td>
<td>View data as source code and disassembled.</td>
</tr>
<tr>
<td>View Raw Data</td>
<td>View data without applied formatting.</td>
</tr>
<tr>
<td>View As Default</td>
<td>Views selected variable in default value format.</td>
</tr>
</tbody>
</table>
IDE Menus

Macintosh Menu Layout

Table 31.16 Data Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View As Binary</td>
<td>Views selected variable as a binary value.</td>
</tr>
<tr>
<td>View As Signed Decimal</td>
<td>Views selected variable as a signed decimal value.</td>
</tr>
<tr>
<td>View As Unsigned Decimal</td>
<td>Views selected variable as an unsigned decimal value.</td>
</tr>
<tr>
<td>View As Hexadecimal</td>
<td>Views selected variable as a hexadecimal value.</td>
</tr>
<tr>
<td>View As Character</td>
<td>Views selected variable as a character value.</td>
</tr>
<tr>
<td>View As C String</td>
<td>Views selected variable as a C string.</td>
</tr>
<tr>
<td>View As Pascal String</td>
<td>Views selected variable as a Pascal string.</td>
</tr>
<tr>
<td>View As Unicode String</td>
<td>Views selected variable as a Unicode string.</td>
</tr>
<tr>
<td>View As Floating Point</td>
<td>Views selected variable as a floating point value.</td>
</tr>
<tr>
<td>View As Enumeration</td>
<td>Views selected variable as an enumerated value.</td>
</tr>
<tr>
<td>View As Fixed</td>
<td>Views selected variable as a 32-bit fixed value.</td>
</tr>
<tr>
<td>View As Fract</td>
<td>Views selected variable as a fract value.</td>
</tr>
</tbody>
</table>

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. This menu’s sections are:

- 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.

Table 31.17 explains the commands of this menu.

The Window menu also lists the names of all open file and project windows. A checkmark appears beside the active window; an underline indicates a modified and unsaved file.
## Table 31.17  Window Menu Commands

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Editor Windows</td>
<td>Arranges all editor windows so that only the title bar is visible.</td>
</tr>
<tr>
<td>Tile Editor Windows</td>
<td>Tiles all editor windows horizontally on the screen so none overlap.</td>
</tr>
<tr>
<td>Tile Editor Windows Vertically</td>
<td>Tiles all editor windows vertically on the screen so none overlap.</td>
</tr>
<tr>
<td>Zoom Window</td>
<td>Restores the active editor window to its previous size and position.</td>
</tr>
<tr>
<td>Collapse Window</td>
<td>Collapses the active editor window so that only its title bar is visible.</td>
</tr>
<tr>
<td>(Minimize Window) or Expand Window</td>
<td>Expands the collapsed editor window to its previous size and position.</td>
</tr>
<tr>
<td>(Maximize Window)</td>
<td></td>
</tr>
<tr>
<td>Save Default Window</td>
<td>Saves the current browser-window settings for later reuse.</td>
</tr>
<tr>
<td>Toolbars</td>
<td>Use the Toolbars submenu to show, hide, reset, and clear window, main, and floating toolbars.</td>
</tr>
<tr>
<td>Browser Contents</td>
<td>Opens or brings to the front a Browser Contents window.</td>
</tr>
<tr>
<td>Class Hierarchy or Class Hierarchy Window</td>
<td>Opens or brings to the front a Class Hierarchy window.</td>
</tr>
<tr>
<td>New Class Browser</td>
<td>Opens or brings to the front a New Class Browser window.</td>
</tr>
<tr>
<td>Build Progress or Build Progress Window</td>
<td>Opens or brings to the front a Build Progress window.</td>
</tr>
<tr>
<td>Errors &amp; Warnings or Errors &amp; Warnings Window</td>
<td>Opens or brings to the front an Errors &amp; Warnings window.</td>
</tr>
<tr>
<td>Project Inspector</td>
<td>Opens or brings to the front a Project Inspector window.</td>
</tr>
</tbody>
</table>
Table 31.17  Window Menu Commands (continued)

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToolServer Worksheet</td>
<td>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.</td>
</tr>
<tr>
<td>Symbolics or Symbolics Window</td>
<td>Opens or brings to the front a Symbolics window.</td>
</tr>
<tr>
<td>Processes or Processes Window</td>
<td>Opens or brings to the front a Processes window.</td>
</tr>
<tr>
<td>Expressions or Expressions Window</td>
<td>Opens or brings to the front an Expressions window.  Use to view, create, modify, and remove expressions.</td>
</tr>
<tr>
<td>Global Variables or Global Variables Window</td>
<td>Opens or brings to the front a Global Variables window.</td>
</tr>
<tr>
<td>Breakpoints or Breakpoints Window</td>
<td>Opens or brings to the front a Breakpoints window.  Use to view, create, modify, and remove breakpoints.</td>
</tr>
<tr>
<td>Registers or Register Window</td>
<td>Opens or brings to the front a Register window.</td>
</tr>
</tbody>
</table>

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.
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 31.18** explains the commands of this menu.

**Table 31.18  Help Menu Commands**

<table>
<thead>
<tr>
<th>Menu command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CodeWarrior Help</strong></td>
<td>Launches a help viewer to display the online help. Click on a link to view a specific IDE topic.</td>
</tr>
<tr>
<td>Index (Windows)</td>
<td>Opens the online help to the Index tab.</td>
</tr>
<tr>
<td>Search (Windows)</td>
<td>Opens the online help to the Search tab.</td>
</tr>
<tr>
<td><strong>Freescale CodeWarrior Website</strong></td>
<td>Launches a browser and automatically points you to the website.</td>
</tr>
<tr>
<td><strong>About Freescale CodeWarrior</strong> (Windows)</td>
<td>Displays the CodeWarrior IDE version and build number information.</td>
</tr>
</tbody>
</table>

**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 Freescale 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 plugins for the CodeWarrior IDE. You can also use this window to enable or disable plugin 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:
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**

Apply Difference

The **Apply Difference** command applies the selected difference from the source file into the destination file.

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**.

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**
- **Uncaught Exceptions Only**.

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.
Menu Commands

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 32.1 explains how to abort the syntax-checking process:

<table>
<thead>
<tr>
<th>On this host…</th>
<th>Do this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Press Esc</td>
</tr>
<tr>
<td>Macintosh</td>
<td>Press Command-. (Command-Period)</td>
</tr>
<tr>
<td>Solaris</td>
<td>Press Esc</td>
</tr>
<tr>
<td>Linux</td>
<td>Press Esc</td>
</tr>
</tbody>
</table>

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.
Menu Commands

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
- View Mixed
- View Raw Data
- View Source
Menu Commands

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.
Menu Commands

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.
Menu Commands

**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.
Menu Commands

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.
Menu Commands

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.

Freescale CodeWarrior Website
The CodeWarrior Website command launches a web browser and displays the CodeWarrior web site.

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.

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.
Menu Commands

---

**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.

---

**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.

---

**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.
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

Minimize Window
Windows equivalent of Collapse Window.
See also: Collapse Window

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.
Menu Commands

**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.
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.

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.
Menu Commands

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.

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.
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**

**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.
Menu Commands

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.
Menu Commands

Resize
The Resize command reveals the Resize submenu.
See also:
- To Largest Height
- To Largest Width
- To Smallest Height
- To Smallest Width

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.
Menu Commands

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.

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.
Menu Commands

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 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**.
Menu Commands

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.
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, as Figure 32.1 shows.

Figure 32.1  Tile Editor Windows Example

![Tile Editor Windows Example](image)

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, as Figure 32.2 shows.
Menu Commands

Figure 32.2  Tile Horizontally Example

```
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
```

**Tile Vertically**

This command resizes open editor windows vertically and arranges them so that none overlap, as Figure 32.3 shows.

Figure 32.3  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
- Hide Window Toolbar
- Reset Window Toolbar
- Clear Window Toolbar
- Show Main Toolbar
- Hide Main Toolbar
- Reset Main Toolbar
- Clear Main Toolbar
- Hide Floating Toolbar
- Show Floating Toolbar
- Reset Floating Toolbar
- Clear Floating Toolbar

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.
Menu Commands

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.

Uncaught Exceptions Only
The Uncaught 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.
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.
Menu Commands

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.
Symbols
#include files, caching 430
#pragma directives, profiler 316
%file command-line string 432
%line command-line string 432
(Scripts) folder 501, 521
.*[._]Data 364
.mcpx 35
\(.\*\) 364
^var 233
__copy_vectors() 318
__PROFILE_ENTRY 308
__PROFILE_EXIT 308
__throw() 505

A
abnormal termination 307
about
breakpoints 203
console applications 77
dockable windows 63
eventpoints 203
Files page in Project window 43
markers 108
projects 29
special breakpoints 203
watchpoints 203
workspaces 73
About Metrowerks CodeWarrior menu command 503
Absolute Path option
in Source Trees preference panel 447
in Type list box 447
abstract, icon for 161
access breakpoint - refer to watchpoints 224
access breakpoint enabled 225
Access Filter display 164
Access Paths settings panel 362, 393
columns
Framework 396
Recursive Search 395
Search Status 395
options
Add 395
Add Default 395
Always Search User Paths 394
Change 395
Host Flags 395
Interpret DOS and Unix Paths 395
Remove 395
Require Framework Style Includes 395
System Paths 446
User Paths 450
Access Target button 278
Action option 335
actions for debugging 192
Activate Browser Coloring option 415
in Text Colors panel 428
Activate Browser option
and relation to Symbolics window 253
in Build Extras panel 514
Activate Syntax Coloring option 415, 420
in Text Colors panel 428, 431, 444
activating automatic code completion 96
Active icon 207
Add button 395
Add Default button 415
Add Files button 127
Add Files menu command 503
Add Window menu command 503
adding
gray background behind IDE. See Use
Multiple Document Interface, turning on.
remote connections 386
source trees 366
Address checkbox 281
Address Line fault 282
Address text box 248
advanced topics
for projects 38
Align submenu 503, 504, 518, 526, 532, 533, 535
Horizontal Center command 518
Left Edges command 518
Vertical Center command 526, 532, 533, 535
All Exceptions command 504
All Info option, in Plugin Diagnostics 433
All Sectors checkbox 271
All Sectors list 271
All Text option button 114, 117, 120
alphabetical sorting of Functions list pop-up 106, 107
Always Search User Paths option 415
Analyzer Can Cause Target Breakpoint checkbox 288
Analyzer Configuration File text box 287
Analyzer Connections target settings panel 285
Analyzer Slot text box 287
Analyzer Type list box 287
Ancestor pop-up 166
Anchor Floating Toolbar command 504
Appears in Menus 335, 336
Appears in Menus checkbox 131
Apple Help Viewer 428
Apple menu 486
Application field 416
applications for the console, about 77
for the console, creating 77
Apply Address Offset checkbox 269
Apply button 140
Apply Difference command 141, 504, 534
Arguments field 416
Arithmetic Optimizations 404
Arm command 289
Array window 242
opening 243
arrays, setting default viewing size for unbounded 422
arrows
  current statement 191
assigning
  Quote Key prefix 351
Attempt To Use Dynamic Type of C++, Object Pascal And SOM Objects option 416
Auto Indent option 416
Auto Repeat 335
Auto Target Libraries option 416
Auto, of Text View list box 251
auto-complete code. See code completion.
Automatic Invocation option 416
Automatically Launch Applications When SYM File Opened option 417
Auto-target Libraries option 416

B
Background option 417
background, desktop
  removing from behind IDE. See Use
    Multiple Document Interface, turning on.
  seeing behind IDE. See Use Multiple
    Document Interface, turning off.
Balance Flash Delay option 417
Editor Settings panel 418
Balance menu command 504
Balance While Typing option 417, 418
balancing punctuation 94
toggling 95
Balloon Help 344, 423, 501
Base Classes field 178
Begin Scope Loop button 280
Begin Test button 282
bestTimeBase 305, 317
BFVALUE 460
Bit Value Modifier list box 249
Bit Value text box 249
BITFIELD 457
Bitfield Description option of Text View pop-up menu 251
Bitfield Name list box 249
Blank Check button 272
Bottom Edges command 504
boxes
  Destination 136
  Pane Collapse 143, 190
  Pane Expand 143, 190
  Source 136
Branch Optimizations 404
Break command 194
Break menu command 505
Break On C++ Exception menu command 505
Break on Java Exceptions command 505
Breakpoint Properties button 206
breakpoint template 212
breakpoint template, defined 204
breakpoint templates
  creating 212
  deleting 213
  specifying the default 214
  working with 212
Breakpoints 204
breakpoints
  Breakpoint Type property 209
  clearing all 210
  Condition property 209
  conditional 204
  default template 212
  defined 203
  disabled 204, 225
  enabled 204
  File-Info property 209
  Hardware property 209
  Hit Count property 209
  Name property 209
  Original Process property 209
  Original-Target property 209
  purpose of 203
  regular 204
  saving sets of 207
  Serial Number property 209
  setting conditional 211
  setting temporary 211
  template 212
  temporary 204
  Thread property 209
  Times Hit property 209
  Times Left property 209
  viewing 207
  working with 207
Breakpoints button 190
Breakpoints column, in editor window 88
Breakpoints menu command 505
Breakpoints window 204
  Active icon 207
  Breakpoint Properties button 206
  Create Breakpoint Template button 205
Groups tab 206
Inactive icon 207
Instances tab 206
opening 207
Rename Breakpoint button 206
saving contents of 207
Set Default Breakpoint Template button 206
Templates tab 206
Breakpoints Window menu command 505
breakpoints, clearing 210
breakpoints, disabling 210
breakpoints, enabling 210, 223
breakpoints, setting 207
breakpoints, viewing properties for 208
Bring To Front menu command 505
Bring Up To Date menu command 51, 52
Bring Up To Date menu command 427, 505
Browse button 121, 248, 250, 264, 268, 276
Browse In Processes Window option 386, 387
browser 150
  Class Browser window 153
  Classes pane 159
  collapsing panes 159
  creating new classes 160, 175, 176
  creating new data members 183
  creating new member functions 180, 181, 183
  expanding panes 158
  hierarchy windows 166
  Member Functions pane 161
  overview 25
  printing class hierarchies 167
  purpose of 147
  setting options 147
  Source pane 163
  status area 163
  viewing data by contents 170
  viewing data by inheritance 166
  working with 147
Browser Access Filters 155
Browser Commands option 418
Editor Settings panel 431
Browser Contents 154
Browser Contents command 505
Browser Contents window 169
Symbols list 170
browser database
defined 147
Browser menu 418
Browser Path option 418
Browser Wizard 175
Build Before Running option 418
Build Extras panel
options
Initial Directory field 430
Use External Debugger 448
Use modification date caching 449
Build Extras settings panel 253, 396
options
Application 398
Arguments 398
Cache Subprojects 397
Dump internal browse information after compile 398
Generate Browser Data From 398
Initial directory 398
Use External Debugger 398
Use modification date caching 397
Build Extras target settings panel 514
Build Progress menu command 505
Build Progress Window menu command 505
Build Settings panel
options
Include file cache 430
Play sound after ‘Bring Up To Date’ & ‘Make’ 356
Save open files before build 356
Show message after building up-to-date project 356
Success 356
Use Local Project Data Storage 357
build system
overview 26
build targets 31
configuring 55
creating 53
management 48
managing 53
moving 50
removing 48, 53
renaming 51, 54
setting default 54
strategies for 40
Bus Noise checkbox 281
Bus Noise test
subtests
Full Range Converging 284
Maximum Invert Convergence 284
Sequential 284
bus noise, defined 284
Button
Choose 341
Delete 342
Export 349
Import 350
New Binding 348
Save 342
buttons
Access Target 278
Add 395
Add Default 415
Add Files 127
Apply 140
Begin Scope Loop 280
Begin Test 282
Blank Check 272
Breakpoint Properties 206
Breakpoints 190

542
<table>
<thead>
<tr>
<th>Command</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse</td>
<td>121, 248, 250, 264, 268, 276</td>
</tr>
<tr>
<td>Calculate Checksum</td>
<td>273</td>
</tr>
<tr>
<td>Cancel</td>
<td>114, 116, 263, 275</td>
</tr>
<tr>
<td>Change</td>
<td>395</td>
</tr>
<tr>
<td>Clear List</td>
<td>127</td>
</tr>
<tr>
<td>Compare</td>
<td>137</td>
</tr>
<tr>
<td>Create Breakpoint Template</td>
<td>205</td>
</tr>
<tr>
<td>Debug</td>
<td>190</td>
</tr>
<tr>
<td>Details</td>
<td>270, 271, 273, 278, 280, 282</td>
</tr>
<tr>
<td>Edit</td>
<td>377</td>
</tr>
<tr>
<td>Erase</td>
<td>272</td>
</tr>
<tr>
<td>Export Panel</td>
<td>424</td>
</tr>
<tr>
<td>Expressions</td>
<td>190</td>
</tr>
<tr>
<td>Factory Settings</td>
<td>427</td>
</tr>
<tr>
<td>Find</td>
<td>114, 116, 119</td>
</tr>
<tr>
<td>Find All</td>
<td>114, 119</td>
</tr>
<tr>
<td>Installed Products</td>
<td>503</td>
</tr>
<tr>
<td>Kill</td>
<td>190</td>
</tr>
<tr>
<td>Line and Column</td>
<td>192</td>
</tr>
<tr>
<td>Load Settings</td>
<td>262, 275</td>
</tr>
<tr>
<td>Next Result</td>
<td>130</td>
</tr>
<tr>
<td>OK</td>
<td>263, 275</td>
</tr>
<tr>
<td>Previous Result</td>
<td>130</td>
</tr>
<tr>
<td>Program</td>
<td>270</td>
</tr>
<tr>
<td>Purge Cache</td>
<td>436</td>
</tr>
<tr>
<td>Read</td>
<td>250</td>
</tr>
<tr>
<td>Redo</td>
<td>141</td>
</tr>
<tr>
<td>Remove</td>
<td>395</td>
</tr>
<tr>
<td>Remove a Set</td>
<td>127</td>
</tr>
<tr>
<td>Rename Breakpoint</td>
<td>206</td>
</tr>
<tr>
<td>Replace</td>
<td>116, 119</td>
</tr>
<tr>
<td>Replace All</td>
<td>116, 119</td>
</tr>
<tr>
<td>Reset Value</td>
<td>250</td>
</tr>
<tr>
<td>resetting in toolbars</td>
<td>346</td>
</tr>
<tr>
<td>Resume</td>
<td>190</td>
</tr>
<tr>
<td>Revert</td>
<td>250</td>
</tr>
<tr>
<td>Run</td>
<td>190</td>
</tr>
<tr>
<td>Save Settings</td>
<td>262, 275</td>
</tr>
<tr>
<td>Save This Set</td>
<td>127</td>
</tr>
<tr>
<td>Set Default Breakpoint Template</td>
<td>206</td>
</tr>
<tr>
<td>Show Log</td>
<td>262, 282</td>
</tr>
<tr>
<td>Source File</td>
<td>191</td>
</tr>
<tr>
<td>Step Into</td>
<td>190</td>
</tr>
<tr>
<td>Step Out</td>
<td>190</td>
</tr>
<tr>
<td>Step Over</td>
<td>190</td>
</tr>
<tr>
<td>Stop</td>
<td>119, 129, 190</td>
</tr>
<tr>
<td>Symbolics</td>
<td>190</td>
</tr>
<tr>
<td>Unapply</td>
<td>140</td>
</tr>
<tr>
<td>Undo</td>
<td>141</td>
</tr>
<tr>
<td>Variables Pane Listing</td>
<td>191</td>
</tr>
<tr>
<td>Verify</td>
<td>270</td>
</tr>
<tr>
<td>Warnings</td>
<td>129</td>
</tr>
<tr>
<td>Write</td>
<td>250</td>
</tr>
<tr>
<td>By Type text/list box</td>
<td>121</td>
</tr>
<tr>
<td>Byte option button</td>
<td>277, 279, 281</td>
</tr>
</tbody>
</table>

**C**

- cache
  - purging 436
- Cache Edited Files Between Debug Sessions option 419
- Cache Subprojects option 419
- Cache Symbolics Between Runs option 419
- Cache window 290
  - opening 290
- caching
  - `#include` files 430
  - precompiled headers 430
- Calculate Checksum button 273
- Can’t Redo menu command 475, 489
- Can’t Undo menu command 475, 489
- Cancel button 114, 116, 263, 275
- Cancel button, in Remove Markers window 109
- Cascade menu command 506
- Case Sensitive checkbox 114, 116, 120, 136
- Case Sensitive option 419
- Change button 395
- Change Program Counter menu command 506
- changing
  - find strings 131
  - line views in a hierarchical window 168
  - register data views 246
  - register values 246
  - remote connections 387
  - source trees 366
- Check Syntax command 506
- Checkbox
  - Numeric Keypad Bindings 348
checkboxes

Address 281
All Sectors 271
Analyzer Can Cause Target Breakpoint 288
Appears in Menus 131
Apply Address Offset 269
Bus Noise 281
Case sensitive 114, 116, 120, 136
Compare text file contents 137
Enable Logging 265
Erase Sectors Individually 271
Ignore extra space 137
Log Message 216
Match whole word 114, 116, 120
Only show different files 137
Project headers 123
Project sources 123
Regular expression 114, 117, 120
Restrict Address Range 269
Search cached sub-targets 123
Search selection only 114, 117
Search sub-folders 121
Search up 114, 117
Speak Message 216
Stop at end of file 114, 117
Stop in Debugger 217, 219, 220
System headers 123
Target Breakpoint Can Cause Analyzer Trigger checkbox 288
Treat as Expression 217
Use Custom Settings 264, 276
Use Selected File 268
Use Target CPU 282
Use Target Initialization 264, 276
View Target Memory Writes 265
Walking 1’s 281

Checkout Status column in Files view of Project window 44
Checksum panel 272
child windows, defined 63
choosing
  a default project 37
  linkers 329
  one character from many in regular expressions 134

class browser
  purpose of windows 153
  working with windows 153
Class Browser menu command 506
Class Browser window 153
  Classes pane 155
  Data Members pane 155
  Member Functions pane 155
  Status area 155

class data
  viewing from hierarchy windows 158
Class Declaration 164
Class Hierarchy 154
Class Hierarchy menu command 507
Class Hierarchy Window menu command 507
class hierarchy windows
  purpose of 165
  working with 165
class view 312
classes
  creating 160, 175, 176
  hiding pane for 160
  showing pane for 160
  sorting list of 161
Classes option 377
Classes pane 159
  in Class Browser window 155
classes.zip 504
Clear All Breakpoints menu command 507
Clear All Watchpoints menu command 507
Clear Breakpoint menu command 507
Clear Eventpoint menu command 507
Clear Floating Toolbar command in Toolbar submenu 507
Clear List button 127
Clear Main Toolbar menu command 507
Clear menu command 507
Clear Watchpoint menu command 508
Clear Window Toolbar command in Toolbar submenu 508
clearing
  all breakpoints 210
all watchpoints 227
breakpoints 210
watchpoints 227
client area, defined 63
Clone Existing Target option 53
Close All command 61
Close All Editor Documents menu command 508
Close All menu command 508
Close Catalog menu command 508
Close command 38, 60
Close menu command 508
Close Non-debugging Windows option 420
Close Workspace menu command 508
closing
all files 61
dockable windows 71
files 60
projects 38
workspaces 75
Code 405
code
adding markers to 109
completing 96
disabling breakpoints 210
disabling eventpoints 222
disabling special breakpoints 228
disabling watchpoints 226
enabling breakpoints 210, 223
enabling special breakpoints 228
enabling watchpoints 226
locating 105
navigating 105
setting breakpoints in 207
setting watchpoints in 225
viewing breakpoint properties 208
viewing eventpoint properties 222
viewing watchpoint properties 226
Code column
in Files view of Project window 44
code completion
activating automatic behavior 96
configuration 96
deactivating automatic behavior 97
for parameter lists 102
navigating window 100
selecting items 101
triggering by keyboard 97
triggering from IDE menu bar 97
Code Completion Delay option 420
Code Completion preference panel 367
options
Automatic Invocation 368
Case sensitive 368
Code Completion Delay 368
Display deprecated items 368
Window follows insertion point 368
Code Completion window 98
code completion, triggering from IDE menu bar 97
Code Formatting preference panel 369
options
Close Braces, Brackets, And Parentheses 371
Format Braces 370
Indent Braces 370
Indent Case Within Switch Statement 371
Indent Code Within Braces 370
Language Settings 369
Place Else On Same Line As Closing Brace 370
Place Opening Brace On Separate Line 370
Use Automatic Code Formatting 369
Code Only option button 115, 117, 120
CodeWarrior
menu reference 473
overview 21
CodeWarrior Glossary command 509
CodeWarrior Help menu command 509
CodeWarrior IDE
Apple menu 486
CodeWarrior menu 487
Data menu 483, 497
Debug menu 482, 495
Edit menu 475, 489
File menu 473, 487
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal Line</td>
<td>168</td>
</tr>
<tr>
<td>Disable Breakpoint</td>
<td>511</td>
</tr>
<tr>
<td>Disable Watchpoint</td>
<td>511</td>
</tr>
<tr>
<td>Disarm</td>
<td>289</td>
</tr>
<tr>
<td>Disassemble</td>
<td>511</td>
</tr>
<tr>
<td>Disconnect</td>
<td>289</td>
</tr>
<tr>
<td>Display Grid</td>
<td>511</td>
</tr>
<tr>
<td>Enable Breakpoint</td>
<td>512</td>
</tr>
<tr>
<td>Enable Watchpoint</td>
<td>512</td>
</tr>
<tr>
<td>Enter Find String</td>
<td>131, 512</td>
</tr>
<tr>
<td>Enter Replace String</td>
<td>512</td>
</tr>
<tr>
<td>Errors And Warnings</td>
<td>513</td>
</tr>
<tr>
<td>Errors And Warnings Window</td>
<td>513</td>
</tr>
<tr>
<td>Exit</td>
<td>513</td>
</tr>
<tr>
<td>Expand Window</td>
<td>513</td>
</tr>
<tr>
<td>Export Project</td>
<td>37, 513, 518</td>
</tr>
<tr>
<td>Export Project as GNU Makefile</td>
<td>513</td>
</tr>
<tr>
<td>Expressions</td>
<td>513</td>
</tr>
<tr>
<td>Expressions Window</td>
<td>513</td>
</tr>
<tr>
<td>File Path</td>
<td>45</td>
</tr>
<tr>
<td>Find</td>
<td>115, 514</td>
</tr>
<tr>
<td>Find and Open 'Filename'</td>
<td>515</td>
</tr>
<tr>
<td>Find and Open File</td>
<td>515</td>
</tr>
<tr>
<td>Find And Replace</td>
<td>516</td>
</tr>
<tr>
<td>Find Definition</td>
<td>514</td>
</tr>
<tr>
<td>Find Definition &amp; Reference</td>
<td>111, 514</td>
</tr>
<tr>
<td>Find In Files</td>
<td>514</td>
</tr>
<tr>
<td>Find In Next File</td>
<td>515</td>
</tr>
<tr>
<td>Find In Previous File</td>
<td>515</td>
</tr>
<tr>
<td>Find Next</td>
<td>130, 515</td>
</tr>
<tr>
<td>Find Previous</td>
<td>131, 515</td>
</tr>
<tr>
<td>Find Previous Selection</td>
<td>515</td>
</tr>
<tr>
<td>Find Reference</td>
<td>111, 516</td>
</tr>
<tr>
<td>Find Selection</td>
<td>132, 516</td>
</tr>
<tr>
<td>Get Next Completion</td>
<td>516</td>
</tr>
<tr>
<td>Get Previous Completion</td>
<td>516</td>
</tr>
<tr>
<td>Global Variables</td>
<td>517</td>
</tr>
<tr>
<td>Global Variables Window</td>
<td>517</td>
</tr>
<tr>
<td>Go Back</td>
<td>154, 517</td>
</tr>
<tr>
<td>Go Forward</td>
<td>154, 517</td>
</tr>
<tr>
<td>Go To Line</td>
<td>517</td>
</tr>
<tr>
<td>Hide Breakpoints</td>
<td>517</td>
</tr>
<tr>
<td>Hide Classes</td>
<td>160</td>
</tr>
<tr>
<td>Hide Classes pane</td>
<td>163</td>
</tr>
<tr>
<td>Hide Window Toolbar</td>
<td>518</td>
</tr>
<tr>
<td>Import Components</td>
<td>518</td>
</tr>
<tr>
<td>Import Project</td>
<td>37, 518</td>
</tr>
<tr>
<td>Insert Reference Template</td>
<td>518</td>
</tr>
<tr>
<td>Kill</td>
<td>194, 518</td>
</tr>
<tr>
<td>Make</td>
<td>519</td>
</tr>
<tr>
<td>Maximize Window</td>
<td>519</td>
</tr>
<tr>
<td>Metrowerks Website</td>
<td>516</td>
</tr>
<tr>
<td>Minimize Window</td>
<td>519</td>
</tr>
<tr>
<td>New</td>
<td>519</td>
</tr>
<tr>
<td>New Class</td>
<td>519</td>
</tr>
<tr>
<td>New Class Browser</td>
<td>519</td>
</tr>
<tr>
<td>New Data</td>
<td>519</td>
</tr>
<tr>
<td>New Event</td>
<td>520</td>
</tr>
<tr>
<td>New Event Set</td>
<td>520</td>
</tr>
<tr>
<td>New Expression</td>
<td>520</td>
</tr>
<tr>
<td>New Item</td>
<td>160</td>
</tr>
<tr>
<td>New Member Function</td>
<td>520</td>
</tr>
<tr>
<td>New Method</td>
<td>520</td>
</tr>
<tr>
<td>New Property</td>
<td>520</td>
</tr>
<tr>
<td>New Text File</td>
<td>520</td>
</tr>
<tr>
<td>Open</td>
<td>521</td>
</tr>
<tr>
<td>Open File</td>
<td>163</td>
</tr>
<tr>
<td>Open In Windows Explorer</td>
<td>45</td>
</tr>
<tr>
<td>Open Recent</td>
<td>521</td>
</tr>
<tr>
<td>Open Scripts Folder</td>
<td>521</td>
</tr>
<tr>
<td>Open Workspace</td>
<td>521</td>
</tr>
<tr>
<td>Page Setup</td>
<td>521</td>
</tr>
<tr>
<td>Pane Collapse</td>
<td>159</td>
</tr>
<tr>
<td>Pane Expand</td>
<td>158</td>
</tr>
<tr>
<td>Precompile</td>
<td>522</td>
</tr>
<tr>
<td>Preferences</td>
<td>522</td>
</tr>
<tr>
<td>Print</td>
<td>522</td>
</tr>
<tr>
<td>Processes</td>
<td>522</td>
</tr>
<tr>
<td>Processes Window</td>
<td>522</td>
</tr>
<tr>
<td>Project Inspector</td>
<td>36</td>
</tr>
<tr>
<td>Redo</td>
<td>522</td>
</tr>
<tr>
<td>Refresh All Data</td>
<td>523</td>
</tr>
<tr>
<td>Register Details Window</td>
<td>247, 523</td>
</tr>
<tr>
<td>Register Windows</td>
<td>523</td>
</tr>
<tr>
<td>Registers</td>
<td>523</td>
</tr>
<tr>
<td>Remove Object Code</td>
<td>523</td>
</tr>
<tr>
<td>Remove Object Code &amp; Compact</td>
<td>523</td>
</tr>
<tr>
<td>Remove Toolbar Item</td>
<td>345</td>
</tr>
</tbody>
</table>
Replace 117, 524
Replace All 524
Replace and Find Next 524
Restart 195
Resume 194, 526
Revert 526
Run 195, 436, 526
Run To Cursor 527
Save Default Window 527
Save Workspace 527
Save Workspace As 528
Select All 528
Send To Back 528
Set Breakpoint 528
Set Default Project 37, 528
Set Default Target 528
Set Eventpoint 528
Set Watchpoint 529
Shift Right 529
Show Breakpoints 507, 529
Show Classes 160
Show Classes pane 163
Show Inherited 155
Show private 156
Show protected 156
Show public 156
Show Types 529
Show Window Toolbar 518
Single Class Hierarchy Window 154
Sort Alphabetical 160, 161
Sort Hierarchical 160
Stack Editor Windows 530
Step Into 193
Step Out 193
Step Over 193, 530
Stop 194
Stop Build 530
Straight Line 168
Switch To Monitor 530
Symbolics 530
Symbolics Window 530
Synchronize Modification Dates 530
Unapply Difference 141
Update Data 289

View Array 535
View as implementor 156
View as subclass 156
View As Unsigned Decimal 535, 536
View as user 156
View Disassembly 537
View Mixed 537
View Source 537
View Variable 537
Zoom Window 537

Commands & Key Bindings menu command 508
Commands tab 333, 335, 347
Commands&KeyBindings.mkb file 349, 350
Comments Only option button 115, 117, 120
Comments option 420
common debugging actions 192
Common Subexpression Elimination 404
Compare button 137
Compare Files command 137
Compare Files menu command 509
Compare Files Setup window 136
Case Sensitive checkbox 136
Compare button 137
Compare Text File Contents checkbox 137
Destination box 136
Ignore Extra Space checkbox 137
Only Show Different Files checkbox 137
Source box 136
Compare Text File Contents checkbox 137
comparing files
differences, applying 141
differences, unapplying 141
overview 135
setup 136, 137
comparing files, explained 139
comparing folders
examining results 144
overview 135
setup 136, 138
comparing folders, explained 142
computation
destination item 136
source item 136
Compile menu command 509
compiler
  avoiding crashes 421
compiler directives 304, 316
Compiler option 420
Compiler option, in Generate Browser Data From menu 429
compiler thread stack
  and avoiding compiler crashes 421
Compiler Thread Stack field 420
Complete Code menu command 509
completing code 96
Component Object Model. See COM.
Concurrent Compiles panel
  options
    Use Concurrent Compiles 437, 447
    User Specified 450
Concurrent Compiles preference panel 357
  options
    Recommended 358
    Use Concurrent Compiles 357
    User Specified 358
condition, breakpoint property 209
conditional access breakpoint 227
conditional breakpoint, defined 211
conditional breakpoints 204
setting 211
conditional eventpoint, defined 223
conditional eventpoints
setting 223
conditional watchpoint, defined 227
conditional watchpoints
setting 227
Configuration panel 275
configuring
  build targets 55
  code completion 96
  projects for a logic analyzer 285
  targets 55
Confirm “Kill Process” When Closing Or Quitting option 421
Confirm Invalid File Modification Dates When Debugging option 421
Connect command 288
Connect menu command 509
Connection list box 264, 276
Connection pop-up menu, in Remote Debugging settings panel 412
Connection Type list box 287
Connection Type option 386, 387
console applications
  creating 77
  console applications, about 77
constant
  adding to a variable 235
Constants option 377
contents
  of register 247
Context Popup Delay option 421
contextual menus 196
  File Path command 45
  Open In Windows Explorer command 45
  using 197
  using to dock a window 66
controlling program execution 187
conventions
  figures 19
  for manual 19
  keyboard shortcuts 20
Copy And Expression Propagation 404
Copy menu command 509
Copy Propagation 404
Copy To Expression command 510
cores, debugging multiple 197
Create Breakpoint Template button 205
Create Design menu command 510
Create Group menu command 510
Create Target command 53
Create Target menu command 510
creating
  a new data member 162
  build targets 53
  console application 78
  console applications 77
  custom project stationery 38
empty projects 34
files (Macintosh) 58
files (Windows) 57
member functions 162
new classes 160, 175, 176
new data member 183
new data members 183
new member function 180
new member functions 181
projects from makefiles 33
projects using stationery 33
subprojects 39
targets 53
cross-platform migration, and opening projects 35
Current Target list pop-up 42
Current Target menu 345
current-statement arrow 191
custom project stationery 38
Customize IDE Commands window 131, 333, 347, 348
Action 335
Appears in Menus 335, 336
Appears in Menus checkbox 131
Auto Repeat 335
Key Bindings 335
Name field 335
New Binding 335
New Group 336
Numeric Keypad Bindings checkbox 351
Cut command 510
CVS 364
Cycle View menu command 510

D
dash 191
data
finding problems 313
sorting 310
viewing 309
Data column
in Files view of Project window 44
data columns
contents 310
Data Line fault 283
data members
completing code 101
creating 162, 183
identifier icons 161
Data Members pane 162
in Class Browser window 155
Data menu 483, 497
data, for debugger, working with 253
database
navigation for browser 150
deactivating automatic code completion 97
Dead Code Elimination 404
Dead Store Elimination 404
Debug button 190
Debug column
in Files view of Project window 44
Debug command 51, 52, 192
Debug menu 421, 482, 495
Clear All Breakpoints command 483, 496
Disable Watchpoint command 483, 496
Enable Breakpoint command 483, 495
Enable Watchpoint command 483, 495, 496
Hide Breakpoints command 483, 496
Debug menu command 511
debugger 436
attaching to a process 258
choosing for an opened symbolics file 386
overview 26
restarting 195
starting 192
working with data 253
working with memory 237
working with variables 229
Debugger Commands option 421
Debugger list box 287
Debugger section, of IDE preference panels 379
Debugger Settings panel 258, 410
options
Auto-target Libraries 411
Cache symbolics between runs 411
Default language entry point 411, 422
Location of Relocated Libraries and Code Resources 410, 433
source code 91
symbols, shortcuts for 94
director 83
overview 25
third-party support 450
Editor section, of IDE preference panels 367
Editor Settings panel
options
Balance Flash Delay 418
Browser Commands 431
Font Preferences 428
Insert Template Commands 431
Left margin click selects line 432
Project Commands 436
Relaxed C popup parsing 437
Selection position 439
Sort function popup 445
Use multiple undo 449
VCS Commands 451
Window position and size 451
Editor Settings preference panel 371
options
Balance Flash Delay 373
Balance while typing 373
Browser Commands 372
Debugger Commands 373
Default file format 373
Drag and drop editing 373
Edit Commands 372
Enable Virtual Space 373
Font preferences 372
Insert Template Commands 372
Left margin click selects line 373
Project Commands 373
Relaxed C popup parsing 373
Selection position 372
Sort function popup 373
Use multiple undo 373
VCS Commands 373
Window position and size 372
editor toolbar 86
editor window 83
adding panes to 89
Breakpoints column 88
collapsing toolbar in 86
expanding toolbar in 86
line and column indicator 89
pane splitter controls 89
removing panes from 89
resizing panes 89
text editing area 88
editor windows
other 88
selecting text in 91
Emacs text editor 432
empty projects
creating 34
Empty Target option 53
Enable Automatic Toolbar Help option 423
Enable Breakpoint menu command 512
Enable Browser option 505
Enable Logging checkbox 265
Enable Remote Debugging option 424
Enable Virtual Space option 424
Enable Watchpoint menu command 512
enabled breakpoint 204
enabled eventpoint 215
enabled watchpoint 225
enabling
plug-in diagnostics 503
End text box 269, 281
end-of-line format 421
enlarging panes, in browser 158
Enter Find String command 131
Enter Find String menu command 512
Enter Replace String menu command 512
Entire Flash option button 273
Enums option 377
Environment Settings option 424
Environment Variable option
of Source Trees preference panel 447
Environment Variable option, in Type pop-up menu 447
environment variables
Macintosh limitations 447
EOL format 421
Erase / Blank Check panel 270
Erase button 272
Erase Sectors Individually checkbox 271
Errors And Warnings menu command 513
Errors And Warnings Window menu command 513
Errors Only option of Plugin Diagnostics 433
eventpoints
defined 214
disabled 215
enabled 215
Log Point 215, 216
Log Point, clearing 217
Log Point, setting 216
Pause Point 215, 217
Pause Point, clearing 218
Pause Point, setting 217
purpose of 203
Script Point 215, 218
Script Point, clearing 219
Script Point, setting 218
setting conditional 223
Skip Point 215, 219
Skip Point, clearing 219
Skip Point, setting 219
Sound Point 215, 220
Sound Point, clearing 221
Sound Point, setting 220
Sound Point, Speak Message 220
Trace Collection Off 215, 221
Trace Collection Off, clearing 221
Trace Collection Off, setting 221
Trace Collection On 215, 221
Trace Collection On, clearing 222
Trace Collection On, setting 222
working with 222
eventpoints, disabling 222
eventpoints, viewing properties for 222
examining debugger data 253
examining memory 237
examining variables 229
exceptions 307
Exceptions In Targeted Classes command in Java
Exceptions submenu 513
executable files
adding to the Other Executables list 408
changing in the Other Executables list 409
removing from the Other Executables list 409
execution
of program, controlling 187
execution, killing 194
execution, resuming 194
execution, stopping 194
Exit menu command 513
exit() 307
Expand Window menu command 513
expanding
browser panes 158
dockable windows 71
Export 349
Export Panel button 334, 354, 424
Export Project as GNU Makefile menu command 513
Export Project command 37
Export Project menu command 513, 518
exporting
projects to XML files 37
Expression Simplification 404
Expressions button 190
Expressions menu command 513
Expressions window 233
adding expressions 234
opening 234
Expressions Window menu command 513
Extension field 424
external editor
using on the Macintosh 360
external editor support 450
F
Factory Settings button 427
Failure option 427
FDI 359, 449
and dockable windows 63
fields
Application 416
Arguments 416
Base Classes 178
files, tasks for managing 57
Find
  by text selection 130
  single-file 113
Find All button 114, 119
Find and compare operations option
  Shielded Folders panel 427
Find And Open ‘Filename’ menu command 515
Find and Open File command 515
Find and Replace
  multiple-file 118
  single-file 115
Find And Replace menu command 516
Find and Replace window
  All Text option button 117
  Cancel button 116
  Case Sensitive checkbox 116
  Code Only option button 117
  Comments Only option button 117
  Find button 116
  Find text/list box 116
  Match Whole Word checkbox 116
  Regular Expression checkbox 117
  Replace All button 116
  Replace button 116
  Replace With text/list box 116
  Search Selection Only checkbox 117
  Search Up checkbox 117
  Stop At End Of File checkbox 117
Find button 114, 116, 119
Find command 115, 514
Find Definition & Reference command 111
Find Definition & Reference menu command 514
Find Definition menu command 514
Find In Files menu command 514
Find in Files window

IDE 5.7 User’s Guide
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>using 130</td>
<td></td>
</tr>
<tr>
<td>Find Next command 130</td>
<td></td>
</tr>
<tr>
<td>Find Next menu command 515</td>
<td></td>
</tr>
<tr>
<td>Find Previous</td>
<td></td>
</tr>
<tr>
<td>using 131</td>
<td></td>
</tr>
<tr>
<td>Find Previous command 131</td>
<td></td>
</tr>
<tr>
<td>enabling in the Customize IDE Commands window 131</td>
<td></td>
</tr>
<tr>
<td>Find Previous menu command 515</td>
<td></td>
</tr>
<tr>
<td>Find Previous Selection menu command 515</td>
<td></td>
</tr>
<tr>
<td>Find Reference command 111</td>
<td></td>
</tr>
<tr>
<td>Find Reference menu command 516</td>
<td></td>
</tr>
<tr>
<td>Find Reference using option</td>
<td></td>
</tr>
<tr>
<td>IDE Extras panel 427</td>
<td></td>
</tr>
<tr>
<td>Find Selection command 132</td>
<td></td>
</tr>
<tr>
<td>Find Selection menu command 516</td>
<td></td>
</tr>
<tr>
<td>Find symbols with prefix 94</td>
<td></td>
</tr>
<tr>
<td>Find symbols with substring 94</td>
<td></td>
</tr>
<tr>
<td>Find text/list box 114, 116, 119</td>
<td></td>
</tr>
<tr>
<td>Find window</td>
<td></td>
</tr>
<tr>
<td>All Text option button 114</td>
<td></td>
</tr>
<tr>
<td>Cancel button 114</td>
<td></td>
</tr>
<tr>
<td>Case Sensitive checkbox 114</td>
<td></td>
</tr>
<tr>
<td>Code Only option button 115</td>
<td></td>
</tr>
<tr>
<td>Comments Only option button 115</td>
<td></td>
</tr>
<tr>
<td>Find All button 114</td>
<td></td>
</tr>
<tr>
<td>Find button 114</td>
<td></td>
</tr>
<tr>
<td>Find text/list box 114</td>
<td></td>
</tr>
<tr>
<td>Match Whole Word checkbox 114</td>
<td></td>
</tr>
<tr>
<td>Regular Expression checkbox 114</td>
<td></td>
</tr>
<tr>
<td>Search Selection Only checkbox 114</td>
<td></td>
</tr>
<tr>
<td>Search Up checkbox 114</td>
<td></td>
</tr>
<tr>
<td>Stop At End Of File checkbox 114</td>
<td></td>
</tr>
<tr>
<td>finding problems 313</td>
<td></td>
</tr>
<tr>
<td>finding text</td>
<td></td>
</tr>
<tr>
<td>overview 113</td>
<td></td>
</tr>
<tr>
<td>Flags pop-up menu 401</td>
<td></td>
</tr>
<tr>
<td>Ignored By Make flag 401</td>
<td></td>
</tr>
<tr>
<td>Launcheable flag 401</td>
<td></td>
</tr>
<tr>
<td>Precompiled File flag 401</td>
<td></td>
</tr>
<tr>
<td>Resource File flag 401</td>
<td></td>
</tr>
<tr>
<td>Flash Base + Offset 269</td>
<td></td>
</tr>
<tr>
<td>Flash Base Address 269</td>
<td></td>
</tr>
<tr>
<td>Flash Configuration panel</td>
<td></td>
</tr>
<tr>
<td>Flash Memory Base Address text box 266</td>
<td></td>
</tr>
<tr>
<td>Flash Programmer pane 262</td>
<td></td>
</tr>
<tr>
<td>flash programmer panels</td>
<td></td>
</tr>
<tr>
<td>Checksum 272</td>
<td></td>
</tr>
<tr>
<td>Erase / Blank Check 270</td>
<td></td>
</tr>
<tr>
<td>Flash Configuration 265</td>
<td></td>
</tr>
<tr>
<td>Program / Verify 267</td>
<td></td>
</tr>
<tr>
<td>Target Configuration 263</td>
<td></td>
</tr>
<tr>
<td>Flash Programmer window 261</td>
<td></td>
</tr>
<tr>
<td>Cancel button 263</td>
<td></td>
</tr>
<tr>
<td>Checksum panel</td>
<td></td>
</tr>
<tr>
<td>Calculate Checksum button 273</td>
<td></td>
</tr>
<tr>
<td>Details button 273</td>
<td></td>
</tr>
<tr>
<td>Entire Flash option button 273</td>
<td></td>
</tr>
<tr>
<td>File On Host option button 273</td>
<td></td>
</tr>
<tr>
<td>File On Target option button 273</td>
<td></td>
</tr>
<tr>
<td>Memory Range On Target option button 273</td>
<td></td>
</tr>
<tr>
<td>Size text box 273</td>
<td></td>
</tr>
<tr>
<td>Start text box 273</td>
<td></td>
</tr>
<tr>
<td>Status 273</td>
<td></td>
</tr>
<tr>
<td>Erase / Blank Check panel</td>
<td></td>
</tr>
<tr>
<td>All Sectors checkbox 271</td>
<td></td>
</tr>
<tr>
<td>All Sectors list 271</td>
<td></td>
</tr>
<tr>
<td>Blank Check button 272</td>
<td></td>
</tr>
<tr>
<td>Details button 271</td>
<td></td>
</tr>
<tr>
<td>Erase button 272</td>
<td></td>
</tr>
<tr>
<td>Erase Sectors individually</td>
<td></td>
</tr>
<tr>
<td>checkbox 271</td>
<td></td>
</tr>
<tr>
<td>Status 271</td>
<td></td>
</tr>
<tr>
<td>Flash Configuration panel</td>
<td></td>
</tr>
<tr>
<td>Device pane 266</td>
<td></td>
</tr>
<tr>
<td>Flash Memory Base Address text box 266</td>
<td></td>
</tr>
<tr>
<td>Organization pane 267</td>
<td></td>
</tr>
<tr>
<td>Sector Address Map pane 267</td>
<td></td>
</tr>
<tr>
<td>Flash Programmer pane 262</td>
<td></td>
</tr>
<tr>
<td>Load Settings button 262</td>
<td></td>
</tr>
<tr>
<td>OK button 263</td>
<td></td>
</tr>
<tr>
<td>opening 261</td>
<td></td>
</tr>
<tr>
<td>Program / Verify panel</td>
<td></td>
</tr>
<tr>
<td>Apply Address Offset checkbox 269</td>
<td></td>
</tr>
<tr>
<td>Browse button 268</td>
<td></td>
</tr>
<tr>
<td>Details button 270</td>
<td></td>
</tr>
</tbody>
</table>
End text box 269
Flash Base + Offset 269
Flash Base Address 269
Offset text box 269
Program button 270
Restrict Address Range checkbox 269
Start text box 269
Status 270
Use Selected File checkbox 268
Use Selected File text box 268
Verify button 270
Save Settings button 262
Show Log button 262
Target Configuration panel
   Browse button 264
   Connection list box 264
   Default Project 264
   Default Target 264
   Enable Logging checkbox 265
   Target Memory Buffer Address text box 264
   Target Memory Buffer Size text box 265
   Target Processor text/list box 264
   Use Custom Settings checkbox 264
   Use Target Initialization checkbox 264
   Use Target Initialization text box 264
   View Target Memory Writes checkbox 265
flat view 311
floating a window 69
Floating Document Interface. See FDI.
floating window type 64
focus bar 50
Folder Compare Results window 142
   Files In Both Folders pane 143
   Files Only In Destination pane 144
   Files Only In Source pane 144
   Pane Collapse box 143
   Pane Expand box 143
   pane resize bar 143
   Selected Item group 144
folders
   comparing 142

Registers 247
searching (multiple) 121
Font & Tabs panel 375
   options
      Font 428
      Scripts 439
      Size 443
      Tab indents selection 446
      Tab Inserts Spaces 446
      Tab Size 446
Font & Tabs preference panel 373, 376
   options
      Auto Indent 375
      Font 374
      Script 374
      Size 374
      Tab indents selection 374
      Tab Inserts Spaces 375
      Tab Size 374
Font option
Font & Tabs panel 428
Font Preferences option
Editor Settings panel 428
Font Settings 375
Foreground option
Text Colors panel 428
Format list box 248
format, for end of line (EOL) 421
formats
   for documentation 18
FPU Registers 245
Framework column, in Access Paths panel 396
Full Range Converging subtest 284
function
   New Data Member 162
function-level profiling 301, 316
functions
   creating new member 162
   locating 105, 106
Functions list box 192
Functions list pop-up 86
   sorting alphabetically 106, 107
   using 106
Functions option 377
G
General Registers 245
General section, of IDE preference panels 355
Generate Browser Data From option 429
Compiler 429
Language Parser 429
Language Parser, Macro file 429
Language Parser, Prefix file 429
None 429
Generate Constructor and Destructor 178
Get Next Completion menu command 516
Get next symbol 94
Get Previous Completion menu command 516
Get previous symbol 94
Global Optimizations settings panel 402
options
Details 403
Faster Execution Speed 403
Optimization Level slider 403
Smaller Code Size 403
Global Register Allocation 404
Global Register Allocation Only For Temporary Values 404
Global Settings panel
options
Maintain Files in Cache 433
Select stack crawl window when task is stopped 439
Global Settings preference panel
options
Auto Target Libraries 385
Automatically launch applications when SYM file opened 384
Cache Edited Files Between Debug Sessions 384
Confirm "Kill Process" when closing or quitting 384
Confirm invalid file modification dates when debugging 384
Don’t step into runtime support code 385
Maintain files in cache 384
Purge Cache 384
Select stack crawl window when task is stopped 384
Global Variables menu command 517
Global Variables window 229
opening 230
viewing for different processes 230
Global Variables Window menu command 517
Globals option 377
Go Back 154
Go Back menu command 517
Go Forward 154
Go Forward menu command 517
Go To Line menu command 517
going back 107
going forward 107
going to a particular line 107
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 429
Grid Size Y option
Layout Editor panel 429
group management 48
grouping
regular expressions 134
groups
moving 50
removing 48
renaming 50
Selected Item 144
touching 51
touching all 51
untouching 52
untouching all 52
Groups tab 206
H
hardware diagnostic panels
Configuration 275
Memory Read / Write 276
Memory Tests 280
Address 283
Bus Noise 284
Bus Noise in address lines 284
Bus Noise in data lines 285
Walking Ones 282
Scope Loop 278
Hardware Diagnostics pane 275
Hardware Diagnostics window 274
Cancel button 275
Configuration panel
Browse button 276
Connection list box 276
Default Project 276
Default Target 276
Target Processor text/list box 276
Use Custom Settings checkbox 276
Use Target Initialization checkbox 276
Use Target Initialization text box 276
Hardware Diagnostics pane 275
Load Settings button 275
Memory Read / Write panel
Access Target button 278
Byte option button 277
Details button 278
Long Word option button 277
Read option button 277
Status 278
Target Address text box 277
Value to write text box 278
Word option button 277
Write option button 277
Memory Tests panel
Address checkbox 281
Begin Test button 282
Bus Noise checkbox 281
Byte option button 281
Details button 282
End text box 281
Long Word option button 282
Passes text box 282
Show Log button 282
Start text box 281
Status 282
Target Scratch Memory End text box 282
Target Scratch Memory Start text box 282
Use Target CPU checkbox 282
Walking 1’s checkbox 281
Word option button 282
OK button 275
opening 274
Save Settings button 275
Scope Loop panel
Begin Scope Loop button 280
Byte option button 279
Details button 280
Long Word option button 279
Read option button 279
Speed slider 280
Status 280
Target Address text box 279
Value to write text box 279
Word option button 279
Write option button 279
hardware tools, working with 261
hardware, breakpoint property 209
headers

caching precompiled headers 430
Help menu 486, 501
Help Preferences panel 361
options
Browser Path 361
Set 361
Hide Breakpoints menu command 517
Hide Classes 160
Hide Classes pane 163
Hide Floating Toolbar command 517
Hide Main Toolbar command in Toolbar submenu 517
Hide non-debugging windows option
Windowing panel 429
Hide Window Toolbar command 518
hiding
classes pane 160
Hierarchy Control 166
hierarchy window 166
hierarchy windows
changing line views 168
using to view class data 158
hit count, breakpoint property 209
Horizontal Center command 518
Host Application for Libraries & Code Resources option
Runtime Settings panel 430
Host Application For Libraries And Code Resources field
of Runtime Settings panel 430
Host Name text box 287
host-specific registers 245
how to
activate automatic code completion 96
add a constant to a variable 235
add a keyword to a keyword set 406
add an executable file 408
add expressions (Expressions window) 234
add markers to a source file 109
add panes to an editor window 89
add remote connections 386
add source trees 366
adding subprojects to a project 39
alphabetize Functions list pop-up order 106, 107
apply file differences 141
arm a logic analyzer 289
attach the debugger to a process 258
balance punctuation 94
change an executable file 409
change line views in a hierarchical window 168
change register data views 246
change register values 246
change remote connections 387
change source trees 366
change the find string 131
choose a default project 37
choose files to compare 137
choose folders to compare 138
clear a breakpoint 210
clear a Log Point 217
clear a Pause Point 218
clear a Script Point 219
clear a Skip Point 219
clear a Sound Point 221
clear a Trace Collection Off eventpoint 221
clear a Trace Collection On eventpoint 222
clear a watchpoint 227
clear all breakpoints 210
clear all watchpoints 227
close a docked window 71
close a workspace 75
close projects 38
collapse a docked window 71
collapse browser panes 159
collapse the editor window toolbar 86
complete code for data members 101
complete code for parameter lists 102
connect to a logic analyzer 289
create a breakpoint template 212
create a console application 78
create a new class 160, 175, 176
create a new data member 183
create a new data members 162
create a new member function 162, 180, 181
create custom project stationery 38
create empty projects 34
create new projects from makefiles 33
create new projects using project stationery 33
deactivate automatic code completion 97
delete a breakpoint template 213
disable a breakpoint 210
disable a watchpoint 226
disable an eventpoint 222
disarm a logic analyzer 289
disconnect from a logic analyzer 289
dock a window by using a contextual menu 66
dock a window by using drag and drop 67
dock windows of the same kind 67
enable a breakpoint 210, 223
enable a watchpoint 226
examine items in the Folder Compare Results window 144
expand a docked window 71
expand browser panes 158
expand the editor window toolbar 86
export projects to XML files 37
float a window 69
generate project link maps 330
go to a particular line 107
hide the classes pane 160
import projects saved as XML files 37
indent text blocks 93
insert a reference template 111
issue command lines 293
kill program execution 194
look up symbol definitions 111
make a summation of two variables 235
make a window an MDI Child 70
manipulate variable formats 232
move a docked window 71
navigate browser data 150
navigate Code Completion window 100
navigate to a marker 109
open a recent workspace 76
open a single-class hierarchical window 169
open a workspace 74
open an Array window 243
open projects 35
open projects created on other hosts 35
open registers in a separate Registers window 247
open subprojects 40
open the Breakpoints window 207
open the Cache window 290
open the Command window 292
open the Expressions window 234
open the Flash Programmer window 261
open the Global Variables window 230
open the Hardware Diagnostics window 274
open the Log window 259
open the Processes window 256, 257
open the Profile window 291
open the Registers window 245
open the Symbolics window 255
open the symbols window 172
open the Target Settings window 391
open the Trace window 290
overstrike text (Windows) 92
print class hierarchies 167
print projects 36
remove a keyword from a keyword set 407
remove a marker from a source file 109
remove all markers from a source file 110
remove an executable file 409
remove panes from an editor window 89
remove remote connections 387
remove source trees 367
replace text in a single file 117
resize panes in an editor window 89
restart the debugger 195
resume program execution 194
run a program 195
save a copy of a workspace 75
save a workspace 74
save projects 35
save the contents of the Breakpoints window 207
search a single file 115
search for text across multiple files 127
search for text across multiple folders 121
search for text across multiple projects 123
search for text across multiple symbolics files 125
search with a text selection 132
select entire routines 92
select item in Code Completion window 101
select lines 92
select multiple lines 92
select rectangular portions of lines 92
select text in editor windows 91
set a breakpoint 207
set a conditional breakpoint 211
set a conditional eventpoint 223
set a conditional watchpoint 227
set a Log Point 216
set a Pause Point 217
set a Script Point 218
set a Skip Point 219
set a Sound Point 220
set a temporary breakpoint 211
set a Trace Collection Off eventpoint 221
set a Trace Collection On eventpoint 222
set a watchpoint 225
show the classes pane 160
sort the classes list 161
specify the default breakpoint template 214
start the debugger 192
step into a routine 193
step out of a routine 193
step over a routine 193
stop program execution 194
suppress dockable windows 70
toggle automatic punctuation balancing 95
toggle the symbol hint 195
trigger code completion by keyboard 97
trigger code completion from IDE menu bar 97
unapply file differences 141
undock a window 68
unfloat a window 69
unindent text blocks 93
update data from a logic analyzer 289
use an external editor on the Macintosh 360
use contextual menus 197
use the default workspace 73
use the document settings pop-up 87
use the Executables pane in the Symbolics window 255, 256
use the Files pane in the Symbolics window 255
use the Find Next command 130
use the Find Previous command 131
use the Functions list pop-up 106
use the Functions pane in the Symbolics window 255
use the Interfaces list pop-up 106
use the symbol hint 196
use the VCS pop-up 87
use virtual space 93
view a file path 45
view breakpoint properties 208
view browser data by contents 170
view browser data by inheritance 166
view class data from hierarchy window 158
view eventpoint properties 222
view global variables for different processes 230
view registers 245
view watchpoint properties 226

I
icon
for Tools menu 500
for VCS menu 500
icons
Active 207
file modification 88
for data members 161
for member functions 161
Inactive 207
IDE
and threading 420
Apple menu 486
Code Completion window 98
CodeWarrior menu 487
Data menu 483, 497
Debug menu 482, 495
Edit menu 475, 489
editing source code 91
editor 83
File menu 473, 478
Flash Programmer window 261
Hardware Diagnostics window 274
hardware tools 261
Help menu 486, 501
linkers 329
Mac-hosted 344
menu reference 473
preferences, working with 353
project manager and build targets 29
Project menu 479, 492
Scripts menu 501
Search menu 478, 490
target settings, working with 389
Tools menu 500
tools overview 24
User’s Guide overview 17
VCS menu 500
Window menu 477, 485, 498
Windows-hosted 344
workspaces 73
IDE Extras 500, 501

IDE Extras panel
  options
  Documents 422
  Find Reference using 427
  Launch Editor 431
  Launch Editor w/ Line # 432
  Menu bar layout 434
  Projects 436
  Symbolics 446
  Use Default Workspace’ 448
  Use External Editor 448
  Use Multiple Document Interface 449
  Use Script menu 449
  Use ToolServer menu 450
  Workspaces 452
  Zoom windows to full screen 452

IDE Extras preference panel 358
  options
  Context popup delay 359
  Documents 359
  Enable automatic Toolbar help 360
  Find Reference using 360
  Launch Editor 359
  Launch Editor w/ Line # 359
  Menu bar layout 359
  Projects 359
  Recent symbolics 359
  Use Default workspace 360
  Use External Editor 360
  Use Multiple Document Interface 359
  Use Script menu 360
  Use Third Party Editor 359
  Use ToolServer menu 360
  Zoom windows to full screen 359
  Use Third Party Editor option 450

IDE Preference Panels list 354
IDE Preference Panels, Font & Tabs 375
IDE Preference Panels, Font Settings 375
IDE preferences
  Activate Browser Coloring 377
  Activate Syntax Coloring 377
  Add 363, 365, 386
  Attempt to use dynamic type of C++, Object
    Pascal and SOM objects 381
  Auto Indent 375
  Auto Target Libraries 385
  Automatic Invocation 368
  Automatically launch applications when
    SYM file opened 384
  Background 377
  Balance Flash Delay 373
  Balance while typing 373
  Browser Commands 372
  Browser Path 361
  Build before running 356
  Cache Edited Files Between Debug
    Sessions 384
  Case sensitive 368
  Change 364, 365, 386
  Choose 365
  Classes 377
  Close Braces, Brackets, And
    Parentheses 371
  Close non-debugging windows 382
  Code Completion Delay 368
  Collapse non-debugging windows 382
  Comments 377
  Compiler thread stack 357
  Confirm "Kill Process" when closing or
    quitting 384
  Confirm invalid file modification dates when
    debugging 384
  Constants 377
  Context popup delay 359
  Debugger Commands 373
  Default file format 373
  Default size for unbounded arrays 381
  Disable third party COM plugins 362
  Display deprecated items 368
  Do nothing 382
  Do nothing to project windows 382
  Documents 359
  Don’t step into runtime support code 385
  Drag and drop editing 373
  Edit 377
  Edit Commands 372
Enable automatic Toolbar help 360
Enable Virtual Space 373
Enums 377
Failure 356
Find and compare operations 363
Find Reference using 360
Font 374
Font preferences 372
Foreground 377
Format Braces 370
Functions 377
Globals 377
Hide non-debugging windows 382
Include file cache 356
Indent Braces 370
Indent Case Within Switch Statement 371
Indent Code Within Braces 370
Insert Template Commands 372
Keywords 377
Language Settings 369
Launch Editor 359
Launch Editor w/ Line # 359
Left margin click selects line 373
Level 362
Macros 378
Maintain files in cache 384
Menu bar layout 359
Minimize non-debugging windows 382
Monitor for debugging 382
Move open windows to debugging monitor when debugging starts 383
Name 365
Open windows on debugging monitor during debugging 383
Other 378
Place Else On Same Line As Closing Brace 370
Place Opening Brace On Separate Line 370
Play sound after ‘Bring Up To Date’ & ‘Make’ 356
Project Commands 373
Project operations 363
Projects 359
Purge Cache 384
Recent symbolics 359
Recommended 358
Regular Expression 363
Relaxed C popup parsing 373
Remote Connection list 386
Remove 364, 365, 386
Save open files before build 356
Script 374
Select stack crawl window when task is stopped 384
Selection position 372
Set 361
Set 1, Set 2, Set 3, Set 4 377
Shielded folder list 363
Show all locals 380
Show message after building up-to-date project 356
Show tasks in separate windows 383
Show values as decimal instead of hex 381
Show variable location 380
Show variable types 380
Show variable values in source code 381
Size 374
Sort function popup 373
Sort functions by method name in symbolics window 381
Source Tree list 365
Strings 377
Success 356
Tab indents selection 374
Tab Inserts Spaces 375
Tab Size 374
Templates 378
Type 365
TypeDefs 378
Use Automatic Code Formatting 369
Use Concurrent Compiles 357
Use Debugging Monitor 382
Use Default workspace 360
Use External Editor 360
Use Local Project Data Storage 357
Use Multiple Document Interface 359
Use multiple undo 373
Use Script menu 360
Use Third Party Editor 359
Use ToolServer menu 360
User Specified 358
Variable values change 380
VCS Commands 373
Watchpoint indicator 380
Window follows insertion point 368
Window position and size 372
Zoom windows to full screen 359

IDE Preferences window 226, 334, 353, 354
Apply button 355
Cancel button 355
Factory Settings button 334, 354
IDE Preference Panels list 354
Import Panel 430
Import Panel button 335, 354
OK button 355
Revert Panel button 334, 354
Save button 335, 355
Ignore Extra Space checkbox 137
Ignored By Make File flag 401
Import button 350
Import Commands 350
Import Components menu command 518
Import Panel 430
Import Project command 37
Import Project menu command 518
importing
  projects saved as XML files 37
In Files page 126, 127
In Files tab 120
In Folders page 120, 121
In Folders tab 120
In Projects page 122, 123
In Projects tab 120
In Symbolics page 124, 125
In Symbolics tab 120
Inactive icon 207
Include file cache option
  Build Settings panel 430
Include Files 182
Include files 179
#include files, caching 430
indenting

Initial Directory field
Build Extras panel 430
Initializer 184
Insert Reference Template 112
Insert Reference Template menu command 518
Insert Template Commands option
  Editor Settings panel 431
inserting a reference template 111
inspecting
  project files 36
Installed Products button 503
Instances tab 206
Instruction Scheduling 405
interface files
  locating 105
Interface menu 51
Interfaces list pop-up
  in Files view of Project window 45
  using 106
interfaces list pop-up 86
interrupt time
  and profiler 320
interrupt time, and profiler 320
IP Address field 386

Java Exceptions Submenu
  No Exceptions command 520
Java Exceptions submenu
  All Exceptions command 504
Exceptions In Targeted Classes command 513
Uncaught Exceptions Only command 534
Java submenu 504, 513, 520, 534

Key Bindings 333, 335
Add 348
Customize 347
key bindings 111
keyboard conventions 20
keyboard shortcuts
  Find symbols with prefix 94
Find symbols with substring 94
Get next symbol 94
Get previous symbol 94

keys
Quote Key prefix 350

keywords
adding to a keyword set 406
removing from a keyword set 407

Keys option
Text Colors panel 431
Kill button 190
Kill command 194
Kill menu command 518
killing program execution 194

L
Language Parser option, in Generate Browser Data From menu 429
Launch Editor option
IDE Extras panel 431
Launch Editor w/ Line # option
IDE Extras panel 432
Launch Remote Host Application option
Remote Debugging settings panel 432
Launchable flag 401
Layout Editor panel options
Grid Size X 429
Grid Size Y 429
Show the component palette when opening a form 442
Show the object inspector when opening a form 442
layout management 48
layouts
moving 50
removing 48
renaming 50
least significant bit 283
Left Edges command 518
Left margin click selects line option
Editor Settings panel 432
Level option
Plugin Settings panel 432

libraries
profiler 303
Lifetime Based Register Allocation 405
line
going to in source code 107
Line And Column button 192
line and column indicator, in editor window 89
%line command-line string 432
Line Display 166
lines, selecting 92
lines, selecting multiple 92
lines, selecting rectangular portions of 92
link maps
generating for projects 330
Link Order page 47
Link Order tab 48
Link Order view 36, 50
Linker option
Target Settings panel 433
linkers 329
choosing 329
linking projects 330
Linux
modifier key mappings 20
list
of symbols in Browser Contents window 170
list boxes
Analyzer Type 287
Bit Value Modifier 249
Bitfield Name 249
Connection 264, 276
Connection Type 287
Debugger 287
File Set 127
Format 248
Functions 192
Project 123
Source 192
Symbolics 125
Target 123
Text View 250, 251
list menus
document settings 87
functions 86
interfaces 86
markers 86
VCS 87
list pop-up menus
Current Target 42
list pop-ups
Ancestor 166
Browser Access Filters 155
document settings 87
functions 86
interfaces 86
markers 86
Symbols 170
VCS 156
lists
All Sectors 271
File 123
File Mappings 420
File Set 127
Symbolics 125
Live Range Splitting 404
Load Settings button 262, 275
locating functions 105, 106
locating interface files 105
locating source code 105
Location of Relocated Libraries and Code
Resources option
   Debugger Settings panel 433
Log Message checkbox 216
Log Point 215, 216
Log Point Settings window 216
   Message text box 216
   Speak Message checkbox 216
   Stop in Debugger checkbox 217
   Treat as Expression checkbox 217
Log Point, clearing 217
Log Point, setting 216
Log System Messages 258
Log System Messages option
   Debugger Settings panel 433
Log Window
   Log System Messages option 258
   Log window 258
opening 259
logic analyzer 285
   Arm command 289
   arming 289
   configuring a project 285
   Connect command 288
   connecting to 289
   Disarm command 289
   disarming 289
   Disconnect command 289
   disconnect from 289
   Update Data command 289
   updating data from 289
   using 288
Logic Analyzer connection options
   Analyzer Can Cause Target Breakpoint checkbox 288
   Analyzer Configuration File text box 287
   Analyzer Slot text box 287
   Analyzer Type list box 287
   Connection Type list box 287
   Debugger list box 287
   Host Name text box 287
   Name text box 287
   Target Breakpoint Can Cause Analyzer Trigger checkbox 288
   Trace Support File text box 288
Long Word option button 277, 279, 282
longjmp() 307
looking up symbol definitions 111
Loop Transformations 405
Loop Unrolling 405
Loop Unrolling (Opt For Speed Only) 405
Loop-Invariant Code Motion 405
LSB 283
M
Mac OS
   QuickHelp 110
   QuickView 110, 111
   THINK Reference 111
Mac OS X API 428
machines, defined 256
Macintosh
creating files 58
using an external editor 360
Macintosh menu layout 486
Macro file option, in Generate Browser Data
   From menu 429
Macros option 378
Maintain Files In Cache option 419
Maintain Files in Cache option
   Global Settings panel 433
Make command 50, 51, 52
Make menu command 519
Make option 427
Make toolbar button 42
Makefile Importer wizard 33
makefiles
   converting into projects 33
managing
   build targets 53
   projects 32
   targets 53
managing files, tasks 57
manipulating program execution 203
   Breakpoints window 204
manual conventions 19
markers 108
   adding to a source file 109
   navigating to 109
   removing all from source files 110
   removing from source files 109
Markers list pop-up 86
Markers list, in Remove Markers window 109
Match Whole Word checkbox 114, 116, 120
matching
   any character with regular expressions 134
   replace strings to find strings with regular
      expressions 134, 135
   with simple regular expressions 133
Maximize Window menu command 519
Maximum Invert Convergence subtest 284
   .mcp 35
MDI 359, 434, 449
   and dockable windows 63
   making a window an MDI child 70
Member Function Declaration 182
member functions
   creating 162, 181
   identifier icons 161
Member Functions pane 161
   in Class Browser window 155
memory aliasing, defined 283
memory dump 537
Memory Range On Target option button 273
Memory Read / Write panel 276
memory tests
   Address 283
   Bus Noise 284
      address lines 284
      data lines 285
   Bus Noise test
      Full Range Converging subtest 284
      Maximum Invert Convergence
         subtest 284
   Sequential subtest 284
   Walking Ones 282
   Walking Ones test
      Address Line fault 282
      Data Line fault 283
      Ones Retention subtest 283
      Retention fault 283
      Walking Ones subtest 283
      Walking Zeros subtest 283
      Zeros Retention subtest 283
Memory Tests panel 280
   Address test 283
   Bus Noise test 284
      address lines 284
      data lines 285
   Walking Ones test 282
memory usage 317
Memory window 237
memory, working with 237
Menu
   Current Target 345
   menu
   Search 111
Menu bar layout option
   IDE Extras panel 434
   menu commands
| About Metrowerks CodeWarrior | 503 |
| Add Files | 503 |
| Add Window | 503 |
| Apply Difference | 141, 504 |
| Arm | 289 |
| Balance | 504 |
| Bottom Edges | 504 |
| Break | 505 |
| Break On C++ Exception | 505 |
| Break on Java Exceptions | 505 |
| Breakpoints | 505 |
| Breakpoints Window | 505 |
| Bring To Front | 505 |
| Bring Up To Date | 505 |
| Browser Contents | 505 |
| Build Progress | 505 |
| Build Progress Window | 505 |
| Can’t Redo | 475, 489 |
| Can’t Undo | 475, 489 |
| Cascade | 506 |
| Change Program Counter | 506 |
| Check Syntax | 506 |
| Class Browser | 506 |
| Class Hierarchy | 507 |
| Class Hierarchy Window | 507 |
| Clear | 507 |
| Clear All Breakpoints | 507 |
| Clear All Watchpoints | 507 |
| Clear Breakpoint | 507 |
| Clear Eventpoint | 507 |
| Clear Watchpoint | 508 |
| Close | 508 |
| Close All | 508 |
| Close All Editor Documents | 508 |
| Close Catalog | 508 |
| Close Workspace | 508 |
| CodeWarrior Help | 509 |
| Collapse Window | 509 |
| Commands & Key Bindings | 508 |
| Compare Files | 137, 509 |
| Compile | 509 |
| Complete Code | 509 |
| Connect | 288, 509 |
| Copy | 509 |

| Copy To Expression | 510 |
| Create Design | 510 |
| Create Group | 510 |
| Create Target | 510 |
| Cycle View | 510 |
| Debug | 511 |
| Delete | 511 |
| Disable Breakpoint | 511 |
| Disable Watchpoint | 511 |
| Disarm | 289 |
| Disassemble | 511 |
| Disconnect | 289 |
| Display Grid | 511 |
| Enable Breakpoint | 512 |
| Enable Watchpoint | 512 |
| Enter Find String | 131, 512 |
| Enter Replace String | 512 |
| Errors And Warnings | 513 |
| Errors And Warnings Window | 513 |
| Exit | 513 |
| Expand Window | 513 |
| Export Project | 513, 518 |
| Export Project as GNU Makefile | 513 |
| Expressions | 513 |
| Expressions Window | 513 |
| Find | 115, 514 |
| Find and Open ‘Filename’ | 515 |
| Find and Open File | 515 |
| Find And Replace | 516 |
| Find Definition | 514 |
| Find Definition & Reference | 514 |
| Find In Files | 514 |
| Find In Next File | 515 |
| Find In Previous File | 515 |
| Find Next | 130, 515 |
| Find Previous | 131, 515 |
| Find Previous Selection | 515 |
| Find Reference | 516 |
| Find Selection | 132, 516 |
| Get Next Completion | 516 |
| Get Previous Completion | 516 |
| Global Variables | 517 |
| Global Variables Window | 517 |
| Go Back | 517 |
OK button 263, 275
Ones Retention subtest 283
Only Show Different Files checkbox 137
Open command 58
Open File 163
Open In Windows Explorer command 45
Open menu command 521
Open Recent menu command 521
Open Scripts Folder menu command 521
Open windows on debugging monitor during debugging option
Windowing panel 435
Open Workspace menu command 521
opening 172
a recent workspace 76
a single-class hierarchical window 169
files 58
Flash Programmer window 261
Hardware Diagnostics window 274
projects 35
projects from other hosts 35
subprojects 40
Symbolics window 255
symbols window 172
workspaces 74
opening last project (default workspace) 448
opening last project, preventing (default workspace) 448
openings
registers in a separate Registers window 247
optimizations
Arithmetic Optimizations 404
Branch Optimizations 404
Common Subexpression Elimination 404
Copy And Expression Propagation 404
Copy Propagation 404
Dead Code Elimination 404
Dead Store Elimination 404
Expression Simplification 404
Global Register Allocation 404
Global Register Allocation Only For Temporary Values 404
Instruction Scheduling 405
Lifetime Based Register Allocation 405
Live Range Splitting 404
Loop Transformations 405
Loop Unrolling 405
Loop Unrolling (Opt For Speed Only) 405
Loop-Invariant Code Motion 405
Peephole Optimization 404
Register Coloring 405
Repeated 405
Strength Reduction 405
Vectorization 405
option buttons
All text 114, 117, 120
Byte 277, 279, 281
Code Only 115, 117, 120
Comments Only 115, 117, 120
Entire Flash 273
File on Host 273
File on Target 273
Long Word 277, 279, 282
Memory Range on Target 273
Read 277, 279
Word 277, 279, 282
Write 277, 279
options 423
Access Paths settings panel 362, 393
Activate Browser 514
Activate Browser Coloring 415
Activate Syntax Coloring 415, 420
Add Default 415
Always Search User Paths 415
Application 416
Arguments 416
Attempt to use dynamic type of C++, Object Pascal and SOM objects 416
Auto Indent 416
Auto Target Libraries 416
Automatic Invocation 416
Automatically Launch Applications When SYM File Opened 417
Auto-target Libraries 416
Background 417
Balance Flash Delay 417
Balance while typing 417, 418
Bring Up To Date 427
Browse in processes window 386, 387
Browser Commands 418
Browser Path 418
Build before running 418
Build Extras settings panel 396
Build Settings preference panel 355
Cache Edited Files Between Debug Sessions 419
Cache Subprojects 419
Cache symbols between runs 419
Case Sensitive 419
Checksum panel 272
choosing host application for non-executable files 430
Classes 377
Close non-debugging windows 420
Code Completion Delay 420
Code Completion preference panel 367
Code Formatting preference panel 369
Collapse non-debugging windows 420
Comments 420
Compiler 420
Compiler thread stack 420
Concurrent Compiles preference panel 357
Configuration panel 275
Confirm “Kill Process” when closing or quitting 421
Confirm invalid file modification dates when debugging 421
Connection Type 386, 387
Constants 377
Context popup delay 421
Debugger Commands 421
Debugger preference panels 379
Debugger Settings 258
Debugger Settings panel 410
Default File Format 421
Default size for unbounded arrays 422
Disable third party COM plugins 422
Display Deprecated Items 422
Display Settings preference panel 379
Do nothing 422
Do nothing to project windows 422
Drag and drop editing 423
Dump internal browse information after compile 423
Edit Commands 423
Edit Language 423
Editor preference panels 367
Editor Settings preference panel 371
Enable automatic Toolbar help 423
Enable remote debugging 424
Enable Virtual Space 424
Enums 377
Environment Settings 424
Erase / Blank Check panel 270
Failure 427
File Mappings settings panel 400
File Type 420
Flash Configuration panel 265
Font & Tabs preference panel 373, 376
Functions 377
General preference panels 355
Generate Browser Data From 429
Global Optimizations settings panel 402
Globals 377
Help Preferences panel 361
IDE Extras preference panel 358
Import Panel 430
Macros 378
Maintain files in cache 419
Make 427
Memory Read / Write panel 276
Memory Tests panel 280
Other 378
Other Executables settings panel 407
Plugin Settings preference panel 361
Program / Verify panel 267
Purge Cache 419
Remote Connections preference panel 385
Remote Debugging settings panel 411
Require Framework Style Includes 437
Runtime Settings panel 398
Scope Loop panel 278
Set 1, Set 2, Set 3, Set 4 377
setting for browser 147
Shielded Folders preference panel 362
Source Trees preference panel 364
Target Configuration panel 263
Target Settings panel 392
Templates 378
TypeDefs 378
Use Multiple Document Interface 63
User specified 411
Window Follows Insertion Point 451
Window Settings preference panel 381
Organization pane 267
original process, breakpoint property 209
original-target, breakpoint property 209
other editor windows 88
Other Executables settings panel 407
Other option 378
Output Directory option
Target Settings panel 435
Overlays tab 48
overstrike 92
overstriking text (Windows) 92
overtype. See overstrike.
overview
of browser 25
of build system 26
of CodeWarrior 21
of debugger 26
of editor 25
of IDE project manager and build targets 29
of IDE tools 24
of IDE User’s Guide 17
of project manager 25
of search engine 25

Pane Expand box 143, 190
Pane resize bar 130, 140, 143, 191
pane resize bar
in File Compare Results window 140
in Folder Compare Results window 143
pane splitter controls, in editor window 89
panel
Display Settings 226
panels
Analyzer Connections 285
Font & Tabs 375
panes
adding to editor window 89
Destination 140
Device 266
Differences 141
Files in Both Folders 143
Files Only in Destination 144
Files Only in Source 144
Flash Programmer 262
Hardware Diagnostics 275
Organization 267
removing from editor window 89
resizing in an editor window 89
Results 130
Sector Address Map 267
Source 140, 191
Source Code 130
Stack 191
Variables 191
parameter lists
completing code 102
Passes text box 282
path caption 88
Pause Point 215, 217
Pause Point, clearing 218
Pause Point, setting 217
Peephole Optimization 404
Play sound after ‘Bring Up To Date’ & ‘Make’ option
Build Settings panel 435
Plugin Diagnostics
All Info option 433
Errors Only option 433

Page Setup command 521
pages
In Files 126
In Folders 120
in project window 43
In Projects 122
In Symbolics 124
PalmQuest reference 428
Pane Collapse 159
Pane Collapse box 143, 190
Pane Expand 158
None option 432
plug-in diagnostics
  disabling 503
  enabling 503
Plugin Settings panel
  options
    Level 432
Plugin Settings preference panel 361
  options
    Disable third party COM plugins 362
    Level 362
plug-ins
  saving information about those installed in IDE 503
  viewing those installed in IDE 503
pop-up menus
  document settings 87
  functions 86
  interfaces 86
  markers 86
  VCS 87
pop-ups
  Ancestor 166
  Browser Access Filters 155
  Symbols 170
  VCS 156
Post-linker option
  Target Settings panel 435
  PPCTimeBase 305, 317
  Precompile menu command 522
  Precompiled File flag 401
precompiled headers
  caching 430
preference panels
  Build Settings 355
  Code Completion 367
  Code Formatting 369
  Concurrent Compiles 357
  Display Settings 379
  Editor Settings 371
  Font & Tabs 373, 376
  Help Preferences 361
  IDE Extras 358
  Plugin Settings 361
Remote Connections 385
  reverting 438
  Shielded Folders 362
  Source Trees 364
  Window Settings 381
preferences
  Activate Browser Coloring 377
  Activate Syntax Coloring 377
  Add 363, 365, 386
  Apply button 355
  Attempt to use dynamic type of C++, Object Pascal and SOM objects 381
  Auto Indent 375
  Auto Target Libraries 385
  Automatic Invocation 368
  Automatically launch applications when SYM file opened 384
  Background 377
  Balance Flash Delay 373
  Balance while typing 373
  Browser Commands 372
  Browser Path 361
  Build before running 356
  Cache Edited Files Between Debug Sessions 384
  Cancel button 355
  Case sensitive 368
  Change 364, 365, 386
  Choose 365
  Classes 377
  Close Braces, Brackets, And Parentheses 371
  Close non-debugging windows 382
  Code Completion Delay 368
  Collapse non-debugging windows 382
  Comments 377
  Compiler thread stack 357
  Confirm "Kill Process" when closing or quitting 384
  Confirm invalid file modification dates when debugging 384
  Constants 377
  Context popup delay 359
  Debugger 379
Debugger Commands 373
Default file format 373
Default size for unbounded arrays 381
Disable third party COM plugins 362
Display deprecated items 368
Do nothing 382
Do nothing to project windows 382
Documents 359
Don’t step into runtime support code 385
Drag and drop editing 373
Edit 377
Edit Commands 372
Editor 367
Enable automatic Toolbar help 360
Enable Virtual Space 373
Enums 377
Export Panel button 334, 354
Factory Settings button 334, 354
Failure 356
Find and compare operations 363
Find Reference using 360
Font 374
Font preferences 372
for IDE 353
Foreground 377
Format Braces 370
Functions 377
General 355
Globals 377
Hide non-debugging windows 382
IDE Preference Panels list 354
IDE window 353
Import Panel button 335, 354
Include file cache 356
Indent Braces 370
Indent Case Within Switch Statement 371
Indent Code Within Braces 370
Insert Template Commands 372
Keywords 377
Language Settings 369
Launch Editor 359
Launch Editor w/ Line # 359
Left margin click selects line 373
Level 362
Macros 378
Maintain files in cache 384
Menu bar layout 359
Minimize non-debugging windows 382
Monitor for debugging 382
Move open windows to debugging monitor when debugging starts 383
Name 365
OK button 355
Open windows on debugging monitor during debugging 383
Other 378
Place Else On Same Line As Closing Brace 370
Place Opening Brace On Separate Line 370
Play sound after ‘Bring Up To Date’ & ‘Make’ 356
Project Commands 373
Project operations 363
Projects 359
Purge Cache 384
Recent symbolics 359
Recommended 358
Regular Expression 363
Relaxed C popup parsing 373
Remote Connection list 386
Remove 364, 365, 386
Revert Panel button 334, 354
Save button 335, 355
Save open files before build 356
Script 374
Select stack crawl window when task is stopped 384
Selection position 372
Set 361
Set 1, Set 2, Set 3, Set 4 377
Shielded folder list 363
Show all locals 380
Show message after building up-to-date project 356
Show tasks in separate window 383
Show values as decimal instead of hex 381
Show variable location 380
Show variable types 380

IDE 5.7 User’s Guide 577
Show variable values in source code 381
Size 374
Sort function popup 373
Sort functions by method name in symbolics window 381
Source Tree list 365
Strings 377
Success 356
Tab indents selection 374
Tab Inserts Spaces 375
Tab Size 374
Templates 378
Type 365
TypeDefs 378
Use Automatic Code Formatting 369
Use Concurrent Compiles 357
Use Debugging Monitor 382
Use Default workspace 360
Use External Editor 360
Use Local Project Data Storage 357
Use Multiple Document Interface 359
Use multiple undo 373
Use Script menu 360
Use Third Party Editor 359
Use ToolServer menu 360
User Specified 358
Variable values change 380
VCS Commands 373
Watchpoint indicator 380
Window follows insertion point 368
Window position and size 372
Zoom windows to full screen 359
Preferences menu command 522
Prefix file option, in Generate Browser Data From menu 429
prefix keys
  Quote Key 350
Pre-linker option
  Target Settings panel 436
preprocessor directives 304
  C/C++ 316
Previous Result button 130
print
  file selections 61
Print command 61, 522
printing
  class hierarchies 167
  files 61
  projects 36
process
  attaching debugger to 258
process cycle
  of software development 21
processes
  related to machines 256
  viewing global variables for 230
Processes menu command 522
Processes window 256, 386
  opening 256, 257
Processes Window menu command 522
products
  saving information about those installed in IDE 503
  viewing those installed in IDE 503
Profile window
  opening 291
profiler
  libraries 303
Profiler Function Reference 318
  ProfilerClear() 322
  ProfilerDump() 321
  ProfilerGetDataSizes() 321
  ProfilerGetStatus() 320
  ProfilerInit() 318
  ProfilerSetStatus() 320
  ProfilerTerm() 320
ProfilerClear() 322
ProfilerDump() 307, 321
ProfilerGetDataSizes() 321
ProfilerGetStatus() 320
ProfilerInit() 317, 318
  warning 308
ProfilerSetStatus() 320
ProfilerTerm() 307, 320
  warning 308
profiling
  activating 300
  by function 301, 316
exceptions 307
inline functions 315
setjmp() 307
program
killing execution 194
resuming execution 194
running 195
stopping execution 194
Program / Verify panel 267
Program Arguments field
of Runtime Settings panel (Windows) 436
Program Arguments option
Runtime Settings panel 436
Program button 270
Program Entry Point option
Debugger Settings panel 436
program execution, manipulating 203
project
configuring for a logic analyzer 285
Project Commands option
Editor Settings panel 436
project data folder 448
Project Headers checkbox 123
Project Inspector command 36
Project list box 123
project manager 29
overview 25
Project menu 436, 479, 492
Remove Object Code command 481, 494
Stop Build command 481, 494
Project operations option
Shielded Folders panel 436
Project Settings 300
Project Sources checkbox 123
project stationery
creating 38
custom 38
Project window
about Files page 43
Current Target list pop-up 42
Files view
Checkout Status column 44
Code column 44
Data column 44
Debug column 44
File column 44
Interfaces list pop-up 45
Sort Order button 45
Target column 44
Touch column 44
Make toolbar button 42
Synchronize Modification Dates toolbar button 42
Target Settings toolbar button 42
project window 41
Link Order page 47
pages 43
Targets page 47
project window, about 41
project, defined 29
projects
about subprojects 39
advanced topics 38
choosing default 37
closing 38
creating custom stationery 38
creating empty 34
creating subprojects 39
creating using makefiles 33
creating using stationery 33
data folder 448
exporting to XML files 37
generating link maps for 330
importing XML versions of 37
inspecting files 36
linking 330
managing 32
opening 35
opening from other hosts 35
printing 36
project window 41
project window pages 43
project window, about 41
reopening last one used (default workspace) 448
reopening last one used, preventing (default workspace) 448
saving 35
searching (multiple) 123
strategies for 40
subprojects, strategies for 40
working with 29
Projects option
IDE Extras panel 436
properties
condition, breakpoint 209
file-info, breakpoint 209
hardware, breakpoint 209
hit count, breakpoint 209
name, breakpoint 209
original process, breakpoint 209
original-target, breakpoint 209
serial number, breakpoint 209
thread, breakpoint 209
times hit, breakpoint 209
times left, breakpoint 209
type, breakpoint 209
punctuation balancing, toggling 95
punctuation, balancing 94
pure virtual
icon for 161
Purge Cache button 436
Purge Cache option 419
purging cache 436
purpose
of breakpoints 203
of Browser Contents window 169
of Classes pane in browser 159
of Data Members pane 162
of eventpoints 203
of Member functions pane 161
of Multi-Class Hierarchy window 165
of Single-Class Hierarchy window 168
of Source pane 163
of special breakpoints 203
of status area in browser 163
of Symbols window 171
of watchpoints 203
QuickView 110, 111, 428
QuickView, Mac OS 111
QuickView, THINK Reference 111
Quote Key prefix 350
assigning 351
R
Read button 250
Read option button 277, 279
Recursive Search column, in Access Paths
panel 395
Redo button 141
Redo menu command 522
reference information
for IDE menus 473
reference template 111
reference template, inserting 111
reference templates (Macintosh) 111
Refresh All Data menu command 523
REGISTER 455
Register Coloring 405
Register Description option
of Text View pop-up menu 251
Register Details option
of Text View pop-up menu 251
Register Details window 247
Address text box 248
Bit Value Modifier list box 249
Bit Value text box 249
Bitfield Description text view option 251
Bitfield Name list box 249
Browse button 248, 250
Description 249
Description File text box 248, 250
Format list box 248
Read button 250
Register Description text view option 251
Register Details text view option 251
Register display 249, 251
Register Name 248
Reset Value button 250
Revert button 250
Text View list box 250, 251
using 250
Q
QuickDraw 434
QuickHelp (Mac OS) 110
Write button 250
XML
  file locations 461
  sample files 462
  specification 453
Register Details Window command 247
Register Details Window menu command 523
Register display 249, 251
Register Name 248
Register Windows menu command 523
registers
  changing data views of 246
  changing values of 246
  FPU Registers 245
  General Registers 245
  host-specific 245
  Register Details window 247
  viewing 245
  viewing details of 247
Registers folder 247
Registers menu command 523
Registers window 244
  opening 245
  opening more than one 247
Registry Key option
  of Source Trees preference panel 447
Registry Key option, in Type pop-up menu 447
regular breakpoints 204
Regular Expression checkbox 114, 117, 120
Regular Expression option
  Shielded Folders panel 437
regular expressions 132
  .*[_]Data 364
  \(.*\) 364
  choosing one character from many 134
CVS 364
defined 132
grouping 134
  matching any character 134
  matching simple expressions 133
  using the find string in the replace
  string 134, 135
Relative to class field 177
Relaxed C popup parsing option
Editor Settings panel 437
release notes 17
remembering last project (default
workspace) 448
remembering last project, turning off (default
workspace) 448
remote connections
  adding 386
  changing 387
  removing 387
Remote Connections preference panel 385
options
  Add 386
  Change 386
  Remote Connection list 386
  Remove 386
Remote Debugging settings panel 411
  Connection pop-up menu 412
options
  Launch remote host application 432
Remove A Set button 127
Remove button 395
Remove button, in Remove Markers window 109
Remove command 48
Remove Markers window 108
  Cancel button 109
  Done button 109
  Markers list 109
  Remove button 109
Remove Object Code & Compact menu
command 523
Remove Object Code menu command 523
Remove Toolbar Item 345
removing
  build targets 48, 53
desktop background from behind IDE. See
  Use Multiple Document Interface, turning
  on.
files 48
gray background from behind IDE. See Use
  Multiple Document Interface, turning off.
grids 48
layouts 48
remote connections 387
source trees 367
targets 48, 53
Rename Breakpoint button 206
Rename command 50, 54
renaming
  build targets 51, 54
  files 50
  groups 50
  layouts 50
  targets 50, 51, 54
reopening last project used
  in default workspace 448
  suppressing in the default workspace 448
Repeated optimizations 405
Replace All button 116, 119
Replace All menu command 524
Replace and Find Next menu command 524
Replace and Find Previous command 524
Replace button 116, 119
Replace command 117
Replacement
  text in a single file 117
  text, overview 113
Require Framework Style Includes 437
Reset Value button 250
Reset Window Toolbar command in Toolbar
  submenu 46, 525
resetting
  toolbars 346
resize bars
  Pane 130, 191
Resize submenu
  To Smallest Height command 534
  To Smallest Width command 534
resizing
  panes in an editor window 89
Resource File flag 401
Restart command 195
restarting
  debugger 195
Restore Window command (Windows) 526
Restrict Address Range checkbox 269
Result Count text box 129
results
  finding problems 313
  of multi-item search 128
  opening 309
  sorting 310
Results pane 130
Resume button 190
Resume command 194
Resume menu command 526
resuming program execution 194
Retention fault 283
Revert button 250
Revert command 62
Revert menu command 526
reverting
  files 62
  preference panels 438
  settings panels 438
  revision control 451, 500
routine
  stepping into 193
  stepping out of 193
  stepping over 193
routine, selecting entirely 92
Run App/Script 341
Run button 190
Run command 51, 52, 195
Run menu command 436, 526
Run To Cursor menu command 527
running
  a program 195
Runtime Settings panel 398
Host Application For Libraries & Code
  Resources field 430
options
  Add 399
  Change 399
  Environment Settings 399
  Host Application for Libraries & Code
    Resources 399, 430
  Program Arguments 399, 436
  Remove 400
  Value 400
Variable 400
  Working Directory 399, 452
  Program Arguments field (Windows) 436

S
Save a Copy As command 60
Save All command 59
Save command 59
Save Default Window menu command 527
Save open files before build option
  Build Settings panel 438
Save project entries using relative paths option
  Target Settings panel 46, 438
Save Settings button 262, 275
Save This Set button 127
Save Workspace As menu command 528
Save Workspace menu command 527
  saving
    a copy of a workspace 75
    all files 59
    file copies 60
    files 59
    information about installed plug-ins 503
    information about installed products 503
    projects 35
    workspaces 74
Scope Loop panel 278
Script Point 215, 218
Script Point Settings window
  Stop in Debugger checkbox 219
Script Point, clearing 219
Script Point, setting 218
  (Scripts) folder 501, 521
Scripts menu 501
Scripts option
  Font & Tabs panel 439
search
  single characters with regular expressions 134
  using finds strings in replace strings with regular expressions 135
Search Cached Sub-Targets checkbox 123
Search Criteria text box 129
search engine
  overview 25
Search In text/list box 121
Search menu 111, 478, 490
Search Results window 128
  Next Result button 130
  Pane resize bar 130
  Previous Result button 130
  Result Count text box 129
  Results pane 130
  Search Criteria text box 129
  setting default size and position of 527
  Source Code pane 130
  Source Code Pane disclosure triangle 130
  Stop button 129
  Warnings button 129
Search Selection Only checkbox 114, 117
Search Status column, in Access Paths panel 395
Search Sub-Folders checkbox 121
Search Up checkbox 114, 117
searching
  choosing one character from many in regular expressions 134
  grouping regular expressions 134
  multiple files 127
  multiple folders 121
  multiple projects 123
  multiple symbolics files 125
  single characters with regular expressions 134
  single files 115
  using finds strings in replace strings with regular expressions 134
  using regular expressions 132
  with simple regular expressions 133
Sector Address Map pane 267
seeing desktop background behind IDE. See Use
  Multiple Document Interface, turning off
Segments tab 48
Select All menu command 528
Select stack crawl window when task is stopped option
  Global Settings panel 439
Selected Item group 144
selecting
Code Completion window items 101
text in editor windows 91
selecting entire routines 92
selecting lines 92
selecting multiple lines 92
selecting rectangular portions of lines 92
Selection position option
Editor Settings panel 439
selections
searching (text) 132
Send To Back menu command 528
Sequential subtest 284
serial number, breakpoint property 209
Set 1, Set 2, Set 3, Set 4 377
Set Breakpoint menu command 528
Set Default Breakpoint Template button 206
Set Default Project command 37
Set Default Project menu command 528
Set Default Target menu command 528
Set Eventpoint menu command 528
Set Watchpoint menu command 529
setjmp() 307
setting
browser options 147
temporary breakpoints 211
setting access breakpoint 225
setting default size and position of windows 527
settings
Add 365, 395, 399, 402
Add Default 395
Always Search User Paths 394
Application 398
Apply button 391
Arguments 398
Auto-target Libraries 411
Cache subprojects 397
Cache symbolics between runs 411
Cancel button 391
Change 365, 395, 399, 402
Choose 365, 393
Clear 393
Compiler 401
Default language entry point 411
Details 403
Dump internal browse information after compile 398
Edit Language 402
Environment Settings 399
Export Panel button 391
Extension 401
Factory Settings button 390
Faster Execution Speed 403
File Mappings list 401
File Type 401
Flags 401
Generate Browser Data From 398
Host Application for Libraries & Code Resources 399
Host Flags 395
IDE window 389
Ignored By Make flag 401
Import Panel button 391
Initial directory 398
Interpret DOS and Unix Paths 395
Launchable flag 401
Linker 392
Log System Messages 411
Name 365
OK button 391
Optimization Level slider 403
Output Directory 393
Post-linker 392
Precompiled File flag 401
Pre-linker 392
Program Arguments 399
Program entry point 410
Remove 365, 395, 400, 402
Require Framework Style Includes 395
Resource File flag 401
Revert Panel button 390
Save button 391
Save project entries using relative paths 393
Smaller Code Size 403
Source Tree list 365
Stop at Watchpoints 411
Stop on application launch 410
Target Name 392
Target Settings Panels list 390
Type 365
Update data every $n$ seconds 411
Use External Debugger 398
Use modification date caching 397
User specified 411
Value 400
Variable 400
Working Directory 399
settings panels
Access Paths 362, 393
Build Extras 396, 514
Debugger Settings 258, 410
File Mappings 400
Global Optimizations 402
Other Executables 407
Remote Debugging 411
reverting 438
Runtime Settings 398
Source Trees 364
Target Settings 392
setup
code completion 96
Shielded Folders panel
options
Find and compare operations 427
Project operations 436
Regular Expression 437
Shielded Folders preference panel 362
options
Add 363
Change 364
Find and compare operations 363
Project operations 363
Regular Expression 363
Remove 364
Shielded folder list 363
Shift Right menu command 529
shortcut conventions 20
Show all locals option
Display Settings panel 439
Show Breakpoints menu command 507, 529
Show Classes 160
Show Classes pane 163
Show Floating Toolbar command 517
Show Floating Toolbar command in Toolbar submenu 529
Show Inherited 155
Show Log button 262, 282
Show Main Toolbar command 517
Show message after building up-to-date project option
Build Settings panel 442
Show private 156
Show protected 156
Show public 156
Show tasks in separate windows option
Display Settings panel 442
Show the component palette when opening a form option
Layout Editor panel 442
Show the object inspector when opening a form option
Layout Editor panel 442
Show Types menu command 529
Show values as decimal instead of hex option
Display Settings panel 442
Show variable location option
Display Settings panel 442
Show variable types option
Display Settings panel 443
Show variable values in source code option
Display Settings panel 443
Show Window Toolbar command 518
Show Window Toolbar command in Toolbar submenu 529
showing
classes pane 160, 161
shrinking panes, in browser 159
Single Class Hierarchy Window 154
single files, searching 115
single-class hierarchical window opening 169
Single-Class Hierarchy window 168
difference from Multi-Class Hierarchy window 168
single-file Find and Replace window 115
single-file Find window 113
size
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>setting default for unbounded arrays</td>
<td>422</td>
</tr>
<tr>
<td>Size option</td>
<td></td>
</tr>
<tr>
<td>Font &amp; Tabs panel</td>
<td>443</td>
</tr>
<tr>
<td>Size text box</td>
<td>273</td>
</tr>
<tr>
<td>Skip Point</td>
<td></td>
</tr>
<tr>
<td>215, 219</td>
<td></td>
</tr>
<tr>
<td>Skip Point, clearing</td>
<td>219</td>
</tr>
<tr>
<td>Skip Point, setting</td>
<td>219</td>
</tr>
<tr>
<td>software</td>
<td></td>
</tr>
<tr>
<td>development process cycle</td>
<td>21</td>
</tr>
<tr>
<td>Solaris</td>
<td></td>
</tr>
<tr>
<td>modifier key mappings</td>
<td>20</td>
</tr>
<tr>
<td>Sort Alphabetic</td>
<td>160, 161</td>
</tr>
<tr>
<td>Sort function popup option</td>
<td></td>
</tr>
<tr>
<td>Editor Settings panel</td>
<td>445</td>
</tr>
<tr>
<td>Sort functions by method name in symbolics</td>
<td></td>
</tr>
<tr>
<td>window option</td>
<td></td>
</tr>
<tr>
<td>Display Settings panel</td>
<td>443</td>
</tr>
<tr>
<td>Sort Hierarchical</td>
<td>160, 161</td>
</tr>
<tr>
<td>Sort Order button</td>
<td></td>
</tr>
<tr>
<td>in Files view of Project window</td>
<td>45</td>
</tr>
<tr>
<td>sorting</td>
<td></td>
</tr>
<tr>
<td>classes list</td>
<td>161</td>
</tr>
<tr>
<td>Functions list pop-up (alphabetically)</td>
<td>106, 107</td>
</tr>
<tr>
<td>sorting data</td>
<td>310</td>
</tr>
<tr>
<td>Sound Point</td>
<td></td>
</tr>
<tr>
<td>215, 220</td>
<td></td>
</tr>
<tr>
<td>Sound Point Settings window</td>
<td></td>
</tr>
<tr>
<td>Stop in Debugger checkbox</td>
<td>220</td>
</tr>
<tr>
<td>Sound Point, clearing</td>
<td>221</td>
</tr>
<tr>
<td>Sound Point, setting</td>
<td>220</td>
</tr>
<tr>
<td>Sound Point, Speak Message</td>
<td>220</td>
</tr>
<tr>
<td>Source box</td>
<td>136</td>
</tr>
<tr>
<td>source code</td>
<td></td>
</tr>
<tr>
<td>disabling breakpoints</td>
<td>210</td>
</tr>
<tr>
<td>disabling eventpoints</td>
<td>222</td>
</tr>
<tr>
<td>disabling special breakpoints</td>
<td>228</td>
</tr>
<tr>
<td>disabling watchpoints</td>
<td>226</td>
</tr>
<tr>
<td>editing</td>
<td>91</td>
</tr>
<tr>
<td>enabling breakpoints</td>
<td>210, 223</td>
</tr>
<tr>
<td>enabling special breakpoints</td>
<td>228</td>
</tr>
<tr>
<td>enabling watchpoints</td>
<td>226</td>
</tr>
<tr>
<td>going to a particular line</td>
<td>107</td>
</tr>
<tr>
<td>locating</td>
<td>105</td>
</tr>
<tr>
<td>setting breakpoints in</td>
<td>207</td>
</tr>
<tr>
<td>setting watchpoints in</td>
<td>225</td>
</tr>
<tr>
<td>viewing breakpoint properties</td>
<td>208</td>
</tr>
<tr>
<td>viewing eventpoint properties</td>
<td>222</td>
</tr>
<tr>
<td>viewing watchpoint properties</td>
<td>226</td>
</tr>
<tr>
<td>Source Code pane</td>
<td>130</td>
</tr>
<tr>
<td>Source Code Pane disclosure triangle</td>
<td></td>
</tr>
<tr>
<td>Source code, navigating</td>
<td>105</td>
</tr>
<tr>
<td>source file</td>
<td></td>
</tr>
<tr>
<td>adding markers to</td>
<td>109</td>
</tr>
<tr>
<td>Source File button</td>
<td>191</td>
</tr>
<tr>
<td>source files</td>
<td></td>
</tr>
<tr>
<td>removing all markers from</td>
<td>110</td>
</tr>
<tr>
<td>removing markers from</td>
<td>109</td>
</tr>
<tr>
<td>source item, for comparison</td>
<td>136</td>
</tr>
<tr>
<td>Source list box</td>
<td>192</td>
</tr>
<tr>
<td>Source pane</td>
<td>140, 163, 191</td>
</tr>
<tr>
<td>in Symbols window</td>
<td>173</td>
</tr>
<tr>
<td>Source Pane disclosure triangle</td>
<td>191</td>
</tr>
<tr>
<td>source relative includes</td>
<td>445</td>
</tr>
<tr>
<td>source trees</td>
<td></td>
</tr>
<tr>
<td>adding</td>
<td>366</td>
</tr>
<tr>
<td>changing</td>
<td>366</td>
</tr>
<tr>
<td>removing</td>
<td>367</td>
</tr>
<tr>
<td>Source Trees panel</td>
<td></td>
</tr>
<tr>
<td>options</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>365</td>
</tr>
<tr>
<td>Change</td>
<td>365</td>
</tr>
<tr>
<td>Choose</td>
<td>365</td>
</tr>
<tr>
<td>Name</td>
<td>365</td>
</tr>
<tr>
<td>Remove</td>
<td>365</td>
</tr>
<tr>
<td>Source Tree list</td>
<td>365</td>
</tr>
<tr>
<td>Type</td>
<td>365, 447</td>
</tr>
<tr>
<td>Source Trees preference panel</td>
<td>364</td>
</tr>
<tr>
<td>Source Trees settings panel</td>
<td>364</td>
</tr>
<tr>
<td>Absolute Path option</td>
<td>447</td>
</tr>
<tr>
<td>Environment Variable option</td>
<td></td>
</tr>
<tr>
<td>Registry Key option</td>
<td>447</td>
</tr>
<tr>
<td>Speak Message checkbox</td>
<td>216</td>
</tr>
<tr>
<td>special breakpoints</td>
<td></td>
</tr>
<tr>
<td>defined</td>
<td>228</td>
</tr>
<tr>
<td>purpose of</td>
<td>203</td>
</tr>
<tr>
<td>special breakpoints, disabling</td>
<td>228</td>
</tr>
<tr>
<td>special breakpoints, enabling</td>
<td>228</td>
</tr>
<tr>
<td>Speed slider</td>
<td>280</td>
</tr>
</tbody>
</table>
using the Files pane 255
using the Functions pane 255
Symbols Window menu command 530
symbols
shortcuts for editing 94
viewing all implementations 172
Symbols list
in Browser Contents window 170
Symbols pane 173
Symbols pop-up 170
Symbols window 171
Source pane 173
Symbols pane 173
toolbar 173
symbols window 172
Synchronize Modification Dates command 47
Synchronize Modification Dates menu command 530
Synchronize Modification Dates toolbar button 42
System Headers checkbox 123
System Paths list
Framework column 396
Recursive Search column 395
Search Status column 395
System Paths option
Access Paths panel 446
T
Tab indents selection option
Font & Tabs panel 446
Tab Inserts Spaces option
Font & Tabs panel 446
Tab Size option
Font & Tabs panel 446
tabs
Groups 206
In Files 120
In Folders 120
In Projects 120
In Symbolics 120
Instances 206
Templates 206
Target Address text box 277, 279
Target Breakpoint Can Cause Analyzer Trigger checkbox 288
Target column
in Files view of Project window 44
Target Configuration panel 263
Target list box 123
target management 48
Target Memory Buffer Address text box 264
Target Memory Buffer Size text box 265
Target Name option
Target Settings panel 447
Target Processor text/list box 264, 276
Target Scratch Memory End text box 282
Target Scratch Memory Start text box 282
target settings
Add 365, 395, 399, 402
Add Default 395
Always Search User Paths 394
Application 398
Apply button 391
Arguments 398
Auto-target Libraries 411
Cache subprojects 397
Cache symbolics between runs 411
Cancel button 391
Change 365, 395, 399, 402
Choose 365, 393
Clear 393
Compiler 401
Connection pop-up menu 412
Default language entry point 411
Details 403
Dump internal browse information after compile 398
Edit Language 402
Environment Settings 399
Export Panel button 391
Extension 401
Factory Settings button 390
Faster Execution Speed 403
File Mappings list 401
File Type 401
Flags 401
for IDE 389
Generate Browser Data From 398
Host Application for Libraries & Code Resources 399
Host Flags 395
Ignored By Make flag 401
Import Panel button 391
Initial directory 398
Interpret DOS and Unix Paths 395
Launchable flag 401
Linker 392
Log System Messages 411
Name 365
OK button 391
Optimization Level slider 403
Output Directory 393
Post-linker 392
Precompiled File flag 401
Pre-linker 392
Program Arguments 399
Program entry point 410
Remove 365, 395, 400, 402
Require Framework Style Includes 395
Resource File flag 401
Revert Panel button 390
Save button 391
Save project entries using relative paths 46, 393, 438
Small Code Size 403
Source Tree list 365
Source Trees 364
Stop at Watchpoints 411
Stop on application launch 410
Target Name 392
Target Settings Panels list 390
Target Settings command 531
Target Settings panel 55, 392
Update data every n seconds 411
Use External Debugger 398
Use modification date caching 397
User specified 411
Value 400
Variable 400
Working Directory 399
Choose 393
Clear 393
Linker 392, 433
Output Directory 393, 435
Post-linker 392, 435
Pre-linker 392, 436
Save project entries using relative paths 46, 393, 438
Target Name 392, 447
target settings panels
Access Paths 393
Analyzer Connections 285
Build Extras 396, 514
Debugger Settings 258, 410
File Mappings 400
Global Optimizations 402
Other Executables 407
Remote Debugging 411
Runtime Settings 398
Target Settings 392
Target Settings Panels list 390
Target Settings toolbar button 42
Target Settings window 389
Apply button 391
Cancel button 391
Export Panel button 391
Factory Settings button 390
Import Panel button 391
OK button 391
opening 391
Revert Panel button 390
Save button 391
Target Settings Panels list 390
targets 31
configuring 55
creating 53
files 48
managing 53
moving 50
removing 48, 53
renaming 50, 51, 54
setting default 54
strategies for 40
Targets page 47
Targets tab 55
Targets view 36, 50, 53

Tasks
- Activating automatic code completion 96
- Adding a constant to a variable 235
- Adding a keyword to a keyword set 406
- Adding an executable file 408
- Adding expressions (Expressions window) 234
- Adding markers to a source file 109
- Adding panes to an editor window 89
- Adding remote connections 386
- Adding source trees 366
- Adding subprojects to a project 39
- Alphabetizing Functions list pop-up order 106, 107
- Applying differences 141
- Arming a logic analyzer 289
- Connecting to a logic analyzer 289
- Creating a breakpoint template 212
- Creating a console application 78
- Creating a new class 160, 175, 176
- Creating a new data member 162, 183
- Creating a new member function 162, 180, 181
- Creating custom project stationery 38
- Creating empty projects 34
- Creating new projects from makefiles 33
- Creating new projects using project stationery 33
- Deactivating automatic code completion 97
- Deleting a breakpoint template 213
- Disabling a breakpoint 210
- Disabling a watchpoint 226
- Disabling an eventpoint 222
- Disarming a logic analyzer 289
- Disconnecting from a logic analyzer 289
- Docking a window by using a contextual menu 66
- Docking a window by using drag and drop 67
- Docking windows of the same kind 67
- Enabling a breakpoint 210, 223
- Enabling a watchpoint 226
- Examining items in the Folder Compare Results window 144
- Exporting projects to XML files 37
- Floating a window 69
- For managing files 57
- Generating project link maps 330
- Going to a particular line 107
- Closing a docked window 71
- Closing a workspace 75
- Closing projects 38
- Collapsing a docked window 71
- Collapsing browser panes 159
- Collapsing the editor window toolbar 86
- Completing code for data members 101
- Completing code for parameter lists 102
- Connecting to a logic analyzer 289
- Creating a breakpoint template 212
- Creating a console application 78
- Creating a new class 160, 175, 176
- Creating a new data member 162, 183
- Creating a new member function 162, 180, 181
- Creating custom project stationery 38
- Creating empty projects 34
- Creating new projects from makefiles 33
- Creating new projects using project stationery 33
- Deactivating automatic code completion 97
- Deleting a breakpoint template 213
- Disabling a breakpoint 210
- Disabling a watchpoint 226
- Disabling an eventpoint 222
- Disarming a logic analyzer 289
- Disconnecting from a logic analyzer 289
- Docking a window by using a contextual menu 66
- Docking a window by using drag and drop 67
- Docking windows of the same kind 67
- Enabling a breakpoint 210, 223
- Enabling a watchpoint 226
- Examining items in the Folder Compare Results window 144
- Exporting projects to XML files 37
- Floating a window 69
- For managing files 57
- Generating project link maps 330
- Going to a particular line 107
hiding the classes pane 160
importing projects saved as XML files 37
indenting text blocks 93
inserting a reference template 111
issuing command lines 293
killing program execution 194
looking up symbol definitions 111
making a summation of two variables 235
making a window an MDI child 70
manipulating variable formats 232
moving a docked window 71
navigating browser data 150
navigating Code Completion window 100
navigating to a marker 109
opening a recent workspace 76
opening a single-class hierarchical window 169
opening a workspace 74
opening an Array window 243
opening projects 35
opening projects created on other hosts 35
opening registers in a separate Registers window 247
opening subprojects 40
opening the Breakpoints window 207
opening the Cache window 290
opening the Command window 292
opening the Expressions window 234
opening the Flash Programmer window 261
opening the Global Variables window 230
opening the Hardware Diagnostics window 274
opening the Log window 259
opening the Processes window 256, 257
opening the Profile window 291
opening the Registers window 245
opening the Symbolics window 255
opening the symbols window 172
opening the Target Settings window 391
opening the Trace window 290
overstriking text (Windows) 92
printing class hierarchies 167
printing projects 36
removing a keyword from a keyword set 407
removing a marker from a source file 109
removing all markers from a source file 110
removing an executable file 409
removing panes from an editor window 89
removing remote connections 387
removing source trees 367
replacing text in a single file 117
resizing panes in an editor window 89
restarting the debugger 195
resuming program execution 194
running a program 195
saving a copy of a workspace 75
saving a workspace 74
saving projects 35
saving the contents of the Breakpoints window 207
searching a single file 115
searching for text across multiple files 127
searching for text across multiple folders 121
searching for text across multiple projects 123
searching for text across multiple symbolics files 125
searching with a text selection 132
selecting entire routines 92
selecting item in Code Completion window 101
selecting lines 92
selecting multiple lines 92
selecting rectangular portions of lines 92
selecting text in editor windows 91
setting a breakpoint 207
setting a conditional breakpoint 211
setting a conditional eventpoint 223
setting a conditional watchpoint 227
setting a Log Point 216
setting a Pause Point 217
setting a Script Point 218
setting a Skip Point 219
setting a Sound Point 220
setting a temporary breakpoint 211
setting a Trace Collection Off eventpoint 221
setting a Trace Collection On eventpoint 222
setting a watchpoint 225
showing the classes pane 160
sorting the classes list 161
specifying the default breakpoint template 214
starting the debugger 192
stepping into a routine 193
stepping over a routine 193
stopping program execution 194
suppressing dockable windows 70
toggling automatic punctuation balancing 95
toggling the symbol hint 195
triggering code completion by keyboard 97
triggering code completion from IDE menu bar 97
unapplying file differences 141
undocking a window 68
unfloating a window 69
unindenting text blocks 93
updating data from a logic analyzer 289
using an external editor on the Macintosh 360
using contextual menus 197
using the default workspace 73
using the document settings pop-up 87
using the Executables pane in the Symbolics window 255, 256
using the Files pane in the Symbolics window 255
using the Find Next command 130
using the Find Previous command 131
using the Functions list pop-up 106
using the Functions pane in the Symbolics window 255
using the Interfaces list pop-up 106
using the symbol hint 196
using the VCS pop-up 87
using virtual space 93
viewing a file path 45
viewing breakpoint properties 208
viewing browser data by contents 170
viewing browser data by inheritance 166
viewing class data from hierarchy windows 158
viewing eventpoint properties 222
viewing global variables for different processes 230
viewing registers 245
viewing watchpoint properties 226
template, default for breakpoints 212
template, for breakpoints 212
Templates option 378
Templates tab 206
templates, creating for breakpoints 212
templates, deleting for breakpoints 213
templates, reference (Macintosh) 111
templates, specifying the default for breakpoints 214
temporary breakpoint, defined 211
temporary breakpoints 204
setting 211
text
changing a find string 131
find by selecting 130
finding 113
overstriking (Windows) 92
replacing 113
searching with a selection 132
text blocks, indenting 93
text blocks, unindenting 93
text boxes
Address 248
Analyzer Configuration File text box 287
Analyzer Slot 287
Bit Value 249
Description File 248, 250
End 269, 281
Flash Memory Base Address 266
Host Name 287
Message 216
Name 287
Offset 269
Passes 282
Result Count 129
Search Criteria 129
Size 273
Start 269, 273, 281
Target Address 277, 279
Target Memory Buffer Address 264
Target Memory Buffer Size 265
Target Scratch Memory End 282
Target Scratch Memory Start 282
Trace Support File 288
Use Selected File 268
Use Target Initialization 264, 276
Value to Write 278, 279

Text Colors panel
options
Activate Browser Coloring 428
Activate Syntax Coloring 428, 431, 444
Foreground 428
Keywords 431
Strings 444

Text Colors preference panel
options
Activate Browser Coloring 377
Activate Syntax Coloring 377
Background 377
Classes 377
Comments 377
Constants 377
Edit 377
Enums 377
Foreground 377
Functions 377
Globals 377
Keywords 377
Macros 378
Other 378
Set 1, Set 2, Set 3, Set 4 377
Strings 377
Templates 378
TypeDefs 378

Text editing area, in editor window 88
Text View list box 250, 251
Auto 251
Text View pop-up menu

Bitfield Description option 251
Register Description option 251
Register Details option 251

text/list boxes
By Type 121
Find 114, 116, 119
Replace With 116, 119
Search in 121
Target Processor 264, 276
text-selection Find 130
THINK Reference 110, 111, 428
third-party editor support 450
third-party text editors
Emacs 432

Thread window
Breakpoints button 190
current-statement arrow 191
dash 191
debut button 190
Expressions button 190
Functions list box 192
Kill button 190
Line And Column button 192
Pane Collapse box 190
Pane Expand box 190
Pane resize bar 191
Resume button 190
run button 190
Source File button 191
Source list box 192
Source pane 191
Source Pane disclosure triangle 191
Stack pane 191
Step Into button 190
Step Out button 190
Step Over button 190
Stop button 190
Symbolsics button 190
Variables pane 191
Variables Pane Listing button 191

thread window 188
thread, breakpoint property 209
threading in IDE 420

IDE 5.7 User's Guide 593
__throw() 505
ticksTimeBase 305, 310, 317
Tile Editor Windows command 531
Tile Editor Windows Vertically command 531
Tile Horizontally command 531
Tile Vertically command 532
time hogs, finding 313
timebase 305, 317
timeMgrTimeBase 305, 317
times hit, breakpoint property 209
times left, breakpoint property 209
To Smallest Height command in Resize submenu 534
To Smallest Width command in Resize submenu 534
toggling
symbol hint 195
toolbar
collapsing in editor window 86
expanding in editor window 86
Toolbar (Editor Window) Elements
Document Settings 345
File Dirty Indicator 345
File Path field 345
Functions 345
Header Files 345
Markers 345
Version Control Menus 345
toolbar buttons
Browser Contents 154
Class Hierarchy 154
Go Back 154
Go Forward 154
Make 42
Single Class Hierarchy Window 154
Synchronize Modification Dates 42
Target Settings 42
Toolbar Items 333, 344
Toolbar submenu
Anchor Floating Toolbar command 504
Clear Floating Toolbar command 507
Clear Main Toolbar command 507
Clear Window Toolbar command 508
Hide Floating Toolbar command 517
Hide Main Toolbar command 517
Reset Window Toolbar command 46, 525
Show Floating Toolbar command 517, 529
Show Main Toolbar command 517
Show Window Toolbar command 529
Toolbars
Add element 344
Clear Elements 345
Customize 342
Elements 342, 343
Icons 344
Instances of 343
Main (floating) 343
Modify 344
Project and Window 343
Remove single element 344
Toolbar Items tab 344
Types 343
toolbars
editor 86
for Symbols window 173
resetting 346
tools
browser 25
build system 26
debugger 26
editor 25
project manager 25
search engine 25
Tools menu 500
icon 500
tools, for hardware 261
ToolServer menu 450
ToolServer Worksheet command 533
ToolTip 344
touch
defined 44
Touch column 51, 52
in Files view of Project window 44
Touch command 51
touching
all files 51
all groups 51
files 51
groups 51
trace
  working with logic analyzer 285
Trace Collection Off 221
Trace Collection Off eventpoint 215
Trace Collection Off eventpoint, clearing 221
Trace Collection Off eventpoint, setting 221
Trace Collection On 221
Trace Collection On eventpoint 215
Trace Collection On eventpoint, clearing 222
Trace Collection On eventpoint, setting 222
Trace Support File text box 288
Trace window 290
  opening 290
Treat as Expression checkbox 217
triggering
  code completion by keyboard 97
  code completion from IDE menu bar 97
turning off
  symbol hint 195
turning on
  symbol hint 195
Type list box
  Absolute Path option 447
  Type option
    Source Trees panel 447
Type pop-up menu
  Environment Variable option 447
  Registry Key option 447
type, breakpoint property 209
TypeDefs option 378
types
  of documentation 19

U
Unanchor Floating Toolbar command 534
Unapply button 140
Unapply Difference command 141, 534
unbounded arrays, setting default size for
  viewing 422
Uncaught Exceptions Only command 534
Undo button 141
Undo command 534
undocking windows 68
unfloating windows 69
  Ungroup command 534
  unindenting text blocks 93
Untouch command 52
untouching
  a file 52
  a group 52
  all files 52
  all groups 52
Update Data command 289
Update Data Every n Seconds option 447
Use Concurrent Compiles option 437, 447
Use Custom Settings checkbox 264, 276
Use Debugging Monitor option 447
Use Default Workspace option 448
Use External Debugger option 448
Use External Editor option 448
Use Local Project Data Storage option 448
Use modification date caching option 449
Use Multiple Document Interface option 63, 449
  turning off 359
  turning on 359
Use multiple undo option 534
  in Editor Settings panel 449
Use Script menu option 449
Use Scripts Menu option 501
Use Selected File checkbox 268
Use Selected File text box 268
Use Target CPU checkbox 282
Use Target Initialization checkbox 264, 276
Use Target Initialization text box 264, 276
Use Third Party Editor option 450
Use ToolServer Menu option 500
Use ToolServer menu option
  IDE Extras panel 450
User Paths list
  Framework column 396
  Recursive Search column 395
  Search Status column 395
User Paths option 450
User Specified option 450
User specified option 411
using
  document settings pop-up 87
Executables pane in the Symbolics window 255, 256
Files pane in the Symbolics window 255
Find Next command 130
Find Previous command 131
Functions list pop-up 106
Functions pane in the Symbolics window 255
Interfaces list pop-up 106
logic analyzer 288
Register Details window 250
symbol hint 196
VCS pop-up 87
virtual space 93

V
Value to Write text box 278, 279
variable formatting 232
Variable Values Change option
  Display Settings panel 451
Variable window 231
variables
  ^var placeholder 233
  adding a constant to 235
  making a summation of 235
  manipulating formats 232
  symbol hint 195
Variables pane 191
Variables Pane Listing button 191
variables, working with 229
VCS 87
  list pop-up 156
  menu 500
  pop-up 87
VCS Commands option
  Editor Settings panel 451
VCS menu 163, 451
  icon 500
VCS pop-up
  using 87
Vectorization 405
Verify button 270
version control 451, 500
Version Control Settings command 535

VCS
Version Control System. See VCS.
Vertical Center command in Align submenu 526,
  532, 533, 535
View Array menu command 535
View as implementor 156
View as subclass 156
View As Unsigned Decimal menu command 535,
  536
View as user 156
View Disassembly menu command 537
view in profiler
  class 312
  detail 311
  flat 311
View Memory As command 537
View Memory command 537
View Mixed menu command 537
View Source menu command 537
View Target Memory Writes checkbox 265
View Variable menu command 537
viewing
  all symbol implementations 172
  breakpoints 207
  browser data by contents 170
  browser data by inheritance 166
  file paths 45
  register details 247
  registers 245
viewing access breakpoint 226
viewing installed plug-ins 503
viewing installed products 503
virtual
  icon for 161
  virtual space, using 93

W
Walking 1’s checkbox 281
Walking Ones subtest 283
Walking Ones test
  Address Line fault 282
  Data Line fault 283
  Retention fault 283
subtests
  Ones Retention 283
<table>
<thead>
<tr>
<th>Window Types</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>docked</td>
<td>Hide non-debugging windows 429</td>
</tr>
<tr>
<td>floating</td>
<td>Minimize non-debugging windows 434</td>
</tr>
<tr>
<td>MDI child</td>
<td>Monitor for debugging 434</td>
</tr>
<tr>
<td></td>
<td>Move open windows to debugging monitor when</td>
</tr>
<tr>
<td></td>
<td>debugging starts 434</td>
</tr>
<tr>
<td></td>
<td>Open windows on debugging monitor during</td>
</tr>
<tr>
<td></td>
<td>debugging 435</td>
</tr>
<tr>
<td></td>
<td>Use Debugging Monitor 447</td>
</tr>
</tbody>
</table>

**Windows**
- creating files 57

**Windows**
- Array 242
  - Browser Contents 169
  - Cache 290
  - Class Browser 153
  - Code Completion 98
  - Command 292
  - Compare Files Setup 136
  - Customize IDE Commands 347
  - dock bars in dockable windows 70
dockable 63
  - dockable, about 63
dockable, turning off 70
dockable, working with 66
docking the same kind of 67
docking with a contextual menu 66
docking with drag and drop 67
editor 83
editor, other 88
Expressions 233
File Compare Results 139
Find (single-file) 113
Find and Replace (multiple-file) 118
Find and Replace (single-file) 115
Flash Programmer 261
floating 69
Folder Compare Results 142
Global Variables 229
Hardware Diagnostics 274
hierarchy 166
IDE Preferences 226, 353
Log 258
making MDI children of 70
Memory 237
New C++ Class 177
New C++ Data Member 184
New C++ Member Function 182
Processes 256
project window 41
Registers 244
remembering size and position of 527
Remove Markers 108
saving default size and position of 527
Search results 128
Symbolics 253
Target Settings 389
Trace 290
undocking 68
unfloating 69
variable 231
Windows menu layout 473
WinHelp (Windows) 110
Wizards
Browser 175
wizards
New Class 160, 175, 176
New Data Member 162, 183
New Member Function 180, 181
New Member Functions 162
Word option button 277, 279, 282
working with IDE preferences 353
with IDE target settings 389
Working Directory option
Runtime Settings panel 452
working with 66
browser 147
class browser windows 153
class hierarchy windows 165
IDE hardware tools 261
logic analyzer 285, 288
working with breakpoint templates 212
working with breakpoints 207
working with debugger data 253
working with dockable windows 66
working with eventpoints 222
working with files 57
working with memory 237
working with projects 29
working with variables 229
workspace, defined 73
workspaces 73
closing 75
opening 74
opening recent 76
saving 74
saving copies of 75
using default 73
Workspaces option
IDE Extras panel 452
workspaces, about 73
Write button 250
Write option button 277, 279

X
XML
exporting projects 37
importing projects 37
Register Details Window
BFVALUE 460
BITFIELD 457
file locations 461
REGISTER 455
sample files 462
Register Details window
specification 453

Z
Zeros Retention subtest 283
Zoom Window menu command 537
Zoom windows to full screen option
IDE Extras panel 452