Freescale, the Freescale logo, CodeWarrior, ColdFire, PowerQUICC, and StarCore are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. QorIQ Qonverge, QUICC Engine are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2008-2013 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>6501 William Cannon Drive West</td>
</tr>
<tr>
<td></td>
<td>Austin, Texas 78735</td>
</tr>
<tr>
<td></td>
<td>U.S.A.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>World Wide Web</th>
<th><a href="http://www.freescale.com/codewarrior">http://www.freescale.com/codewarrior</a></th>
</tr>
</thead>
</table>

| Technical Support      | http://www.freescale.com/support     |
# Table of Contents

## 1 Introduction

1.1 Release Notes .................................................. 1  
1.2 Documentation Structure ....................................... 1  
1.2.1 Documentation Formats .................................... 1  
1.2.2 Documentation Types ....................................... 2  
1.3 Manual Conventions ........................................... 2  
1.3.1 Figure Conventions ....................................... 2  
1.3.2 Keyboard Conventions .................................... 2  
1.4 CodeWarrior IDE Overview ..................................... 3  
1.4.1 Development Cycle ....................................... 3  
1.4.2 CodeWarrior IDE Advantages .............................. 5  

## 2 IDE Extensions

2.1 CodeWarrior Projects View ................................... 7  
2.1.1 Active Configuration ...................................... 8  
2.1.2 Tree and List View ....................................... 9  
2.1.3 Column Headers .......................................... 10  
2.1.4 Quick Search ........................................... 11  
2.1.5 Filtering ................................................ 12  
2.2 Command Line Interface ....................................... 13  
build .......................................................... 14  
getOptions .................................................... 14  
generateMakefiles ............................................ 15  
references .................................................... 16  
setOptions .................................................... 17  
updateWorkspace .............................................. 18  
2.3 Commander View ............................................. 18  
2.3.1 Command Groups ......................................... 20  
2.3.2 Customizing Commander View ............................ 21  
2.3.3 Pinning Commander View ................................. 22  
2.4 Concurrent Compilation ....................................... 23  
2.5 Console View .............................................. 24
# Table of Contents

2.6 Context Menus .............................................. 25
2.7 Diagnostic Information Export ................................. 26
  2.7.1 General Settings for Diagnostic Information .............. 26
  2.7.2 Export Diagnostic Information ........................... 28
2.8 Extracting CodeWarrior Configuration Details ................ 32
2.9 Find and Open File ......................................... 33
2.10 Importing Files ............................................ 34
  2.10.1 CodeWarrior Drag and Drop Support ...................... 34
  2.10.2 Using the Import Wizard ................................ 35
    2.10.2.1 Import Existing Project ............................ 35
    2.10.2.2 Import Example Project ............................ 38
2.11 Key Mappings ............................................. 40
2.12 Linker Command File Navigation ............................. 42
2.13 Multiple Compiler Support ................................ 43
2.14 New External File ......................................... 45
2.15 Problems View ............................................ 46
2.16 Referenced Projects ....................................... 47
  2.16.1 Create a Referenced Project ............................ 47
  2.16.2 Displaying Referenced Projects in CodeWarrior Project View ............ 49
  2.16.3 Automatic Linking with Referenced Project Build Artifact .............. 51
  2.16.4 Circular Build Dependencies ............................ 51
2.17 Target Management via Remote System Explorer ............... 52
  2.17.1 Creating Remote System ............................... 52
  2.17.2 Creating Hardware or Simulator Connection Configuration .......... 55
  2.17.3 Creating Hardware or Simulator Target Configuration ................ 56
  2.17.4 Creating TRK Target Configuration .......................... 60
  2.17.5 Remote Systems View ................................ 61
    2.17.5.1 Modifying Target or Connection Configuration ............... 61
    2.17.5.2 Exporting the Target or Connection Configuration .............. 62
    2.17.5.3 Importing a Target or Connection Configuration ............... 63
    2.17.5.4 Apply to Project ................................. 63
    2.17.5.5 Apply to Connection ............................... 64
    2.17.5.6 Automatic Removal of Unreferenced Remote Systems .............. 65
  2.17.6 Automatic Project Remote System Setting Cache .............. 66
    2.17.6.1 Remote System Missing ............................ 67
# Table of Contents

2.17.6.2 Remote System Changed ............................................ 69
2.17.7 Compatibility with Older Products .................................... 70
  2.17.7.1 Display of Launch Configurations Needing Migration .......... 70
  2.17.7.2 Migrating Launch Configurations ................................. 72
2.18 Target Processor Selection ............................................. 79
2.19 Viewing CodeWarrior Plug-ins ......................................... 79

3 Debugger ........................................................................... 83
  3.1 About Debugger ............................................................. 84
  3.2 Breakpoints ................................................................. 84
    3.2.1 Breakpoints View ..................................................... 85
    3.2.2 Breakpoint Annotations ............................................. 86
    3.2.3 Regular Breakpoints .................................................. 86
      3.2.3.1 Setting Line Breakpoint ....................................... 87
      3.2.3.2 Setting Method Breakpoint .................................... 88
    3.2.4 Special Breakpoints .................................................. 88
      3.2.4.1 Setting Special Breakpoint .................................... 89
    3.2.5 Breakpoint Persistence .............................................. 89
    3.2.6 Working with Breakpoints ......................................... 90
      3.2.6.1 Modify Breakpoint Properties ................................ 90
      3.2.6.2 Restricting Breakpoints to Selected Targets and Threads ... 92
      3.2.6.3 Limiting New Breakpoints to Active Debug Context ........ 93
      3.2.6.4 Grouping Breakpoints ........................................... 94
      3.2.6.5 Disabling Breakpoints .......................................... 95
      3.2.6.6 Enabling Breakpoints .......................................... 95
      3.2.6.7 Removing Breakpoints ......................................... 95
      3.2.6.8 Removing All Breakpoints ....................................... 96
      3.2.6.9 Undo Delete Breakpoint ........................................ 96
      3.2.6.10 Redo Delete Breakpoint ....................................... 96
      3.2.6.11 Skipping All Breakpoints ...................................... 96
    3.2.7 Breakpoint Actions .................................................. 96
      3.2.7.1 Breakpoint Actions Preferences Page ......................... 97
      3.2.7.2 Adding Breakpoint Action ..................................... 98
      3.2.7.3 Attaching Breakpoint Actions to Breakpoints ............... 99
    3.2.8 Selecting Breakpoint Template .................................... 100

*CodeWarrior Common Features Guide*
# Table of Contents

3.3 Build While Debugging .................................................. 101
3.4 Cache View .............................................................. 102
  3.4.1 Opening Cache View ............................................. 103
  3.4.2 Preserving Sorting ................................................ 104
  3.4.3 Cache View Drop-down Menu .................................... 105
3.5 CodeWarrior Debugger Settings ..................................... 107
  3.5.1 Modifying Debugger Settings .................................. 107
  3.5.2 Reverting Debugger Settings .................................. 108
  3.5.3 Stopping Debugger at Program Entry Point .................. 108
3.6 Core Index Indicators in Homogeneous Multicore Environment ... 109
  3.6.1 System Browser View .......................................... 109
    3.6.1.1 Kernel Awareness ....................................... 110
    3.6.1.2 OS Application ........................................ 110
  3.6.2 Console View ..................................................... 111
3.7 Debug Perspective .................................................... 111
3.8 Debug View ............................................................ 112
  3.8.1 Common Debugging Actions ..................................... 113
    3.8.1.1 Starting the Debugger .................................. 114
    3.8.1.2 Stepping Into Routine Call ............................ 114
    3.8.1.3 Stepping Out of Routine Call .......................... 114
    3.8.1.4 Stepping Over Routine Call .............................. 115
    3.8.1.5 Stopping Program Execution ............................ 115
    3.8.1.6 Resuming Program Execution ............................ 115
    3.8.1.7 Running a Program ...................................... 115
    3.8.1.8 Disconnecting a Core ................................... 116
    3.8.1.9 Restarting Debugger .................................... 116
    3.8.1.10 Debugging in Instruction Stepping Mode .............. 116
    3.8.1.11 Changing the Program Counter Value .................. 117
3.9 Disassembly View ..................................................... 117
3.10 Environment Variables in Launch Configuration .................. 118
3.11 Flash Programmer .................................................... 119
  3.11.1 Create a Flash Programmer Target Task ...................... 120
  3.11.2 Configure the Flash Programmer Target Task ............... 122
    3.11.2.1 Add a Flash Device .................................. 123
    3.11.2.2 Specify Target RAM Settings .......................... 124
Table of Contents

- 3.11.2.3 Add Flash Programmer Actions ........................................ 125
- 3.11.3 Run Flash Programmer Target Task .................................... 134
- 3.12 Flash File to Target ............................................................ 135
- 3.12.1 Erasing Flash Device ...................................................... 137
- 3.12.2 Programming a File ......................................................... 137
- 3.13 Hardware Diagnostics ........................................................ 138
- 3.13.1 Creating Hardware Diagnostics Task ................................ 138
- 3.13.2 Working with Hardware Diagnostic Action Editor ............... 139
- 3.13.2.1 Action Type ............................................................... 140
- 3.13.2.2 Memory Access .......................................................... 141
- 3.13.2.3 Loop Speed ............................................................... 141
- 3.13.2.4 Memory Tests ............................................................ 142
- 3.13.3 Memory Test Use Cases .................................................. 146
- 3.13.3.1 Use Case 1: Run Host based Scope Loop on the Target .... 146
- 3.13.3.2 Use Case 2: Run Target based Memory Tests on the Target .. 147
- 3.14 Import/Export/Fill Memory .................................................. 147
- 3.14.1 Creating a Task for Import/Export/Fill Memory .................... 147
- 3.14.2 Importing Data from a File into Memory ............................ 149
- 3.14.3 Exporting Memory Contents to a File ............................... 151
- 3.14.4 Fill Memory with a Data Pattern ...................................... 153
- 3.15 Launch Group ................................................................. 155
- 3.15.1 Creating a Launch Group ................................................ 155
- 3.15.2 Launching the Launch Group ........................................... 159
- 3.16 Load Multiple Binaries ....................................................... 159
- 3.16.1 Viewing Binaries ............................................................ 161
- 3.17 Memory View ................................................................. 162
- 3.17.1 Opening Memory View .................................................... 162
- 3.17.2 Adding Memory Monitor ................................................ 163
- 3.17.3 Adding Memory Renderings ............................................. 164
- 3.17.4 Mixed Source Rendering ............................................... 165
- 3.17.5 Setting Memory Access Size .......................................... 166
- 3.17.6 Exporting Memory ........................................................ 167
- 3.17.7 Importing Memory ........................................................ 168
- 3.17.8 Setting Watchpoint in Memory View ................................. 169
- 3.17.9 Clearing Watchpoints from the Memory View .................... 170
# Table of Contents

3.18 Memory Browser View ............................................. 170
3.19 Memory Management Unit Configurator .......................... 171
  3.19.1 Creating an MMU Configuration .............................. 172
  3.19.2 Saving MMU Configurator Settings ............................ 175
  3.19.3 MMU Configurator Toolbar ................................... 175
  3.19.4 MMU Configurator Pages ..................................... 176
    3.19.4.1 General Page ......................................... 176
    3.19.4.2 Program MATT Page .................................... 178
    3.19.4.3 Data MATT Page ....................................... 182
  3.19.4.4 Saving MMU Configurator Generated Code. ............... 186
  3.19.5 Opening the MMU Configurator View ......................... 188
3.20 Multicore Debugging ............................................. 189
  3.20.1 Multicore Suspend ........................................... 190
  3.20.2 Multicore Resume ............................................ 190
  3.20.3 Multicore Terminate ......................................... 191
  3.20.4 Multicore Restart ............................................ 191
3.21 Multicore Groups .................................................. 191
  3.21.1 Creating a Multicore Group .................................. 192
  3.21.2 Modifying a Multicore Group ................................ 195
  3.21.3 Editing a Multicore Group ................................... 195
  3.21.4 Using Multicore Group Debugging Commands ................. 197
  3.21.5 Multicore Breakpoint Halt Groups. .......................... 199
3.22 Multicore Reset .................................................... 200
  3.22.1 On Demand Reset ............................................. 202
3.23 Path Mappings ..................................................... 202
  3.23.1 Automatic Path Mappings .................................... 203
  3.23.2 Manual Path Mappings ....................................... 205
    3.23.2.1 Adding Path Mapping to Workspace .................... 208
3.24 Redirecting Standard Output Streams to Socket .................. 209
3.25 Refreshing Data During Run Time ................................ 211
3.26 Registers View ................................................... 212
  3.26.1 Displaying the Registers View ............................... 213
  3.26.2 Viewing Registers .......................................... 214
  3.26.3 Changing Register Values ................................... 214
  3.26.4 Exporting Registers ......................................... 215
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.26.5</td>
<td>216</td>
</tr>
<tr>
<td>3.26.6</td>
<td>217</td>
</tr>
<tr>
<td>3.27</td>
<td>217</td>
</tr>
<tr>
<td>3.27.1</td>
<td>219</td>
</tr>
<tr>
<td>3.27.2</td>
<td>220</td>
</tr>
<tr>
<td>3.28</td>
<td>221</td>
</tr>
<tr>
<td>3.28.1</td>
<td>222</td>
</tr>
<tr>
<td>3.29</td>
<td>223</td>
</tr>
<tr>
<td>3.29.1</td>
<td>224</td>
</tr>
<tr>
<td>3.29.2</td>
<td>224</td>
</tr>
<tr>
<td>3.30</td>
<td>226</td>
</tr>
<tr>
<td>3.31</td>
<td>227</td>
</tr>
<tr>
<td>3.31.1</td>
<td>227</td>
</tr>
<tr>
<td>3.32</td>
<td>229</td>
</tr>
<tr>
<td>3.33</td>
<td>230</td>
</tr>
<tr>
<td>3.33.1</td>
<td>230</td>
</tr>
<tr>
<td>3.34</td>
<td>232</td>
</tr>
<tr>
<td>3.34.1</td>
<td>232</td>
</tr>
<tr>
<td>3.34.2</td>
<td>233</td>
</tr>
<tr>
<td>3.35</td>
<td>233</td>
</tr>
<tr>
<td>3.35.1</td>
<td>233</td>
</tr>
<tr>
<td>3.35.2</td>
<td>234</td>
</tr>
<tr>
<td>3.35.3</td>
<td>235</td>
</tr>
<tr>
<td>3.35.3.1</td>
<td>235</td>
</tr>
<tr>
<td>3.35.4</td>
<td>236</td>
</tr>
<tr>
<td>3.35.5</td>
<td>237</td>
</tr>
<tr>
<td>3.36</td>
<td>238</td>
</tr>
<tr>
<td>3.36.1</td>
<td>239</td>
</tr>
<tr>
<td>3.36.2</td>
<td>240</td>
</tr>
<tr>
<td>3.36.3</td>
<td>241</td>
</tr>
<tr>
<td>3.36.4</td>
<td>242</td>
</tr>
<tr>
<td>3.36.5</td>
<td>243</td>
</tr>
<tr>
<td>3.36.6</td>
<td>243</td>
</tr>
<tr>
<td>3.36.7</td>
<td>244</td>
</tr>
<tr>
<td>3.36.8</td>
<td>244</td>
</tr>
</tbody>
</table>
# Table of Contents

## 4 Debugger Shell

4.1 Executing Previously Issued Commands ........................................ 246
4.2 Using Code Hints ........................................................................ 247
4.3 Using Auto-Completion ................................................................ 247
4.4 Command-Line Debugger Shell ......................................................... 248
  4.4.1 Setting Hardware Breakpoints .................................................. 248
4.5 Debugger Shell Commands ............................................................... 248
  - about ......................................................................................... 248
  - alias ......................................................................................... 249
  - bp ............................................................................................. 249
  - cd ............................................................................................. 251
  - change ....................................................................................... 251
  - cls ............................................................................................. 255
  - cmdwin::ca ............................................................................... 255
  - cmdwin::caln ........................................................................... 256
  - config ....................................................................................... 257
  - copy ........................................................................................ 264
  - debug ...................................................................................... 264
  - dir ............................................................................................ 265
  - disassemble ............................................................................... 266
  - display ..................................................................................... 267
  - evaluate .................................................................................... 272
  - finish ....................................................................................... 273
  - fl::blankcheck ......................................................................... 274
  - fl::checksum ......................................................................... 274
  - fl::device ............................................................................... 275
  - fl::diagnose ............................................................................ 276
  - fl::disconnect ......................................................................... 277
  - fl::dump .................................................................................. 277
  - fl::erase ................................................................................. 278
  - fl::image ............................................................................... 279
  - fl::protect .............................................................................. 280
  - fl::secure ............................................................................... 280
  - fl::target ............................................................................... 281
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl::verify</td>
<td>282</td>
</tr>
<tr>
<td>fl::write</td>
<td>282</td>
</tr>
<tr>
<td>funcs</td>
<td>283</td>
</tr>
<tr>
<td>getpid</td>
<td>283</td>
</tr>
<tr>
<td>go</td>
<td>283</td>
</tr>
<tr>
<td>help</td>
<td>284</td>
</tr>
<tr>
<td>history</td>
<td>285</td>
</tr>
<tr>
<td>jtagclock</td>
<td>285</td>
</tr>
<tr>
<td>kill</td>
<td>286</td>
</tr>
<tr>
<td>launch</td>
<td>286</td>
</tr>
<tr>
<td>linux::displaylinuxlist</td>
<td>286</td>
</tr>
<tr>
<td>linux::loadsymbols</td>
<td>288</td>
</tr>
<tr>
<td>linux::refreshmodules</td>
<td>288</td>
</tr>
<tr>
<td>linux::selectmodule</td>
<td>288</td>
</tr>
<tr>
<td>linux::unloadsymbols</td>
<td>289</td>
</tr>
<tr>
<td>loadsym</td>
<td>289</td>
</tr>
<tr>
<td>log</td>
<td>289</td>
</tr>
<tr>
<td>mc::config</td>
<td>290</td>
</tr>
<tr>
<td>mc::go</td>
<td>291</td>
</tr>
<tr>
<td>mc::group</td>
<td>291</td>
</tr>
<tr>
<td>mc::kill</td>
<td>292</td>
</tr>
<tr>
<td>mc::reset</td>
<td>293</td>
</tr>
<tr>
<td>mc::restart</td>
<td>293</td>
</tr>
<tr>
<td>mc::stop</td>
<td>293</td>
</tr>
<tr>
<td>mc::type</td>
<td>294</td>
</tr>
<tr>
<td>mem</td>
<td>294</td>
</tr>
<tr>
<td>next</td>
<td>297</td>
</tr>
<tr>
<td>nexti</td>
<td>298</td>
</tr>
<tr>
<td>oneframe</td>
<td>298</td>
</tr>
<tr>
<td>pwd</td>
<td>299</td>
</tr>
<tr>
<td>quitIDE</td>
<td>299</td>
</tr>
<tr>
<td>radix</td>
<td>299</td>
</tr>
<tr>
<td>redirect</td>
<td>300</td>
</tr>
<tr>
<td>refresh</td>
<td>302</td>
</tr>
<tr>
<td>reg</td>
<td>302</td>
</tr>
</tbody>
</table>
## Table of Contents

- reset ................................................................. 306
- restart ............................................................ 306
- restore .............................................................. 307
- run ................................................................. 308
- save ................................................................. 309
- setpc ............................................................... 310
- setpicloadaddr .................................................. 312
- stack ............................................................... 312
- status ............................................................... 313
- step ................................................................. 313
- stepti ............................................................... 314
- stop ................................................................. 315
- switchtarget ..................................................... 315
- var ................................................................. 317
- wait ................................................................. 319
- watchpoint ....................................................... 319

5  **Debugger Script Migration**  321

  5.1 Command-Line Syntax ........................................... 321
  5.2 Launching a Debug Session .................................... 322
  5.3 Stepping ....................................................... 323
  5.4 Config Settings ................................................ 324

**Index**  325
Introduction

This manual describes the CodeWarrior IDE and debugger features that are common across all the CodeWarrior products. This chapter presents an overview of the manual and the CodeWarrior IDE.

This chapter explains:

- 1.1 Release Notes
- 1.2 Documentation Structure
- 1.3 Manual Conventions
- 1.4 CodeWarrior IDE Overview

NOTE The CodeWarrior Common Features Guide may describe features that are not available for your product. Further, the figures show a typical user interface, which may differ slightly from your CodeWarrior product. Refer your product’s Targeting Manual for details of its product-specific features.

1.1 Release Notes

Before using the CodeWarrior IDE, read the release notes. These notes contain important information about the last-minute changes, bug fixes, incompatible elements, or other sections that may not be included in this manual.

NOTE The release notes for specific components of the CodeWarrior IDE are located in the Release_Notes folder in the CodeWarrior installation directory.

1.2 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.

1.2.1 Documentation Formats

CodeWarrior documentation presents information in the following formats:
1.2.2 Documentation Types

Each CodeWarrior manual focuses on a particular information type:

- **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** — Provide specific information required to create software that operates on a particular platform or microprocessor. Examples include the Targeting Windows manuals.
- **Reference manuals** — Provide specialized information about coding libraries, programming languages, and IDE. Examples include the C Compiler Reference.
- **Core manuals** — Core manuals explain the core technologies available in the CodeWarrior IDE. Examples include CodeWarrior Common Features Guide.

1.3 Manual Conventions

This topic explains conventions in the CodeWarrior Common Features Guide.

1.3.1 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 labeled images identify the specific host.

1.3.2 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.

At any time, you can obtain a list of available key bindings using Key Assist (Help > Key Assist or Ctrl+Shift+L).
1.4 CodeWarrior IDE Overview

The CodeWarrior IDE provides an efficient and flexible software-development tool suite. This topic explains the software development cycle and the advantages of using the CodeWarrior IDE for development.

This topic explains:

- 1.4.1 Development Cycle
- 1.4.2 CodeWarrior IDE Advantages

1.4.1 Development Cycle

A software developer follows a general process to develop a project:

1. Begin with an idea for a new software.
2. Implement the new idea in source code.
3. Compile the source code into machine code.
4. Link the machine code and create an executable file.
5. Correct errors (debug).
6. Compile, link, and release a final executable file. The following figure shows the development cycle as a flowchart.
Introduction
CodeWarrior IDE Overview

Figure 1.1 Development Cycle Diagram

Legend
- Start or End of the process
- Development Cycle stage
- Decision stage
- Process flow
1.4.2 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 CodeWarrior IDE runs on popular operating systems, such as Windows, Solaris, and Linux. It uses virtually the same graphical user interface (GUI) across all Freescale Eclipse-based products.

- Multiple-language support
  Choose from multiple programming languages when developing software. The CodeWarrior 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 CodeWarrior IDE supports many common desktop and embedded processor families, such as x86, PowerPC, and MIPS.

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

The CodeWarrior IDE is composed of various plug-ins, each of which provide a specific functionality to the IDE. This chapter explains how to work with various extensions (plug-ins) in the Eclipse IDE.

This chapter explains:

- 2.1 CodeWarrior Projects View
- 2.2 Command Line Interface
- 2.3 Commander View
- 2.4 Concurrent Compilation
- 2.5 Console View
- 2.6 Context Menus
- 2.7 Diagnostic Information Export
- 2.8 Extracting CodeWarrior Configuration Details
- 2.9 Find and Open File
- 2.10 Importing Files
- 2.11 Key Mappings
- 2.12 Linker Command File Navigation
- 2.13 Multiple Compiler Support
- 2.14 New External File
- 2.15 Problems View
- 2.16 Referenced Projects
- 2.17 Target Management via Remote System Explorer
- 2.18 Target Processor Selection
- 2.19 Viewing CodeWarrior Plug-ins

2.1 CodeWarrior Projects View

The CodeWarrior Projects view displays all the resources in a workspace.
IDE Extensions
CodeWarrior Projects View

Figure 2.1 CodeWarrior Projects View

The CodeWarrior Projects view is an enhanced version of the C/C++ Projects view with the following improvements:

- 2.1.1 Active Configuration
- 2.1.2 Tree and List View
- 2.1.3 Column Headers
- 2.1.4 Quick Search
- 2.1.5 Filtering

2.1.1 Active Configuration

The CodeWarrior Projects view displays the name of active configuration associated with a project. Click the configuration name to view the context menu that displays all the configurations available to the project. You can switch to different configurations using this context menu.
2.1.2 Tree and List View

**CodeWarrior Project** view supports both hierarchal tree and flat list viewing of the resources in a workspace. The table below lists the toolbar icons that can be used to switch the viewing of resources.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Hierarchal View" /></td>
<td>Click the <strong>Show files in a hierarchal view</strong> icon in the <strong>CodeWarrior Project</strong> view toolbar to display the resources in hierarchal tree view.</td>
</tr>
<tr>
<td><img src="image" alt="Flat View" /></td>
<td>Click the <strong>Show files in a flat view</strong> icon in the <strong>CodeWarrior Project</strong> view toolbar to display the resources in flat list view.</td>
</tr>
</tbody>
</table>
IDE Extensions
CodeWarrior Projects View

2.1.3 Column Headers

You can click on the column header in the CodeWarrior Projects view to sort the list of files and folders based on the column. A small triangle in the column header indicates the active column and the sort order. If a column is active, clicking on its header toggles between the descending and ascending order.

NOTE The files can be sorted in both hierarchal and flat list views. Sorting is not case-sensitive for strings.

To add a column header in the CodeWarrior Projects view:

1. Select Customize Column Headers from the CodeWarrior Projects view pull-down menu.
2. Check a check box to enable or disable the corresponding column in the CodeWarrior Projects view. Alternatively, you can click Select All or Deselect All to enable or disable all the columns listed in the dialog box.
3. Click OK.
NOTE You cannot customize the FileName column using the Customize Column Headers dialog box.

The column header is added in the Codewarrior Projects view.

2.1.4 Quick Search

Codewarrior Projects view provides Quick Search that lets you filter the files in the current view based on the expression you enter. Quick Search provides the following features:

- Type Ahead — Type the first few letters of the file name and the Codewarrior Projects view automatically selects the appropriate file based on the string typed.
- Wildcard character support — You can also use basic wildcard characters, such as? and *, to extend your search.

NOTE The Codewarrior Project view automatically switches to the flat view when an expression is entered in the Search Text text box.

Figure 2.4 Codewarrior Projects View — Quick Search

Click the Popup icon in the Codewarrior Project view toolbar to specify the fields in which the Eclipse IDE searches for the expression typed in the Search Text text box.
IDE Extensions

**CodeWarrior Projects View**

**NOTE** The fields displayed when you click the **Popup** button depends on the headers enabled in the view.

Click the **Erase Text** icon in the **CodeWarrior Project** view toolbar to clear the Quick Search query. The CodeWarrior Projects view reverts to the normal view displaying all the folders and files in the workspace.

### 2.1.5 Filtering

**CodeWarrior Projects** view enables you to filter the elements being displayed. The following four filters have been added to filter the content in the **Project** view.

- **Generated Files** — Filters the output directory associated with each build configuration. This contains all files generated by a build including the executable files, object files, ephemeral makefiles, dependency files, map files, and other such elements. Typically, these files are all contained within a directory named after the build configuration. The entire directory is filtered.
- **Includes** — Filters out the **Includes** element, which shows the include paths the project is using and the included files.
- **Launch Configurations** — Filters `.launch` files, which are the launch configurations stored with the project. Typically, these files are stored in a `Debug_Settings` folder. In such case, the entire folder is filtered.
- **Referenced Projects** — Filters the Referenced Projects element that shows what other projects and build-configurations are referenced by the project.

To filter the content of the **Projects** view:

1. Click the inverted triangle icon in the **Project** view.
   
   The context menu appears.
2. Select **Filters**.
   
   The **C Element Filters** dialog box appears.
3. Check the Filter element you want to exclude from the **Project** view.
4. Click **OK**.

### 2.2 Command Line Interface

A new command-line tool, *ecd.exe*, is installed along with the *cwide.exe* that allows you to run build commands.

To create an Eclipse build from the *ecd* command line:

1. Click the *ecd.exe* file located in the `<CWInstallDir>eclipse\` folder to invoke the *ecd* command line.
2. In the *ecd* command line, type the following command:

   ```
   ecd -build -data my_workspace_path -project my_project_path
   ```

**NOTE** Projects specified by the `-project` flag that are not present in the workspace (either the default one or the one specified by the `-data` flag) are automatically imported in the workspace as existing project in the file system, and recorded in the workspace metadata.

The *ecd* commands are listed below:
IDE Extensions
Command Line Interface

- build
- getOptions
- generateMakefiles
- references
- setOptions
- updateWorkspace

**build**

Builds a set of C/C++ projects. Multiple-project flags can be passed on the same command invocation. The build tool output is generated on the command line, and the build result is returned by ecd.exe return code, as 0 for success, and -1 for failure.

**Syntax**

ecd.exe -build [-verbose] [-cleanAll] [-project path [-config name | -allConfigs] -cleanBuild]

**Parameters**

- **-cleanBuild**
  
  The **-cleanBuild** command applies to the preceding **-project** only.

- **-cleanAll**

  The **-cleanAll** command applies to all **-project** flags.

- **-config**

  The build configuration name. If the **-config** flag isn't specified, the default build configuration is used.

**getOptions**

Prints to the standard output C/C++ managed build, launch configuration or RSE system settings.

**Syntax**

ecd.exe -getOptions -project path [-config name | -allConfigs] [-file path] [-option option-name] [-launchConfig name | -allLaunchConfigs] [-rseSystem name | -allRseSystems]
IDE Extensions
Command Line Interface

Parameters

- **-config**
  The build configuration name. If the `-config` flag isn’t specified, the default build configuration is used.

- **-allConfigs**
  Specifies that all build configurations will be edited or listed

- **-file**
  The file path of a file included in the project. If the `-file` flag is specified, a file-level setting is retrieved instead of a build configuration level setting(s). The `-file` flag does not apply to the `-launchConfig`, `-allLaunchConfigs` and `-rseSystem` flags.

- **-option**
  The option id of the setting. If the option setting isn’t specified, all options are printed in a key=value format instead of a single option value, which could be used for discovering the list of option ids in a given build configuration, launch configuration or RSE system.

- **-Option-id**
  The option id of the setting.

- **-launchConfig**
  The launch configuration name. Allows retrieving launch configuration settings.

- **-allLaunchConfigs**
  Allow retrieving all launch configuration settings.

- **-rseSystem**
  The RSE system name. Allow retrieving RSE system settings.

---

**generateMakefiles**

Create the makefiles required to build a C/C++ project.

**Syntax**

```
ecd.exe -generateMakefiles [-verbose] [ -project path [ - config name ] [-allConfig]
```

**Parameters**

- **-config**
IDE Extensions
Command Line Interface

The build configuration name. If the -config flag isn't specified, the default build configuration is used.

-data workspace-path

The -data workspace-path flag can be used to specify a custom workspace.

references

List, add or remove all the referenced project and build configurations in a project.

Syntax

ecd.exe -references -project path [-config name | -allConfigs] (-list | -add | -remove) referencedProjectLocation [buildConfigurationName]

Parameters

-config name

The name of the build configuration to edit or list referenced project. If the -config flag is omitted, the active build configuration will be used

-allConfigs

Specifies that all build configurations will be edited or listed.

-list

List all the referenced projects and build configurations

-add referencedProjectLocation [buildConfigurationName]

Adds a new referenced project, specified by the 'referencedProjectLocation', which can be either an absolute path, or a variable relative path (relative to the path variables defined in the project specified by the -project flag). If the buildConfigurationName is specified, a specific the build configuration rather than the active build configuration will be referenced.

-remove referencedProjectLocation [buildConfigurationName]

Removes an existing referenced project, specified by the 'referencedProjectLocation', which can be either an absolute path, or a variable relative path (relative to the path variables defined in the project specified by the -project flag). If the buildConfigurationName is specified, only the specific referenced build configuration will be removed, otherwise all references to the specified project will be removed.
setOptions

Modify C/C++ managed build, launch configuration or RSE system settings.

Syntax

ecd.exe -setOptions -project path [-config name | -allConfigs | -rseSystem name | -launchConfig name | -allLaunchConfigs] [-file path] (-set | -prepend | -append | -insert) option-name option-value

Parameters

-config

The build configuration name. If the `-config' flag isn't specified, the default build configuration will be used.

-file

The path of a file included in the project. If the `file' flag is specified, a file-level setting is changed instead of a build configuration level setting. The -file flag does not apply to the -launchConfig, -allLaunchConfigs and -rseSystem flags.

-set | -prepend | -append | -insert

A setting can be either changed by replacing its previous value by the new specified one, using the -set flag, or prepended or appended to the existing value using the -prepend and -append flags respectively. The -insert flag can be used for updating exist macros values in macro settings.

option-id

The option id of the setting. A complete option-id list can be obtained by using the -getOptions command documented above.

option-value

The new value of the setting to be changed.

-launchConfig

The launch configuration name. Allows modifying launch configuration settings.

-allLaunchConfigs

Allow retrieving all launch configuration settings.

-rseSystem

The RSE system name. Allow retrieving RSE system settings.
IDE Extensions

Commander View

**updateWorkspace**

Update a workspace .metadata by including any project already located in the workspace file system directory. Optionally, supports redirecting the standard output to a logfile. Also supports leaving the Workbench UI open with the `-noclose` flag.

**Syntax**

ecd.exe -updateWorkspace -data workspace-path [-logfile path] [-noclose]

### 2.3 Commander View

**Commander** view provides quick access to some of the common and basic CodeWarrior operations. This concept is similar to that of the Quick Launch Bar in Windows. The **Commander** view provides an icon and a descriptive label for each action. The Commander view is optional and customizable.

**Commander** view is not only a new place for existing commands, but also for new commands which optimize common user workflows. Some Commander view commands map directly to existing commands in the IDE. For example, the **Welcome screen** command and the **Build All** command. These two commands behave identically to the commands in the IDE menu and toolbar. However, most commands are either improvements on existing commands or commands which reduce the number of steps to get to an existing user interface or functionality.

In **Commander** view, some commands have variant commands that appear next to the root command. The root command works for a single project; while the All variant works for all projects in the workspace. For example, the **Build** command. These are two independent commands on a single line. The All variant is always enabled because it doesn't need a project context, while the **Build** command is enabled only when there is a selection in the **CodeWarrior Projects** view or the view is pinned to a project ([2.3.3 Pinning Commander View](#)). A variant action does not have an icon and is visible only if its base command is shown.

In **Commander** view, each command has a key binding, which is displayed in the tooltip of the command. By default, the **Commander** view is docked under the **CodeWarrior Projects** view but like any other view it can be moved to any location within the CodeWarrior IDE.

**NOTE**  Commands displayed in the **Commander** view may vary from product to product.

The following figure displays the **Commander** view.
The following table describes the commands in the **Commander** view.

**Table 2.2 Commands in Commander View**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import Project</td>
<td>Imports an existing project into the workspace.</td>
</tr>
<tr>
<td>Import example project</td>
<td>Imports one or more example projects into the workspace.</td>
</tr>
<tr>
<td>Import MCU executable file</td>
<td>Imports an existing MCU executable file. This command is available with Microcontroller products only.</td>
</tr>
<tr>
<td>New MCU Project</td>
<td>Creates a new MCU bareboard project. This command is available with Microcontroller products only.</td>
</tr>
<tr>
<td>Build</td>
<td>Builds the selected or pinned project. (All) Build all projects.</td>
</tr>
</tbody>
</table>
2.3.1 Command Groups

In Commander view, the commands are grouped in the following four groups:

- Project Creation
- Build/Debug
- Settings
- Miscellaneous

Each group is a collapsible section, as shown in the following figure:

---

Table 2.2 Commands in Commander View

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean</td>
<td>Cleans the selected or the pinned project. (All) Clean all projects.</td>
</tr>
<tr>
<td>Project Settings</td>
<td>Shows the property page for the selected or the pinned project.</td>
</tr>
<tr>
<td>Build Settings</td>
<td>Shows the build properties for the selected or the pinned project. The build properties are simply a specific page with in project's properties.</td>
</tr>
<tr>
<td>Debug Settings</td>
<td>Opens the editor for the active launch configuration of the selected or pinned project.</td>
</tr>
<tr>
<td>Welcome Screen</td>
<td>Opens the Welcome screen.</td>
</tr>
<tr>
<td>Quick Access</td>
<td>Provides quick access to any IDE element.</td>
</tr>
<tr>
<td>Flash Programmer</td>
<td>Opens Flash programmer.</td>
</tr>
</tbody>
</table>

---

CodeWarrior Common Features Guide
2.3.2 Customizing Commander View

You can choose the commands to be displayed in the Commander view. The Customize option enables you to customize the content of the Commander view.

To customize the Commander view:

1. Click the View Menu button from the Commander view toolbar.

The pop-up menu appears.

Figure 2.8 Customize Menu
2. Select Customize.

The Commander View Customization dialog box appears.

Figure 2.9 Customizing Commander View

3. Expand a command group.

4. Uncheck/check the command you want to hide/display.

5. Click OK.

The selected commands appear in the Commander view.

NOTE Clicking the Restore Default button sets the command set choices to the default combination.

2.3.3 Pinning Commander View

If you have multiple projects in your workspace but work mostly on a particular project, then you are allowed to pin the Commander view to a specific project. So whenever you perform another task from the Commander view, it uses the pinned project for its context regardless of the project selected in the CodeWarrior Projects view.
Click the **Pin to Project** button in the **Commander** view to set the context to a particular project. A label also appears in the top-right corner to the Commander view specifying the project the view is pinned to.

The following figure displays the pinned **Commander** view:

![Figure 2.10 Pinning Commander View](image)

### 2.4 Concurrent Compilation

The concurrent compilation feature enables you to specify number of processes to compile the project.

To enable the concurrent compilation for a project:

1. In the CodeWarrior Project view explorer, right click on the project folder. A context menu appears.
2. Select **Properties** from the context menu. The **Properties for <project>** dialog box opens.
3. Select **C/C++ Build** from left panel of the **Properties for <project>** dialog box. The C/C++ build properties appear in the right panel of the **Properties for <project>** dialog box.
4. Select the **Behaviour** tab. The C/C++ build behavior properties appear under the **Behaviour** tab in the **Properties for <project>** dialog box.
Figure 2.11 Properties for <Project> Dialog Box

5. Check the Use parallel build check box.

   The Use optimal jobs number and the Use parallel jobs options are enabled.

   • Use optimal jobs number option - Lets the system determine the optimal number of parallel jobs to perform.

   • Use parallel jobs option - Lets you specify the maximum number of parallel jobs to perform.

   NOTE CodeWarrior Power Architecture does not support using parallel jobs. For this reason, the Use parallel jobs option is not available in CodeWarrior Power Architecture 10.0.2.

6. Select the Use optimal jobs number option or the Use parallel jobs option.

7. Click Apply.

8. Click OK.

You have now specified number of processes to compile the project.

2.5 Console View

The CodeWarrior Console view displays the output from the build (standard out and standard error) as it is generated by the build process. Double-clicking the error or warning message in the Console view moves the cursor to the error-source in the Editor window.
Context Menus provide shortcuts to frequently used menu commands. The available menu commands change, based on the context of the selected item.

Use context menus to apply context-specific commands to selected items. Right-click on an item to open a context menu for that item. The context menu appears, displaying menu commands applicable to the selected item.

Examples of situations in which the debugger displays a context menu are:

- Changing the format of variables displayed in variable panes
- Manipulating breakpoints and the program counter in source code panes
- Viewing memory in separate views

**TIP** To discover additional features, try right-clicking in each IDE view to see what commands are presented in the context menu that appears.

The following figure shows the Context Menu > Variables view displays.
2.7 Diagnostic Information Export

The Diagnostic Information Export feature allows you to export error log information to Freescale support group to diagnose the issue you have encountered while working on the CodeWarrior product. You can export diagnostic information in the following two ways:

- Whenever an error dialog invokes to inform some exception has occurred, the dialog box displays an option to open the Export wizard. You can then select the files you want to send to Freescale support.
- You can manually open the Export wizard to generate an archive of logs and files to report any issue that you have encountered.

2.7.1 General Settings for Diagnostic Information

To set general settings for diagnostic information, follow the steps given below:

1. Select Windows > Preferences from the IDE menu bar.

   The Preferences dialog box appears.
2. Expand the **General** group and select **Diagnostic Information**.

   The **Diagnostic Information** page appears.

3. Enter the number of days for which you want to display the diagnostic information details in the export wizard.

4. Select the **Privacy** option by dragging the bar to low, medium and high.

   Privacy level setting is used to filter the content of the logs.
   - **Low**: The file is sent as is.
   - **Medium**: The personal information is obfuscated. You can click on the customize option to view or modify filter.
   - **High**: The personal information is removed. Filters are used in the rest of the content.

5. Click **Customize** to set privacy filters.

   The **Customize Filters** dialog box appears.
IDE Extensions
Diagnostic Information Export

Figure 2.15 Diagnostic Information - Customize Filter

- You can add, remove, and modify filters and click **OK**.

6. Enter **Contact Name** and **Contact Email** in the contact information text box. This information is optional though Freescale will not share this information with anyone.

7. Click **Restore Defaults** to apply default factory settings.

8. Click **OK**.

You have set the general settings for diagnostic information.

2.7.2 Export Diagnostic Information

To export diagnostic information into an archive file in workspace, follow the steps given below:

1. Select **File > Export** from the IDE menu bar

   The Export dialog box appears.

2. Expand the **General** group and select **Diagnostic Information** option.
3. **Click Next.**

   The **Diagnostic Information Export Wizard** appears.
IDE Extensions
Diagnostic Information Export

Figure 2.17 Diagnostic Information Export Wizard Dialog Box

4. Check the Source checkbox to select the information that will be exported into the archive file.

**NOTE** You must select at least one file for export.

5. Click **Browse** to select a different archive file location.

6. Select the **Privacy option** or click **Customize** to set your privacy level. The Customize Filters dialog box appears.

**NOTE** The Customize Filters dialog can be opened through **Customize** button in the Diagnostic Information Export Wizard (**Figure 2.17**) or in the Preferences dialog box (**Figure 2.14**).

7. Click **Preview** to view the text that will be sent to Freescale from the wizard.
   The **Preview details** dialog box appears.
You can also check if more filters are needed to protect any sensitive information from leakage.

8. Click **OK**.

9. Click **Next** on the Diagnostic Information Export Wizard dialog box.

The **Reproducible Details** page appears.
10. Enter the reproducible steps and any other relevant information in the Details to recreate the issue text box.

11. Click Add to add additional files to the archive file for diagnosis.

12. Click Finish.

### 2.8 Extracting CodeWarrior Configuration Details

To extract the configuration details of the currently installed CodeWarrior features and associated plug-ins:

   
   The About Freescale CodeWarrior dialog box appears.

2. Click Installation Details.
   
   The CodeWarrior Development Studio Installation Details dialog box appears.
3. Select the **Configuration** tab.
   
The configuration data appears.

![Figure 2.20 Configuration Tab](image)

4. Click the **Copy to Clipboard** button to copy the configuration data.

5. Paste the copied data in any text editor, such as notepad or winword and save the data.

6. Click **Close**.

### 2.9 Find and Open File

The **Find and Open File** dialog box enables you to open a selected path or file in the Editor area.

To open particular path or file in the Editor area:
IDE Extensions

Importing Files

1. Select File > Open Path from the IDE menu bar.
   The Find and Open File dialog box appears.

Figure 2.21 Find and Open File Dialog Box

2. Enter a file descriptor. The file descriptor can be a simple file name, a partial path or a full path. The path delimiters can also be different from that of the native platform delimiters. For example, you can use “/” on a Windows host instead of “\”.

3. Click OK.

Eclipse IDE performs the following actions:

- Scans for a matching file descriptor in all the open editor windows. If a match is found, the IDE activates the open editor window in the Editor area.
- If no open editor windows match the specified file descriptor, IDE searches for a matching file in the accessible paths of the current project. If a match is found, IDE opens the file in a new editor window in the Editor area. If the file is not found, IDE generate a beep sound.

**NOTE** The Open Path feature is also invoked when a file name is selected in an #include directive in a source file. In such a case, the IDE opens the file in the Editor area without displaying the Find and Open File dialog box.

2.10 Importing Files

You can import files into the Workbench by:

- 2.10.1 CodeWarrior Drag and Drop Support
- 2.10.2 Using the Import Wizard

2.10.1 CodeWarrior Drag and Drop Support

The CodeWarrior Drag and Drop support extends the following features to the CodeWarrior IDE.

- Allows user to drop different files and folders to the Workbench window.
IDE Extensions
Importing Files

- Allows user to drop multiple files and folders simultaneously and handle them properly in a sequence.
- Supports files and directories created by the earlier versions of CodeWarrior (.mcp files).

**NOTE** A classic project file has the .mcp extension.

- Resolves potential handling ownership conflict between different components over the dropped objects.

For example, to create a link to a project existing in a different workspace from the current workspace:
1. Open the workspace using Windows Explorer (In Linux, you can open the workspace using Shell).
2. Select and drag the project folder over the CodeWarrior IDE.

The CodeWarrior IDE effectively handles the files and folders dropped to the Workbench. A link to the existing project is created in the CodeWarrior Projects view.

### 2.10.2 Using the Import Wizard

The Import wizard helps you import resources into the Workbench.

This section explains:
- 2.10.2.1 Import Existing Project
- 2.10.2.2 Import Example Project

#### 2.10.2.1 Import Existing Project

The Import Existing Project wizard lets you import existing project into the workspace. To import existing project:

1. Select **File > Import** from the IDE menu bar.
   - The Import dialog box appears.
2. Expand the **General** folder.
3. Select **Existing Project into Workspace**.
4. Click **Next**.
   The **Import Project** wizard appears.
Figure 2.23 Import Projects Dialog Box

5. Select **Select root directory** or **Select archive file** option and click the associated **Browse** to locate the directory or file containing the projects.

The list of existing projects appear under the Project group.

6. Under **Projects**, select the project or projects which you would like to import.

**NOTE**  When there are projects with the same name in the list, only one of them is selected by default and allow you to change the selections as needed.

If any error occurs during the import, the project list is updated and all the projects that were already imported are disabled.
NOTE You may click Copy projects into workspace checkbox to copy the project into workspace. You may also click Add Project to working sets checkbox to include the project in working sets.

7. Click Finish.

2.10.2.2 Import Example Project

The Import Example Project wizard lets you import example project into the workspace. To import an example project:

1. Select File > Import from the IDE menu bar.

The Import dialog box appears.

Figure 2.24 Import Dialog Box

2. Expand the CodeWarrior folder.

3. Select Example Project.
4. Click Next.

The Import Project wizard appears.

**Figure 2.25 Import Projects Dialog Box**

5. Select **Select root directory** or **Select archive file** option and click the associated **Browse** to locate the directory or file containing the projects.

The list of existing projects appear under the Project group.

6. Under **Projects**, select the project or projects which you would like to import.

**NOTE**  When there are projects with the same name in the list, only one of them is selected by default and allow you to change the selections as needed.  
If any error occurs during the import, the project list is updated and all the projects that were already imported are disabled.
7. Click Finish.

### 2.11 Key Mappings

CodeWarrior Eclipse IDE accepts keyboard shortcuts, or *key bindings*, for frequently used operations. At any time, you can obtain a list of available key bindings using Key Assist. To activate the Key Assist view, select *Help > Key Assist*.

Alternatively, press **Ctrl+Shift+L** keys to display a list of available key bindings in Eclipse.

**NOTE** Key bindings can vary based on the current context of Eclipse, platform and locale. The current platform and locale is determined when Eclipse starts, and does not vary over the course of an Eclipse instance.

The following table lists and defines the key mappings for Classic IDE and Eclipse IDE.

**Table 2.3  Key Mappings - Classic IDE and Eclipse IDE**

<table>
<thead>
<tr>
<th>Behaviour</th>
<th>Classic IDE</th>
<th>Eclipse IDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Ctrl + Shift + N</td>
<td>Ctrl + N</td>
</tr>
<tr>
<td>Open</td>
<td>Ctrl + O</td>
<td>Ctrl + Shift + A</td>
</tr>
<tr>
<td>Open Path (Eclipse IDE)</td>
<td>Ctrl + Shift + W</td>
<td>Ctrl + Shift + W</td>
</tr>
<tr>
<td>Close</td>
<td>Ctrl + W</td>
<td>Ctrl + F4</td>
</tr>
<tr>
<td>Close All</td>
<td>Ctrl + Shift + W</td>
<td>Ctrl + Shift + W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ctrl + Shift + F4</td>
</tr>
<tr>
<td>Save</td>
<td>Ctrl + S</td>
<td>Ctrl + S</td>
</tr>
<tr>
<td>Save All</td>
<td>Ctrl + Shift + S</td>
<td>Ctrl + Shift + S</td>
</tr>
<tr>
<td>Print</td>
<td>Ctrl + P</td>
<td>Ctrl + P</td>
</tr>
<tr>
<td>Undo</td>
<td>Ctrl + Z</td>
<td>Ctrl + Z</td>
</tr>
<tr>
<td></td>
<td>Alt + Backspace</td>
<td></td>
</tr>
<tr>
<td>Redo</td>
<td>Ctrl + Shift + Z</td>
<td>Ctrl + Y</td>
</tr>
</tbody>
</table>
# IDE Extensions

## Key Mappings

### Table 2.3 Key Mappings - Classic IDE and Eclipse IDE

<table>
<thead>
<tr>
<th>Behaviour</th>
<th>Classic IDE</th>
<th>Eclipse IDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Ctrl + X</td>
<td>Ctrl + X</td>
</tr>
<tr>
<td></td>
<td>Shift + Delete</td>
<td>Shift + Delete</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl + C</td>
<td>Ctrl + C</td>
</tr>
<tr>
<td></td>
<td>Ctrl + Insert</td>
<td>Ctrl + Insert</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl + V</td>
<td>Ctrl + V</td>
</tr>
<tr>
<td></td>
<td>Shift + Insert</td>
<td>Shift + Insert</td>
</tr>
<tr>
<td>Delete</td>
<td>Del</td>
<td>Del</td>
</tr>
<tr>
<td>Select All</td>
<td>Ctrl + A</td>
<td>Ctrl + A</td>
</tr>
<tr>
<td>Find Next</td>
<td>F3</td>
<td>Ctrl + K</td>
</tr>
<tr>
<td>Find Previous</td>
<td>Shift + F3</td>
<td>Ctrl + Shift + K</td>
</tr>
<tr>
<td>Go to Line</td>
<td>Ctrl + G</td>
<td>Ctrl + L</td>
</tr>
<tr>
<td>Debug</td>
<td>F5</td>
<td>F11</td>
</tr>
<tr>
<td>Run</td>
<td>Ctrl + F5</td>
<td>Ctrl + F11</td>
</tr>
<tr>
<td>Step Over</td>
<td>F10</td>
<td>F6</td>
</tr>
<tr>
<td>Step Into</td>
<td>F11</td>
<td>F5</td>
</tr>
<tr>
<td>Step Out</td>
<td>Shift + F11</td>
<td>F7</td>
</tr>
<tr>
<td>Step Return (Eclipse IDE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable/Disable Breakpoint</td>
<td>Ctrl + F9</td>
<td>Ctrl + Shift + B</td>
</tr>
<tr>
<td>Toggle Breakpoint (Eclipse IDE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Make</td>
<td>F7</td>
<td>Ctrl + B</td>
</tr>
<tr>
<td>Build All (Eclipse IDE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Line Up</td>
<td>Up</td>
<td>Alt + Up</td>
</tr>
<tr>
<td>Move Line Down</td>
<td>Down</td>
<td>Alt + Down</td>
</tr>
</tbody>
</table>
2.12 Linker Command File Navigation

The linker command file (LCF) navigation feature enables you to click on strings containing path names in the CodeWarrior Editor and navigate to the specified file.

**NOTE** The CodeWarrior Editor recognizes the files with `.lcf`, `.cmd`, and `.13k` file extensions as LCF files. A file with `.txt` file extension is not recognized as a LCF file.

To navigate to a LCF file:

1. Open the LCF file in the text editor.

2. In the LCF file, right-click on a line that refers to a text file.

   The context-menu appears.

3. Select Open Reference from the context menu.
2.13 Multiple Compiler Support

This feature enables you to switch between multiple versions of a toolchain. When you acquire a new version of the command line tools associated with an integration plug-in, multiple compiler feature allows you to add the new version into the list of available toolchain versions.

To switch between multiple versions of a toolchain:
1. In the CodeWarrior Project view explorer, right-click on the project folder. A context menu appears.
2. Select Properties from the context menu. The Properties for <project> dialog box opens.
3. Select C/C++ Build > Settings from the left panel of the Properties for <project> dialog box. The C/C++ build settings appear in the right panel of the Properties for <project> dialog box.
4. Select the Build Tool Versions tab. The build tool version settings appear in the Build Tool Versions panel.

The referenced text file opens in the Editor window.
IDE Extensions

Multiple Compiler Support

Figure 2.28 Properties for <Project> Dialog Box

The following table lists and defines the **Build Tool Versions** panel controls.

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Tool Versions table</td>
<td>Lists multiple toolchain versions.</td>
</tr>
<tr>
<td>Add button</td>
<td>Enables you to add a new toolchain version.</td>
</tr>
<tr>
<td>Edit button</td>
<td>Enables you to edit the currently selected toolchain version.</td>
</tr>
<tr>
<td>Delete button</td>
<td>Enables you to delete a toolchain version.</td>
</tr>
<tr>
<td>Set as Default button</td>
<td>Enable you to set the currently selected toolchain versions as default toolchain version for building projects.</td>
</tr>
<tr>
<td>Restore Defaults button</td>
<td>Enables you to restore the default toolchain version.</td>
</tr>
</tbody>
</table>
5. Click **Apply**.
6. Click **OK**.

Now, you can switch between multiple versions of a toolchain.

### 2.14 New External File

CodeWarrior Eclipse IDE now supports creating, opening, and saving files that are located outside the current workspace. To create a non-project file:

1. Click **File > New > Other**.
   
   The **New** wizard appears.

**Figure 2.29 New — Select a Wizard Page**

2. Select **New External File** under the **General** category.
3. Click **Next**.
   
   The **New External File** page appears.
4. Specify the path and filename.
5. Click Finish.

IDE opens the file in a new editor window in the Editor view.

2.15 Problems View

The Problems view displays build errors and warnings in a tree table control. The Problems view also displays the information, such as description, resource, path, location, and type for build errors and warnings. Double-click an error/warning to go to the location in the source from where the error/warning was generated.
Clicking the Toggle visibility of the details pane button in the Problems view displays the Details pane. The Details pane displays full description of the selected error/warning.

Figure 2.32 Problems View — Details Pane

2.16 Referenced Projects

Referenced projects allow you to create build dependencies between projects. If project A is setup as a referenced project for project B, then project A will be built before each project B build. Referenced projects are automatically imported and opened when a project is imported in the workspace, so referenced projects can be used to automatically populate the workspace with a set of projects.

This topic explains:

- 2.16.1 Create a Referenced Project
- 2.16.2 Displaying Referenced Projects in CodeWarrior Project View
- 2.16.3 Automatic Linking with Referenced Project Build Artifact
- 2.16.4 Circular Build Dependencies

2.16.1 Create a Referenced Project

To create a referenced project B in project A, follow the steps given below:
1. Select and right-click the project in the CodeWarrior Projects View.
2. Select Properties from the context menu that appears.
   The Properties dialog box appears.
3. Click C/C++ Build option and click References tab on the C/C++ Build properties page.
4. Select the project B in the tree.
5. Click **OK** to include Project B in Project A.

**NOTE** You can also create a project reference by dragging ‘Project B’ onto ‘Project A’ in the CodeWarrior Projects view.

The following table lists various options available in **C/C++ Build > References** page.

**Table 2.5 C/C++ Build > References Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build referenced projects and build configurations when building this project</td>
<td>If checked, enable building referenced projects and build configurations while building project. This option is also available from the CodeWarrior Projects view, by right clicking on the ‘Referenced Projects’ element.</td>
</tr>
<tr>
<td>Expand All</td>
<td>Click to expand all referenced projects in the tree.</td>
</tr>
<tr>
<td>Collapse All</td>
<td>Click to collapse all the referenced projects in the tree.</td>
</tr>
</tbody>
</table>
2.16.2 Displaying Referenced Projects in CodeWarrior Project View

CodeWarrior Projects view displays the referenced projects and highlights which build configuration is referenced.

The following figure shows the Referenced Project in CodeWarrior Projects View.

**Table 2.5  C/C++ Build > References Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move Up/Move Down</td>
<td>Change the referenced project build order. The first project shown will be built first.</td>
</tr>
<tr>
<td>Edit Location</td>
<td>Allows you to change the location of referenced project. The project location is recorded automatically as a project-relative path, and can make use of the project path variables to be portable across machines</td>
</tr>
<tr>
<td>Build Projects referencing this build configuration</td>
<td>Causes projects that include the current project as a referenced project to be included in the current build dependency, and be built when the current project is built.</td>
</tr>
<tr>
<td>Severity level for missing references projects build error makers</td>
<td>Allows you to select Error or Warning to display severity level for missing references.</td>
</tr>
<tr>
<td>Restore Defaults</td>
<td>Restores default factory settings</td>
</tr>
<tr>
<td>Apply</td>
<td>Save your changes without closing dialog box.</td>
</tr>
<tr>
<td>OK</td>
<td>Save your changes and close dialog box.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Close dialog box without saving.</td>
</tr>
</tbody>
</table>
NOTE All the project references appear under a Referenced Projects folder, which only shows if the project contains at least one referenced project.

The following table shows the context menu options available through the CodeWarrior Projects view:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include in Build</td>
<td>It allows you to quickly toggle whether all referenced projects and build configurations will be included in the build or not.</td>
</tr>
<tr>
<td>Open/Import Project</td>
<td>It is available when the referenced project element is selected, and enabled if the referenced project is either closed or does not exist in the workspace. The action imports and/or opens the referenced project in the workspace.</td>
</tr>
<tr>
<td>Close Project</td>
<td>It is available when the referenced project element is selected, and enabled if the referenced project is open in the workspace. It allows you to close the referenced project in the workspace.</td>
</tr>
</tbody>
</table>
2.16.3 Automatic Linking with Referenced Project Build Artifact

A project can automatically include the build artifact of its referenced project in its linker input settings by using the `${ReferencedConfigurationsOutput}` build variable as shown in figure below. The `${ReferencedConfigurationsOutput}` build variable contains automatically the list of the referenced projects build artifacts paths.

**Figure 2.35 Referenced Project Build Artifact Dialog Box**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Location</td>
<td>Allows you to edit the location of the referenced project.</td>
</tr>
<tr>
<td>Remove Referenced Project</td>
<td>Allows you to delete the referenced project.</td>
</tr>
</tbody>
</table>

2.16.4 Circular Build Dependencies

If two projects depend on each other through referenced projects, a build error will be generated, since circular build references is not permissible by design.

If multiple projects needs to be built at the same time, a single referenced project can be used for the first dependency, and the second project can set the 'Projects referencing this build configuration' flag, so that
IDE Extensions

Target Management via Remote System Explorer

building the first or second project will cause the other one to be built automatically. This also applies to referenced build configuration for single or multiple projects.

For example, if two different build configurations (Debug and Release) of a single project need to be built no matter which one of the two build configuration is active, the 'Debug' build configuration can reference the 'Release' build configuration in the 'References' tab, and the 'Release' build configuration can have its Projects referencing this build configuration flag set to 'true', so that both the 'Debug' and 'Release' build configurations will be built together, no matter which one is active.

2.17 Target Management via Remote System Explorer

Remote System Explorer operates with remote system entities. The CodeWarrior uses two types of remote systems for describing Freescale hardware with respect to debug process:

1. Target Configuration
2. Connection Configuration

A target configuration defines initialization, and target parameters. The Remote System Explorer provides data models and frameworks to configure and manage both target and connection configurations.

The configuration model for bareboard debug uses a target and a connection configuration that allows you to define a single target configuration that can be referred by multiple connection configurations. Each such configuration is implemented as Remote System host.

2.17.1 Creating Remote System

To create a remote system:

1. Click File > New > Other.

The New wizard appears.
2. Select Connection under the Remote System Explorer category.

3. Click Next.

   The New Connection page appears.
4. Expand **CodeWarrior Bareboard Debugging** and select a remote target type.

A remote target type represents a particular type of remote system. The supported remote target types are:

- **Hardware or Simulator Connection** — Connection configuration for a hardware-based or simulated system. For more information, refer to the 2.17.2 Creating Hardware or Simulator Connection Configuration.

- **Hardware or Simulator Target** — Target configuration for a hardware-based or simulated target. For more information, refer to the 2.17.3 Creating Hardware or Simulator Target Configuration.

- **TRK Target** — System configuration for a system running the TRK debug agent. For more information, refer to the 2.17.4 Creating TRK Target Configuration.

5. Click **Next**.

The new configuration settings appear. You need to specify configuration settings depending upon the remote target type selected in the **New Connection** page.

6. Click **Finish**.
2.17.2 Creating Hardware or Simulator Connection Configuration

A hardware or simulator connection configuration enables you to create a connection configuration for a hardware-based or simulated target. To create a hardware or simulator connection configuration:

1. Click File > New > Other.
   The New wizard appears.
2. Select Connection under the Remote System Explorer category.
3. Click Next.
   The New Connection page appears.
4. Select Hardware or Simulator Connection from the CodeWarrior Bareboard Debugging category.
   The New Connection — Hardware or Simulator Connection page appears.

**Figure 2.38 New Connection — Hardware or Simulator Connection Configuration**

5. Select a parent profile from the Parent Profile drop-down list.
IDE Extensions
Target Management via Remote System Explorer

6. Type a configuration name in the **Name** text box.
7. Type connection description in the **Description** text box.
8. Select remote system template from the **Template** drop-down list. The remote system template is a pre-defined configuration fully supported by CodeWarrior debugger.
   - Select **None** from the template drop-down list when the current remote system configuration does not use any reference template.
9. Click **Apply Defaults** to load settings from reference template in the current configuration.
10. Select a target from the **Target** drop-down list.
    A target type is a CodeWarrior abstraction that represents the users target processor layout. This can be a simple processor or a set of processors as defined by a JTAG configuration file or a Power Architecture® device tree blob file.
11. Click **Edit** to add or remove target types.
12. Click **New** to create a new target configuration.
13. Select a connection type from the **Connection type** drop-down list.
    - Select **CCSSIM2 ISS** to specify setting for CodeWarrior Connection Server (CCS).
    - Select **Ethernet TAP** to specify settings for ethernet TAP connection.
    - Select **USB TAP** to specify settings for USB TAP connection.
14. Click **Finish**.
    The hardware or simulator connection configuration appears in the 2.17.5 Remote Systems View.

2.17.3 Creating Hardware or Simulator Target Configuration

A hardware or simulator target configuration enables you to connect to your target via a direct hardware connection or simulate your target. To create a bareboard or simulator remote system:

1. Click **File > New > Other**.
    The New wizard appears.
2. Select **Connection** under the **Remote System Explorer** category.
3. Click **Next**.
    The New Connection page appears.
4. Select **Hardware or Simulator Target** from the **CodeWarrior Bareboard Debugging** category.
    The New Connection — Hardware or Simulator Target page appears.
5. Select a parent profile from the **Parent Profile** drop-down list.
6. Type a configuration name in the **Name** text box.
7. Type target description in the **Description** text box.
8. Select remote system template from the **Template** drop-down list. The remote system template is a pre-defined configuration fully supported by CodeWarrior debugger.
   - Select **None** from the **template** drop-down list when the current remote system configuration does not use any reference template.
9. Click **Apply Defaults** to load settings from reference template in the current configuration.

**NOTE**  The differences between reference template and current configuration is highlighted. You can disable highlighting from Remote Systems view’s pop-up menu. The color used for highlighting can be changed from global preferences.
10. Select a target configuration from the **Target type** drop-down list.

A target type is a CodeWarrior abstraction that represents the user’s target processor layout. This can be a simple processor or a set of processors as defined by a JTAG configuration file or a Power Architecture® device tree blob file.

11. Click **Edit** to edit the current target configuration. Click **New** to create a new target configuration. For more information, refer to the 3.21.3 *Editing a Multicore Group*.

12. Click the **Initialization** tab.

The initialization settings page appears.

**Figure 2.40  Hardware or Simulator Target — Initialization Settings Page**

13. Specify the initialization settings to suit your needs.

14. Click the **Memory** tab.

The memory settings page appears.
15. Specify the target settings to suit your needs.

16. Click the I/O Model tab.

   The I/O Model page appears.

17. Specify the I/O model settings to suit your needs.

18. Click the Advanced tab.

   The Advanced page appears.
IDE Extensions
Target Management via Remote System Explorer

Figure 2.43 Hardware or Simulator Target — Advanced Settings Page

19. Check the Target is emulated by Palladium checkbox if required.
20. Click Finish.

   The hardware or simulator target appears in the 2.17.5 Remote Systems View.

2.17.4 Creating TRK Target Configuration

A TRK target enables you to create a target configuration for a target running the TRK debug agent. To create a TRK target configuration:
1. Click File > New > Other.
   The New wizard appears.
2. Select Connection under the Remote System Explorer category.
3. Click Next.
   The New Connection page appears.
4. Select TRK Target.
   The New Connection — TRK Target page appears.
5. Select a parent profile from the Parent Profile drop-down list.
6. Type a configuration name in the Name text box.
7. Type connection description in the Description text box.
8. Select remote system template from the Template drop-down list. The remote system template is a pre-defined configuration fully supported by CodeWarrior debugger.
   - Select None from the template drop-down list when the current remote system configuration does not use any reference template.
9. Click Apply Defaults to load settings from reference template in the current configuration.
10. Select a target type from the Target type drop-down list or click Edit to import the target type.
11. Click the Initialization tab.
   The initialization settings page appears. Specify the initialization settings to suit your needs.
12. Click the System tab.
   Specify the system settings to suit your needs.
13. Click Finish.

   The TRK target configuration appears in the 2.17.5 Remote Systems View.

2.17.5 Remote Systems View


Figure 2.44 Remote Systems View

2.17.5.1 Modifying Target or Connection Configuration

The Remote System view enables you to modify settings for a target or connection. To change target or connection settings:

2. Right-click a remote system name and select Properties from the context menu.

   The Properties for <Remote System> dialog box appears.
IDE Extensions
Target Management via Remote System Explorer

Figure 2.45 Properties for <Target or Connection> Dialog Box

3. Change the settings in this page to suit your needs.
4. Click OK.
The changes are applied to the target.

2.17.5.2 Exporting the Target or Connection Configuration

You can export a target or connection to an external file. To export a target or connection:
2. Right-click a remote system name and select Export from the context menu.
   The Save As dialog box appears.
3. Type a file name in the File name drop-down list.
4. Click Save.
2.17.5.3 Importing a Target or Connection Configuration

You can import a target or connection from an external file. To import a target or connection:
2. Right-click in the view and select Import from the context menu.
   The Open dialog box appears.
3. Select a remote system file.
4. Click Open.

2.17.5.4 Apply to Project

This feature allows you to set the active target and component of a launch configuration. To set the active target and component:
2. Right-click on the host and select Apply to Project from the context menu.
   The context menu with different projects and launch configurations appears.
3. Click on the item to apply the target selection in a project.
   The projects items are displayed with a check mark to show if the item has the target selected.
4. Click **Apply To All** option to apply the target selection for all projects.

### 2.17.5.5 Apply to Connection

This feature allows you to set a target configuration to a connection, without having to open the **Debug Configurations** dialog box. To set a target configuration:

2. Right-click on the host and select **Apply to Connection** from the context menu.
   The context menu with different target configurations appears.
3. Click on the item to apply the target configuration to the connection configuration. The selected connection is displayed with a check mark.

### 2.17.5.6 Automatic Removal of Unreferenced Remote Systems

When a CodeWarrior Remote System is no longer referenced by an open project, you may delete it automatically. To remove unreferenced remote systems:

1. Click **Window > Preferences > C/C++ > Debug > CodeWarrior Debugger**. The **CodeWarrior Debugger** dialog box appears.
2. Select options available in Remote System Settings drop-down list.
   - Do nothing
   - Show a dialog; ask me what to do
   - Delete the Remote System
3. Click OK.

The unreferenced remote systems are removed.

2.17.6 Automatic Project Remote System Setting Cache

The APSC feature automatically stores the settings of the Remote Systems referenced by a project's launch configurations. When the project is opened on a different machine or in a different workspace, the APSC feature automatically re-creates the missing Remote Systems for the project. This feature will also update and merge any Remote System setting that has changed between its project and workspace version.
APSC feature allows you to update the Remote System tree when a project is open in the workspace and its APSC cache doesn't match the current Remote System settings. APSC provides following two operations:

1. Remote System Missing
2. Remote System Changed

### 2.17.6.1 Remote System Missing

The Remote system missing dialog box appears when a project is open in the workspace and its APSC cache contains Remote Systems that do not exist. In such case, you will be asked to create missing objects.

The following figure shows Remote System Missing dialog box.

**Figure 2.49 Remote System Missing dialog box**

![Remote System Missing](image)

**NOTE**  
By default, the Remote System Missing dialog is not displayed unless the Remote System Project Cache workspace preferences are changed. Any missing host will be automatically re-created.

If you select **Yes** to create the missing Remote System, a new Remote System will be created and initialized with the cached settings.

If you select **No**, the Remote System Missing dialog box will be closed.

You can click on the **Configure Project Cache Settings** link to directly change the Remote System Project Cache preferences, to avoid automatically displaying this dialog box in future.

### 2.17.6.1.1 Remote System Project Cache Preferences

Projects that reference Remote Systems contain an internal cache of the referenced Remote Systems. When such projects are imported in the workspace, the Remote Systems in the cache may differ from the actual Remote Systems in the workspace or be missing entirely.

To configure merge settings:

1. Click Window > Preferences.
   
   The Preferences dialog box appears.

   **Figure 2.50 Remote System Project Cache Preferences Dialog Box**

2. Expand the Remote System tree control from the left-pane of the Preferences dialog box.
   
   You can configure the way the dialog appear by changing the Remote System Project Cache page as shown in Figure 2.50.

3. Select how to handle differing RSE System options.

4. Select how to handle missing RSE Systems options.

5. Select an action from the When a RSE System is no longer referenced by an open project dropdown list box.

6. Click OK to apply changes.
You have set the preferences for Remote System to handle differing Remote Systems and missing Remote Systems.

### 2.17.6.2 Remote System Changed

The Remote System Changed dialog box appears when a project is open in the workspace and its APSC cache contains Remote Systems that have different settings to the ones in the existing Remote System tree. In such case, you will be asked to update, discard or create a new set of objects for the cached Remote System settings.

The following figure shows **Remote System Changed** dialog box.

**Figure 2.51 Remote System Changed dialog box**

The **Remote System Changed** dialog box provides following three options to resolve the version differences:

1. Replace the current version with the project version.
2. Discard the project version and update the project to use the current version.
3. Create a new Remote System for the project version.

Select appropriate option and click **Ok**.
IDE Extensions
Target Management via Remote System Explorer

You can click on the Configure Project Cache Settings link to directly change the Remote System Project Cache preferences, to avoid automatically displaying this dialog box in future. For details refer 2.17.6.1.1 Remote System Project Cache Preferences.

NOTE You may keep the file containing the referenced remote systems, available in the <Project_name>/Referenced Systems folder in a version control system for future use.

2.17.7 Compatibility with Older Products

The CodeWarrior connection configuration and target configuration have moved from the Launching framework to the Remote System Explorer. This enables you to share the same connection and target configuration among many launch configurations and you can see all configuration for a Multicore system at a glance.

The following sections will show the facilities for migrating older projects to use RSE.

2.17.7.1 Display of Launch Configurations Needing Migration

CodeWarrior enables you to display migration candidates as information, warnings, or errors in the Problems view. You can also set preference to ignore or automatically migrate the migration candidates.
To configure migration preference:
1. Click **Window > Preferences**.
   
   The **Preferences** dialog box appears.
2. Expand the **Run/Debug > Launching** tree control from the left-pane of the **Preferences** dialog box.
3. Select **Launch Configurations**.

   The launch configuration preferences appear in the right-pane of the **Preferences** dialog box.
4. Select a value from the **Available migrations can be shown in the workbench as problems or applied automatically** drop-down list box.
   - **Ignore** — Ignores the migration candidates.
   - **Show as information** — Displays migration candidates as information in the **Problems** view.
   - **Show as warning** — Displays migration candidates as warnings in the **Problems** view.
   - **Show as error** — Displays migration candidates as errors in the **Problems** view.
   - **Apply automatically** — Automatically migrates all migration candidates.

5. Click **Apply**.

6. Click **OK**.

For projects created using Launching framework, a migration facility is provided.

### 2.17.7.2 Migrating Launch Configurations

For projects created using Launching framework, a migration facility is provided. CodeWarrior enables you to migrate launch configurations using these methods:
2.17.7.2.1 Migration using Smart Migration

You need to configure CodeWarrior to migrate launch configurations using Smart Migration method. To configure CodeWarrior for Smart Migration:

1. Click Window > Preferences.
   
   The Preferences dialog box appears.

2. Expand the C/C++ > Debug tree control from the left-pane of the Preferences dialog box.

   
   The CodeWarrior debugger preferences appear in the right-pane of the Preferences dialog box.

4. Select a migration option from the Smart Migration Selection drop-down list box.
IDE Extensions
**Target Management via Remote System Explorer**

- Do nothing — launch configuration not migrated, and no warning is displayed.
- Show a dialog; ask me what to do — Displays a dialog box to select a migration option.
- Perform Smart Migration, don’t reuse existing systems — Automatically migrate launch configurations without using existing Remote System.
- Perform Smart Migration, reuse existing system if possible — Automatically migrate launch configurations using existing remote systems (if possible).

5. Click **Apply**.
6. Click **OK**.

Now, when you open a project that was created using Launching framework, the CodeWarrior launches the Smart Migration utility (Figure 2.55) to migrate all migration candidates.

To migrate using Smart Migration
1. Open the Project in CodeWarrior
   The **Launch Configuration Migration to RSE Required** dialog box appears.

**Figure 2.55  Launch Configuration Migration using Smart Migration**

2. Select **Yes, Perform a Smart Migration now** option.
3. Check the **Re-use existing RSE system configuration if possible** option to reuse existing Remote System configurations. Otherwise, Smart Migration migrates launch configurations without re-using existing Remote System configurations.

**NOTE**  Click **No, ignore the problem for now** stops the migration process. You can still migrate launch configuration using Quick Fix or by directly editing the launch configurations.
IDE Extensions
Target Management via Remote System Explorer

4. Check Do this every time; don’t show this dialog again to save your selection as general preference.

5. Click OK.

The Migrate Launch Configuration for RSE dialog box appears.

Figure 2.56  Migrate Launch Configuration for RSE Dialog Box

6. Specify a name for the launch configuration in the Create new RSE system named: text box.

7. Click OK.

You have migrated using Smart Migration.

2.17.7.2.2 Migration Using Quick Fix

To migrate a launch configuration using Quick Fix:
1. Select a migration candidate in the Problems view.
2. Right-click and select Quick Fix from the context menu.

The Quick Fix dialog box appears.
3. Select a fix from the **Select a Fix** list box.
   - Update launch configuration(s) in the same project to use a shared Remote System configuration — Assigns the selected remote system to all launch configurations from a single project.
   - Update launch configuration(s) to use a shared Remote System configuration — Assigns the selected remote system to all launch configurations from workspace.
   - Migrate the launch configuration to be compatible with current tooling — Enables you to choose remote system for each selected launch configuration.

4. Select launch configurations to be migrated from the **Problems** table.

5. Click **Finish**.

   The **Migrate Launch Configuration for RSE** dialog box appears.
Figure 2.58 Migrate Launch Configuration for RSE Dialog Box

6. Select a remote system setting.
   - Select **Use existing RSE system** option to select an existing remote system from the drop-down list box.

   **NOTE** Remote systems compatible with the selected launch configuration are only listed in this drop-down list box.

   - Select **Create new RSE system named** to create a new remote system by the specified name in the text box.

7. Click **OK**.

   Alternatively, you can invoke the launch configuration migration dialog box using the **Preferences** dialog box:

   1. Select **Window > Preferences**.

      The **Preferences** dialog box appears.

   2. Expand the **Run/Debug > Launching** tree control from the left-pane of the **Preferences** dialog box.

   3. Select **Launch Configurations**.

      The launch configuration preferences appear in the right-pane of the **Preferences** dialog box.
4. Click Migrate.

The Select Launch Configurations dialog box appears.
5. Select the launch configurations to migrate.
6. Click OK.

The Migrate Launch Configuration for RSE dialog box (Figure 2.58) appears.

7. Select a remote system setting.
   • Select Use existing RSE system option to select an existing remote system from the drop-down list box.
   • Select Create new RSE system named to create a new remote system by the specified name in the text box.
8. Click OK.

2.18 Target Processor Selection

The target process selection combo box enables you to refine the list of target processors in the combo box. When you start entering text in the combo box text field, with every character you type, the choices are reduced to ones which contains the sequence of characters typed in the text field.

2.19 Viewing CodeWarrior Plug-ins

To view the currently installed CodeWarrior features and the associated plug-ins:
1. Select Help > About Freescale CodeWarrior from the IDE menu bar.

The About CodeWarrior Development Studio dialog box appears.
IDE Extensions

Viewing CodeWarrior Plug-ins

Figure 2.61 About CodeWarrior Development Studio Dialog Box

2. Click the Installation Details button.
   The CodeWarrior Development Studio Installation Details dialog box appears.

3. Select the Features tab.
   The features data appears.
4. Select a feature and click the **Plug-in Details** button to view the list of plug-ins associated with the selected feature.

For example, if you select the Eclipse TM Project feature and click the **Plug-in Details** button, the **Feature Plug-ins** dialog box containing the list of plug-ins associated with the Eclipse TM Project feature appears.
Figure 2.63 Feature Plug-ins Dialog Box
This chapter explains how to work with the debugger to control program execution. This chapter explains:

- 3.1 About Debugger
- 3.2 Breakpoints
- 3.3 Build While Debugging
- 3.4 Cache View
- 3.5 CodeWarrior Debugger Settings
- 3.6 Core Index Indicators in Homogeneous Multicore Environment
- 3.7 Debug Perspective
- 3.8 Debug View
- 3.9 Disassembly View
- 3.10 Environment Variables in Launch Configuration
- 3.11 Flash Programmer
- 3.12 Flash File to Target
- 3.13 Hardware Diagnostics
- 3.14 Import/Export/Fill Memory
- 3.15 Launch Group
- 3.16 Load Multiple Binaries
- 3.17 Memory View
- 3.18 Memory Browser View
- 3.19 Memory Management Unit Configurator
- 3.20 Multicore Debugging
- 3.21 Multicore Groups
- 3.22 Multicore Reset
- 3.23 Path Mappings
- 3.24 Redirecting Standard Output Streams to Socket
- 3.25 Refreshing Data During Run Time
A debugger controls program execution and shows the internal operation of a computer program. You can use the debugger to find problems while the program executes and observe how a program uses memory to complete tasks.

These tasks can be performed using the CodeWarrior debugger:

- attach to a running process,
- manipulate the contents of cache, registers, and memory,
- change the program-counter (PC) value,
- execute debugger commands from a command line interface,
- connect to target hardware or simulators,
- render the same data in different formats or byte ordering,
- perform hardware diagnostics,
- program the flash memory,
- manipulate target memory, and
- configure target-hardware subsystems.

### 3.2 Breakpoints

You use a breakpoint to halt program execution on a particular line of source code. Once execution halts, you can examine your program's current state and check register and variable values. You can also change these values and alter the flow of normal program execution. Setting breakpoints helps you debug your program and verify its efficiency.
The types of breakpoints are:

- Regular — Halts the program execution.
- Conditional — Halts the program execution when a specified condition is met.
- Special — Halts the program execution and then removes the breakpoint that caused the halt.

Breakpoints have enabled and disabled states. The following table defines these states.

### Table 3.1 Breakpoint States

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

### 3.2.1 Breakpoints View

The **Breakpoints** view lists all the breakpoints set in the workbench projects. This view also allows breakpoints to be grouped by type, project, file, or working sets, and supports nested groupings. If you double-click a breakpoint displayed by this view, the source code editor displays the source code statement on which this breakpoint is set.

Select **Window > Show View > Breakpoints** from the IDE menu bar to open the **Breakpoints** view.

**TIP** Alternatively, press the **Alt+Shift+Q, B** key combination to open the **Breakpoints** view.

### Figure 3.1 Breakpoints View

![Breakpoints View Image]
3.2.2 Breakpoint Annotations

This CodeWarrior feature enables to change editor breakpoint annotations.

To change breakpoint annotations:
1. Select Window > Preferences.  
   The Preferences dialog box appears.
2. Select General > Editors > Text Editors > Annotations.  
   The annotations appear in the right panel of the Preferences dialog box.

Figure 3.2 Breakpoint Annotations

3. Select Breakpoints (org.eclipse.debug.core.breakpoint) from the Annotation types list box.
4. Specify settings for the selected annotation.
5. Click Apply.
6. Click OK.

You have changed editor breakpoint annotations.

3.2.3 Regular Breakpoints

Regular breakpoints suspend the execution of a thread before a line of code or method is executed.  
Regular breakpoints include:
Breakpoints

- Line — Suspends the thread execution when the line of code it applies to is executed.
- Method — Suspends the thread execution when the method that it applies to, is entered or exited (or both).

The following figure shows an editor window and the marker bar to the left of the source code. Breakpoint icons appear in this marker bar.

**Figure 3.3 Setting Regular Breakpoints**

**NOTE** You can add a breakpoint while debugging your project. Double-click the marker bar to the left of a source code line to set a breakpoint at that line.

### 3.2.3.1 Setting Line Breakpoint

Line breakpoints are set on an executable line of a program. To set a line breakpoint at a line of source code:

1. Open the source code file in the editor and place the cursor on the line where you want to set the breakpoint.
2. Select Run > Toggle Line Breakpoint from the IDE menu bar. You can also double-click on the marker bar next to the source code line.

A breakpoint appears in the **Breakpoints** view. A breakpoint icon appears on the marker bar, directly to the left of the line where you added the breakpoint. The line where the breakpoint is set is highlighted in the editor area. The line appears highlighted in the C/C++ perspective also.

When the breakpoint is enabled, the thread execution suspends before that line of code executes. The debugger selects the suspended thread and displays its stack frames.
3.2.3.2 Setting Method Breakpoint

Method breakpoints are set on methods that do not have source code. To set a method breakpoint on a line of source code:

1. Open the source code file in the editor.
2. Select Window > Show View > Outline from the IDE menu bar.
   
   The Outline view displays an outline of the structured elements of the C/C++ file that is currently open in the editor area.
3. Select the method where you want to add a breakpoint.
4. Select Run > Toggle Breakpoint from the IDE menu bar. You can also select Toggle Breakpoint, from the context menu.

   A breakpoint appears in the Breakpoints view. A breakpoint appears on the marker bar in the file's editor for the method that was selected, if source code exists for the class.

When the breakpoint is enabled, thread execution suspends before the method enters or exits.

3.2.4 Special Breakpoints

A special breakpoint is different from a regular breakpoint. A special breakpoint can be one of these types:

- Hardware — Hardware breakpoints are implemented by the processor hardware. The number of hardware breakpoints available varies by processor type.
- Software — Software breakpoints are implemented by replacing some code in the target with special opcodes. These opcodes stop the core as soon as they are executed. Software breakpoints only work if the code is running out of RAM. There is no restriction on the number of software breakpoints in a project.

Special breakpoints have enabled and disabled states. The following table describes these states.
3.2.4.1 Setting Special Breakpoint

A special breakpoint is not a regular breakpoint and therefore, cannot be set by double clicking. Select Set Special Breakpoints > Software or Hardware from any of the following views to set a special breakpoint.

- Editor — From the context menu of the Editor ruler.
- Disassembly — From the context menu of the Disassembly ruler.
- Outline — From the context menu of the selected C++ class method.

**TIP** To add a special breakpoint while debugging your project, right-click the marker bar to the left of a source code line and select Special Breakpoints > Software or Hardware from the context menu.

3.2.5 Breakpoint Persistence

This CodeWarrior debugger feature preserves the enable/disable state of a breakpoint instance. If the user changes the enabled/disable state of the breakpoint instances, the debugger now serializes the breakpoint state for all breakpoint instances. As a result, next time the user starts a new debug session, the breakpoint instances are created as enabled/disabled according to the serialized state.
3.2.6 Working with Breakpoints

This topic describes the following:

- 3.2.6.1 Modify Breakpoint Properties
- 3.2.6.2 Restricting Breakpoints to Selected Targets and Threads
- 3.2.6.3 Limiting New Breakpoints to Active Debug Context
- 3.2.6.4 Grouping Breakpoints
- 3.2.6.5 Disabling Breakpoints
- 3.2.6.6 Enabling Breakpoints
- 3.2.6.7 Removing Breakpoints
- 3.2.6.8 Removing All Breakpoints
- 3.2.6.9 Undo Delete Breakpoint
- 3.2.6.10 Redo Delete Breakpoint
- 3.2.6.11 Skipping All Breakpoints

3.2.6.1 Modify Breakpoint Properties

To view or modify breakpoint properties for a breakpoint using the Properties for dialog box. You can open the Properties for dialog box using one of the following methods:

- From the Breakpoint view - right-click and select Properties from the context menu.
- From the editor area - right-click on breakpoint and select Breakpoint Properties from the context menu.
The following figure shows the Properties for dialog box. The following table describes each breakpoint property.

**Figure 3.5  Properties for Dialog Box**

![Properties for dialog box](image)

**Table 3.3  Breakpoint Properties**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions</td>
<td>Enables you to attach one or more breakpoint actions to a single breakpoint. For example, when a breakpoint is encountered you could both log a message and play a sound. Actions are executed in the order they appear in the <strong>Actions for this breakpoint</strong> table.</td>
</tr>
<tr>
<td>Common</td>
<td>Displays common properties of a breakpoint. Additionally, you can define a condition that determines when the breakpoint will be encountered. A condition for a breakpoint can be any logical expression that returns true or false value.</td>
</tr>
</tbody>
</table>
3.2.6.2 Restricting Breakpoints to Selected Targets and Threads

You can restrict a breakpoint to one or more threads of a target. This process enables you to work on selected threads of a target.

To restrict a breakpoint to one or more process threads:

1. Select the Breakpoints view.
2. Right-click on the breakpoint you want to restrict and select Properties from the context menu.
   
   The Properties for dialog box appears.
3. From the left panel, select Filtering.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtering</td>
<td>Enables you to restrict the breakpoint to the selected targets and threads.</td>
</tr>
<tr>
<td>Instances</td>
<td>Displays real-time breakpoint information that helps identify the address and the way a breakpoint is installed on a target.</td>
</tr>
</tbody>
</table>
4. From the **Restrict to Selected Targets and Threads** list, select the check boxes adjacent to threads you want to restrict the breakpoint.

5. Click **OK**.

The breakpoint is applied to the selected targets and threads.

### 3.2.6.3 Limiting New Breakpoints to Active Debug Context

If a breakpoint is set in a file shared by multiple cores; the breakpoint is set for all cores by default. To enable limiting new breakpoints on an active debug context:

1. Debug your project.
2. Select the **Breakpoints** view.
3. Click the **Limit New Breakpoints to Active Debug Context** icon from the **Breakpoints** view.
4. Double-click the marker bar to the left of a source code line to set a breakpoint at that line.
NOTE If no debug context exists, the breakpoint is installed in all contexts as normal.

Once set, the breakpoint filtering is maintained for the individual context during a Restart but is lost after a Terminate. After a Terminate, the breakpoint is installed in all debug contexts.

3.2.6.4 Grouping Breakpoints

Grouping breakpoints helps you view a list of breakpoints that match specified criteria.

To group breakpoints:
1. Click the Breakpoints view pull-down menu.

Figure 3.7 Grouping Breakpoints

2. Select Group By.
   - Breakpoints — Displays a standard list of breakpoints.
   - Breakpoint types — Groups all breakpoints by their types.
   - Breakpoint Working Sets — Groups all breakpoints as user defined problem-specific sets that can be quickly enabled and disabled.
   - Files — Groups all breakpoints by the files they are set in.
   - Projects — Groups all breakpoints by the project in which they are set.
   - Resource Working Sets — Groups all breakpoints into resource-specific working sets that can be quickly enabled and disabled.
   - Advanced — Displays the Group Breakpoints dialog box that enables you to specify nested grouping for the Breakpoints view. For example, you group breakpoints by Breakpoint Types and then group them by Projects and Working Sets.

3. Select appropriate group by command.
3.2.6.5 Disabling Breakpoints

Disabling a breakpoint prevents it from affecting program execution and is easier than clearing or creating new breakpoints.

**NOTE** Disabled breakpoints can be enabled without losing any information. To enable breakpoints, refer to 3.2.6.6 Enabling Breakpoints.

To disable a breakpoint:
1. Right-click on an enabled breakpoint in the marker bar.
2. Select **Disable Breakpoint** from the context menu.
   The breakpoint icon changes to . The disabled breakpoint icon indicates that the breakpoint does not halt program execution.

3.2.6.6 Enabling Breakpoints

The program execution suspends whenever an enabled breakpoint is encountered in the source code. Enabling a breakpoint is easier than clearing or creating a new breakpoint.

**NOTE** Enabled breakpoints can be disabled without losing any information. To disable breakpoints, refer to the topic 3.2.6.5 Disabling Breakpoints.

To disable a breakpoint:
1. Right-click on a disabled breakpoint in the marker bar.
2. Select **Enable Breakpoint** from the context menu.
   The breakpoint icon changes to . The enabled breakpoint icon indicates that it suspends the program execution whenever encountered in the source code.

3.2.6.7 Removing Breakpoints

To remove a breakpoint:
1. Right-click on a breakpoint in the **Breakpoint** view.
2. Select the **Remove Selected Breakpoints** from the context menu.
   The selected breakpoint is removed.

**NOTE** Alternatively, click the **Remove Selected Breakpoints** icon in the **Breakpoints** view.
3.2.6.8 Removing All Breakpoints

To remove all breakpoints:
1. Right-click in the Breakpoints view.
2. Select Remove All Breakpoints from the context menu.

**NOTE** Alternatively, click the Remove All Breakpoints icon in the Breakpoints view.

3.2.6.9 Undo Delete Breakpoint

This feature allows you to undo delete breakpoints from breakpoints view. It is useful when by mistake you have deleted a breakpoint with some elaborated conditions. To undo delete breakpoint:
1. Select Edit > Undo Delete Breakpoint from the IDE menu bar.

3.2.6.10 Redo Delete Breakpoint

To redo delete breakpoint:
1. Select Edit > Redo Delete Breakpoint from the IDE menu bar.

3.2.6.11 Skipping All Breakpoints

To ignore all active breakpoints, click the Skip All Breakpoints icon. All active breakpoints are skipped by the debugger during program/source code execution.

**NOTE** Skipped breakpoints do not suspend execution until they are activated.

Click the Skip All Breakpoints icon to re-activate all breakpoints.

3.2.7 Breakpoint Actions

This topic explains CodeWarrior enhancements to standard breakpoint behavior. While the standard behavior of breakpoints of a debugger is to stop execution at a specific spot, you can use breakpoint actions to extend the breakpoint behavior and define other actions that occur when program execution reaches the breakpoint.

Breakpoint actions let you:
- specify specific tasks to perform,
- manage a list of actions, where each action has specific properties,
• attach specific actions to individual breakpoints,
• control the order in which the breakpoint actions occur, and
• execute the Debugger Shell commands.

You can associate more than one action with a breakpoint. The debugger executes the associated breakpoint actions when the program execution encounters the breakpoint. The following table lists and describes breakpoint actions.

### Table 3.4 Breakpoint Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger Shell Action</td>
<td>Executes Debugger Shell commands or a Debugger Shell script.</td>
</tr>
<tr>
<td>Sound Action</td>
<td>Plays the specified sound.</td>
</tr>
<tr>
<td>Log Action</td>
<td>Logs messages to a console. The messages can be literal strings or the result of an expression that the debugger evaluates.</td>
</tr>
<tr>
<td>Resume Action</td>
<td>Halt the program execution for a specified time and then resumes the program execution.</td>
</tr>
<tr>
<td>External Tool Action</td>
<td>Invokes a program, which is external to the debugger.</td>
</tr>
</tbody>
</table>

### 3.2.7.1 Breakpoint Actions Preferences Page

You use the Breakpoint Actions preferences page to manage a global list of breakpoint actions available in a workspace. Each action is an instance of a specific action type.

The Breakpoint Actions preferences page:
• shows a list of available actions,
• lets you create new actions for a selected action type, and
• lets you add, edit, and delete existing actions.

To open the Breakpoint Actions preferences page:
1. Select Window > Preferences from the IDE menu bar.
   The Preferences dialog box appears.
2. Select C/C++ > Debug > Breakpoint Actions.
   The Breakpoint Actions preferences page appears.
3.2.7.2 Adding Breakpoint Action

To add a breakpoint action:

1. Open the Breakpoint Actions preferences page.
2. Click New.
   - The New Breakpoint Action dialog box appears.
3. In the **Action name** text box, enter a name for the new action.

4. Use the **Action type** drop-down list to select the type of action you want to create.

5. Specify additional breakpoint-action properties, according to the action type that you specified.

   For example, to display a specified log message when the debugger encounters a breakpoint, specify the log message in the **Log Action** breakpoint action.

6. Click **OK**.

   The **New Breakpoint Action** dialog box closes. The new breakpoint action appears in the **Breakpoint Actions** preferences page table.

### 3.2.7.3 Attaching Breakpoint Actions to Breakpoints

To use a breakpoint action, you must attach it to an existing breakpoint.

**NOTE** To attach breakpoint actions to a breakpoint, add the associated breakpoint actions in the **Breakpoint Actions** preference page.

To attach breakpoint actions to a breakpoint:

1. Initiate a debugging session.

2. In the editor area of the **Debug** perspective, set a breakpoint.

3. Open the Breakpoints view.
Debugger

Breakpoints

a. From the IDE menu bar, select Window > Show View.
   The Show View dialog box appears.
b. Select Debug > Breakpoints.
c. Click OK.
4. In the Breakpoints view, right-click on a breakpoint.
5. Select Properties from the context menu.
   The Properties for dialog box appears.
6. Select Actions from the left panel of the Properties for dialog box.
   The Actions page appears.
7. Follow these sub-steps for each breakpoint action that you want to attach to the breakpoint:
   a. Select the breakpoint action from the Available actions table.
   b. Click Attach.
      The selected breakpoint action moves from the Available actions table to the Actions for this breakpoint table.

NOTE The debugger executes the breakpoint actions in the order shown in the Actions for this breakpoint table.

8. To reorder the breakpoint actions in the Actions for this breakpoint table:
   a. Select the action in the table.
   b. Click Up to move the selected action up in the table.
   c. Click Down to move the selected action down in the table.

During a debugging session, the debugger executes the breakpoint actions when the breakpoint is encountered.

3.2.8 Selecting Breakpoint Template

When you set a line or function breakpoint in the template code from the IDE, the breakpoint is set on all template instances. This feature allows you to enable or disable a breakpoint for a particular core.
To disable breakpoint for a particular core:
1. Initiate a debugging session.
2. Open the Breakpoints view.
3. Click on the + sign to expand a breakpoint.
4. Clear the check box for the core for which you do not want the breakpoint applied.

### 3.3 Build While Debugging

The debug session locks the debugged elf when Create and Use Copy of Executable option is not checked, see 3.30 Symbolics. If you make changes to the source files and rebuild the project while a debug session is on, the build commands are invoked but the locked files are not overwritten and a link-time error is generated.

This feature enables automatic termination of the debug sessions when initiating a build that produces executables locked by those debug sessions.

To enable build while debugging:

1. Select **Window > Preferences** from the IDE menu bar.
   
   The **Preferences** dialog box appears.

2. Select **C/C++ > Debug > CodeWarrior Debugger**.

   CodeWarrior debugger preferences appears in the right-panel of the **Preferences** dialog box.
3. Check the **Automatically terminate debug session on project rebuild, don’t ask** check box.
4. Click **Apply**.
5. Click **OK**.

**NOTE** Applying this setting immediately effects the project.

Now the debug session will automatically terminate while initiating a build that produces executables locked by those debug sessions

### 3.4 Cache View

**Cache** view helps you view, modify, and control a hardware cache. Use the **Cache** view to examine instruction and data cache for L1 and L2 cache for the supported targets.
3.4.1 Opening Cache View

To open the Cache view:
1. Start a debugging session.
2. From the CodeWarrior menu bar, select Window > Show View > Other.
   The Show View dialog box appears.

Figure 3.12 Show View Dialog Box

3. Expand the Debug group and Select Cache.
4. Click OK.
   The Cache view appears.

TIP  Alternatively, start typing Cache in the type filter text text box. The Show View dialog box filters the list of the views and displays only the views matching the characters typed in the text box. Select Cache from the filtered list and click OK.
NOTE  If the Select Cache drop-down list is grayed out in the Cache view, then the current target does not support viewing cache.

3.4.2 Preserving Sorting

To preserve sorting of the cache when you update and refresh the cache:
1. Start a debugging session.
2. Open the Cache view. For more information about how to open the Cache view, refer 3.4.1 Opening Cache View.
3. Select the Preserve Sorting command from the Cache view pull-down menu.
NOTE  This option is disabled by default. If enabled, every operation that triggers cache refresh, such as step, and run to breakpoint will have to wait for the cache data loading and sorting.

3.4.3 Cache View Drop-down Menu

You can perform various actions on the cache using the Cache view drop-down menu. Alternatively, you can use the Cache view toolbar that includes the same commands as the Cache view pull-down menu. The following table lists the Cache view drop-down menu commands and their description.

Table 3.5  Cache View Drop-down Menu Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Commits changes in the Cache view to the cache register of the target hardware, if supported by the target hardware.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Reads data from the target hardware and updates the cache display.</td>
</tr>
</tbody>
</table>
### Table 3.5 Cache View Drop-down Menu Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidate</td>
<td>Invalidates the entire content of the cache.</td>
</tr>
<tr>
<td>Flush</td>
<td>Flushes the entire content of the cache.</td>
</tr>
<tr>
<td></td>
<td>Flushing the cache involves committing uncommitted data to the next level of the memory hierarchy, and then invalidating the data within the cache.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the cache. Locking cache prevents the cache from fetching the new lines or discarding the current valid lines.</td>
</tr>
<tr>
<td>Synchronize</td>
<td>Synchronize cache data with memory data.</td>
</tr>
<tr>
<td>Enable</td>
<td>Turns on the cache.</td>
</tr>
<tr>
<td>Disable LRU</td>
<td>Removes the Least Recently Used attribute from the existing display for each cache line.</td>
</tr>
<tr>
<td>Inverse LRU</td>
<td>Displays the inverse of the Least Recently Used attribute for each cache line.</td>
</tr>
<tr>
<td>Copy Cache</td>
<td>Copies the cache contents to the system clipboard.</td>
</tr>
<tr>
<td>Export Cache</td>
<td>Exports the cache contents to a file.</td>
</tr>
<tr>
<td>Search</td>
<td>Finds an occurrence of a string in the cache lines.</td>
</tr>
<tr>
<td>Search Again</td>
<td>Finds the next occurrence of a string in the cache lines.</td>
</tr>
<tr>
<td>Preserves sorting</td>
<td>Preserves sorting of the cache when the cache data is updated and the cache is refreshing.</td>
</tr>
<tr>
<td></td>
<td>This option is disabled by default. If enabled, every operation that triggers cache refresh (like step, run to breakpoint) will have to wait for cache data loading and sorting.</td>
</tr>
<tr>
<td>View Memory</td>
<td>Views the corresponding memory for the selected cache lines.</td>
</tr>
<tr>
<td>Lock Line</td>
<td>Locks the selected cache lines.</td>
</tr>
<tr>
<td>Invalidate Line</td>
<td>Invalidates the selected cache lines.</td>
</tr>
<tr>
<td>Flush Line</td>
<td>Flushes the entire contents of the selected cache lines.</td>
</tr>
<tr>
<td>Synchronize Line</td>
<td>Synchronize selected cache data with memory data.</td>
</tr>
</tbody>
</table>
A CodeWarrior project can have multiple associated launch configurations. A launch configuration is a named collection of settings that the CodeWarrior tools use. For example, the project you created in the tutorial chapter had two associated launch configurations.

The CodeWarrior project wizard generates launch configurations with names that follow the pattern `projectname - configtype - targettype`, where:

- `projectname` represents the name of the project.
- `configtype` represents the type of launch configuration.
- `targettype` represents the type of target software or hardware on which the launch configuration acts.

Launch configurations for debugging code lets you specify settings such as:

- Files that belong to the launch configuration
- Behavior of the debugger and related debugging tools

If you use the CodeWarrior wizard to create a new project, the IDE creates two debugger-related launch configurations:

- `Debug` configuration that produces unoptimized code for development purposes.
- `Release` configuration that produces code intended for production purposes.

### 3.5.1 Modifying Debugger Settings

If you use the CodeWarrior wizard to create a new project, the IDE sets the debugger settings to default values. You can modify these settings as per the requirement.

To change debugger settings:

---

**Table 3.5 Cache View Drop-down Menu Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock Way</td>
<td>Locks the cache ways specified with the Lock Ways menu command.</td>
</tr>
<tr>
<td></td>
<td>Locking a cache way means that the data contained in that way must not change. If the cache needs to discard a line, it will not discard the locked lines, such as the lines explicitly locked, or the lines belonging to locked ways.</td>
</tr>
<tr>
<td>Unlock Way</td>
<td>Unlocks the cache ways specified with the Lock Ways command.</td>
</tr>
<tr>
<td>Lock Ways</td>
<td>Specifies the cache ways on which the Lock Way and Unlock Way commands operate.</td>
</tr>
</tbody>
</table>
Debugger

CodeWarrior Debugger Settings

1. In the CodeWarrior Project View explorer, right click on the project folder.
   A context menu appears.

2. Select **Debug As > Debug Configurations**.
   The **Debug Configurations** dialog box appears.
   The left panel of the **Debug Configurations** dialog box lists the debug configurations that apply to the current project.

3. Expand the **CodeWarrior** configuration.

4. From the expanded list, click the name of the debug configuration you want to modify.
   The **Debug Configurations** dialog box shows the settings for the selected configuration.

5. Click the **Debugger** tab.
   The **Debugger** page appears.

6. Change the settings in this page to suit your needs.

7. Click the **Apply** button.
   The IDE saves your new settings.

**NOTE**

You can select other pages and modify their settings. When you finish, you can click the **Debug** button to start a new debugging session, or click the **Close** button to save your changes and close the **Debug Configuration** dialog box.

---

**3.5.2 Reverting Debugger Settings**

You can revert pending changes and restore last saved settings. To undo pending changes, click the **Revert** button at the bottom of the **Debug Configurations** dialog box.

The IDE restores the last set of saved settings to all pages of the **Debug Configurations** dialog box. Also, the IDE disables the **Revert** button until you make new changes.

**3.5.3 Stopping Debugger at Program Entry Point**

This feature enables you to specify debugger settings for the CodeWarrior Debugger to remain stopped at program entry point.

To specify debugger settings to stop debugger at program entry point:

1. In the CodeWarrior Project View explorer, right click on the project folder.
   A context menu appears.
2. Select **Debug As** > **Debug Configurations**.
   The **Debug Configurations** dialog box appears. The left panel of the **Debug Configurations** dialog box lists the debug configurations that apply to the current project.

3. Expand the **CodeWarrior** configuration.

4. From the expanded list, click the name of the debug configuration you want to modify.
   The **Debug Configurations** dialog box shows the settings for the selected configuration.

5. Click the **Debugger** tab.
   The **Debugger** page appears.

6. Check the **Stop on startup at** check box.
   The Program entry point and the **User Specified** options are enabled.

7. Select the **Program entry point** option.

   **NOTE** To stop the debugger at a user-specified function, select the **User specified** option and type the function name in the text box.

8. Click **Apply**.
   The IDE saves the settings for the debugger to remain stopped at program entry point.

### 3.6 Core Index Indicators in Homogeneous Multicore Environment

This feature enables you to identify the core(s) being debugged, when you debug a target with two or more cores of the same architecture.

The core index is displayed in these three views:

- **Debug view**
  For information on how the core index is displayed in the **Debug** view, see the <product> Targeting Manual.

- **3.6.1 System Browser View**
- **3.6.2 Console View**

#### 3.6.1 System Browser View

The **System Browser** serves these two types of debug sessions:

- **3.6.1.1 Kernel Awareness**
- **3.6.1.2 OS Application**
3.6.1.1 Kernel Awareness

In a Kernel Awareness debug session, the core index is displayed under these scenarios:

- Multiple homogeneous cores, each running a single core Operating System (OS)
- Multicore OS

The System Browser view displays content for the active debug context. For Kernel Awareness, the label of the process object, as shown in the Debug view, is displayed at the top of the System Browser view's client area. This label contains the index of the core the OS is running on, and is referred to as the context label.

For example, if the user is performing Kernel Awareness on a P4080 target, and the user is looking at Linux running on the 5th e500 core, then the top of the System Browser client area shows a label that contains core 4.

In a multicore OS scenario, the system browser shows kernel threads for all cores being managed by the OS. The System Browser view that displays kernel threads indicates the core index for each thread.

The following figure shows the core index information for kernel threads in a multicore environment.

Figure 3.15 System Browser View — Kernel Awareness

3.6.1.2 OS Application

OS Application debugging happens through a connection with an agent running on the OS. The connection to the agent is through TCP/IP or COM port. In this scenario, the agent does not have information about the core it is running on, nor does the user specify it when configuring the launch. The user simply specifies the IP address or COM port where the agent is running.

The System Browser view shows the IP address or COM port in the context label.
3.6.2 Console View

The console associated with a process object displays the label of that process, as it appears in the **Debug** view. When debugging a homogeneous multicore target, this label contains the core index.

The following figure shows the core index in the **Console** view.

![Figure 3.17 Console View](image)

3.7 Debug Perspective

A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources.

The **Debug** perspective lets you manage how the Workbench debugs and runs a program. You can control your program’s execution by setting breakpoints, suspending launched programs, stepping through your code, and examining the values of variables.

The Debug perspective displays this information:

- The stack frame of the suspended threads of each target that you are debugging
Debugger

Debug View

- Each thread in your program represented as a node in the tree
- The process of each program that you are running

The Debug perspective also drives the Source view. As you step through your program, the Source view highlights the location of the execution pointer.

The following figure shows a Debug perspective.

Figure 3.18 Debug Perspective

3.8 Debug View

Views support editors and provide alternate presentations as well as ways to navigate the information in your Workbench. The Debug view (Figure 3.19) shows the target debugging information in a tree hierarchy. For more information on the tree hierarchy and target debugging information, see the C/C++ Development User Guide.

Use the Debug view to perform these tasks:
- Clear all terminated processes.
- Start a new debug session for the selected process.
- Resume execution of the currently suspended debug target.
• Halt execution of the currently selected thread in a debug target.
• Terminate the selected debug session and/or process.
• Detach the debugger from the selected process.
• Execute the current line, including any routines, and proceed to the next statement.
• Execute the current line, following execution inside a routine.
• Re-enter the selected stack frame.
• Examine a program as it steps into disassembled code.

For more information on debugging actions, see 3.8.1 Common Debugging Actions.

Figure 3.19 Debug View

3.8.1 Common Debugging Actions

This topic explains how to perform common debugging actions that correct source-code errors, control program execution, and observe memory behavior. The common debugging actions are:

• 3.8.1.1 Starting the Debugger
• 3.8.1.2 Stepping Into Routine Call
• 3.8.1.3 Stepping Out of Routine Call
• 3.8.1.4 Stepping Over Routine Call
• 3.8.1.5 Stopping Program Execution
• 3.8.1.6 Resuming Program Execution
• 3.8.1.7 Running a Program
• 3.8.1.8 Disconnecting a Core
• 3.8.1.9 Restarting Debugger
• 3.8.1.10 Debugging in Instruction Stepping Mode
3.8.1.1 Starting the Debugger

Use the debug command in the 4.4 Command-Line Debugger Shell to begin a debugging session. The debugger then takes control of program execution, starting at the main entry point of the program.

Select Run > Debug or click the Debug button (shown at left) in the 3.8 Debug View toolbar to start the debugger.

The IDE opens a new 3.8 Debug View.

NOTE Some projects require additional configuration before a debugging session can begin. For more information, refer the <product> Targeting Manual.

3.8.1.2 Stepping Into Routine Call

Use the step command in the 4.4 Command-Line Debugger Shell to execute one source-code statement at a time and follow execution in a routine call.

Select Run > Step Into or click the Step Into button (shown at left) in the 3.8 Debug View toolbar to step into a routine.

After the debugger executes the source-code statement, the current-statement arrow moves to the next statement. The debugger uses these rules to find the next statement:

- 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 in the calling routine.

3.8.1.3 Stepping Out of Routine Call

Use the Step Return command in the 4.4 Command-Line Debugger Shell 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 call chain.

Select Run > Step Return or click the Step Return button (shown at left) in the 3.8 Debug View toolbar to step out of a routine.

The current routine executes and returns to its caller; then program execution stops.
3.8.1.4 Stepping Over Routine Call

Use the \texttt{next} command in the \underline{4.4 Command-Line Debugger Shell} 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 it reaches:

- end of the called routine,
- breakpoint,
- breakpoint,
- or an eventpoint that stops execution.

Select \texttt{Run > Step Over} or click the \texttt{Step Over} button (shown at left) in the \underline{3.8 Debug View} toolbar to step over a routine.

The current statement or routine executes; then program execution stops.

3.8.1.5 Stopping Program Execution

Use the \texttt{kill} command in the \underline{4.4 Command-Line Debugger Shell} to stop program execution during a debugging session.

Select \texttt{Run > Terminate} or click the \texttt{Terminate} button (shown at left) in the \underline{3.8 Debug View} toolbar to stop program execution.

The operating system surrenders control to the debugger, which stops the program execution.

\underline{NOTE} When working with a processor that has multiple cores, you can choose \texttt{Run > Multicore Terminate} to stop selected group of cores.

3.8.1.6 Resuming Program Execution

Use the \texttt{go} command in the \underline{4.4 Command-Line Debugger Shell} to resume execution of a suspended debugging session.

Select \texttt{Run > Resume} or click the \texttt{Debug} button (shown at left) in the \underline{3.8 Debug View} toolbar to resume program execution.

The suspended session resumes.

3.8.1.7 Running a Program

Use the \texttt{run} command in the \underline{4.4 Command-Line Debugger Shell} to execute a program outside of the debugger control.
Select Run > Run or click the Run button (shown at left) in the 3.8 Debug View toolbar to begin program execution.

The program runs outside of debugger control. Further, any watchpoints and breakpoints (special, hardware, and software) are not hit.

**NOTE** The run command is shortcut for debug, go, and disconnect actions. The run command downloads the code to the target, puts the core in running mode, and then disconnects from the target.

### 3.8.1.8 Disconnecting a Core

Click the Disconnect button (shown at left) in the 3.8 Debug View toolbar to disconnect a core from the target.

The effect of the disconnect command is same as of the terminate command. The only difference between the two commands is that the disconnect command activates when a debug session is running.

### 3.8.1.9 Restarting Debugger

You can restart debug session in following ways:

- Use the restart command in the 4.4 Command-Line Debugger Shell after stopping program execution. The debugger goes back to the beginning of the program and begins execution again.
- Right-click Thread in the Debug View and select Restart option from the context menu.
- Click the Restart button in the Debug View toolbar to restart debug session.

**NOTE** Restart action is considerable faster to re-launch a debug session as it skips over loading executable debug information and target register descriptors.

### 3.8.1.10 Debugging in Instruction Stepping Mode

Use the stepi command in the 4.4 Command-Line Debugger Shell to debug a program in instruction stepping mode. In this mode, you can debug the program in 3.9 Disassembly View instead of the source view.

You can also switch to instruction stepping mode by clicking the Instruction Stepping Mode button (shown at left) in the 3.8 Debug View toolbar.
3.8.1.11 Changing the Program Counter Value

To change the program-counter value:

1. Initiate a debugging session.
2. In the editor view, place the cursor on the line you want the debugger to execute.
3. Right-click on the line.
   A context menu appears.
4. From the context menu, select Move To Line.
   The debugger moves the program counter to the location you specified. The editor view shows the new location.

**CAUTION** Changing the program-counter value because doing so can cause your program to malfunction. For example, if you set the program counter outside the address range of the current function, the processor will skip the instructions that clean up the stack and return execution to the correct place in the calling function. Your program will then behave in an unpredictable way.

3.9 Disassembly View

The **Disassembly** view (Figure 3.20) shows the loaded program as assembly language instructions mixed with source code for comparison. The next instruction to be executed is indicated by an arrow marker and highlighted in the view.

You can perform these tasks in the **Disassembly** view:

- Set breakpoints at the start of any assembly language instruction
- Enable and disable breakpoints and set their properties
- Step through the disassembled instructions of your program
- Jump to specific instructions in the program
3.10 Environment Variables in Launch Configuration

CodeWarrior enables you to use environment or eclipse variables to specify the path of the launch executable.

To specify an environment or eclipse variable:

1. Click Run > Debug Configuration.

   The Debug Configurations dialog box appears.

2. Select a launch configuration from the left-panel of the Debug Configurations dialog box.

3. Click Variables.

   The Select build variable dialog box appears.
4. Select a variable from the variable list.
5. Click **OK**.

Now you can use environment or eclipse variables to specify the path of the launch executable.

### 3.11 Flash Programmer

Flash programmer is a CodeWarrior plug-in that lets you program the flash memory of the supported target boards from within the IDE. The flash programmer can program the flash memory of the target board with code from a CodeWarrior IDE project or a file. The flash programmer enables you to perform following actions on a flash device:

- 3.11.2.3.1 Erase/Blank Check Actions.
- 3.11.2.3.2 Program/Verify Actions.
- 3.11.2.3.3 Checksum Actions.
- 3.11.2.3.4 Diagnostics Actions.
- 3.11.2.3.5 Dump Flash Actions.
- 3.11.2.3.6 Protect/Unprotect Actions.
- 3.11.2.3.7 Secure/Unsecure Actions.
The flash programmer runs as a target task in the Eclipse IDE. To program the flash memory on a target board, you need to perform the following tasks:

- 3.11.1 Create a Flash Programmer Target Task
- 3.11.2 Configure the Flash Programmer Target Task
- 3.11.3 Run Flash Programmer Target Task

**NOTE**  Click the Save button or press CTRL + s to save task settings.

### 3.11.1 Create a Flash Programmer Target Task

1. Select **Windows > Show Views > Others** from the IDE menu bar. The Show View dialog box appears.

   **Figure 3.22  Show View Dialog Box**

   ![Show View Dialog Box](image)

2. Expand the **Debug** group and select **Target Tasks**.

3. Click **OK**.

   The **Target Tasks** view appears.
4. Click the Create a new Target Task button in the Target Tasks view toolbar. The Create New Target Task wizard appears.

5. In the Task Name text box, type name for the new flash programming target task.
6. Select a launch configuration from the Run Configuration drop-down list.
   - Select Active Debug Context when flash programmer is used over an active debug session.
   - Select a project-specific debug context when flash programmer is used without an active debug session.
7. Select Flash Programmer from the Task Type drop-down list.
8. Click Finish.

The target task is created and the Flash Programmer Task editor window appears. You use this window to configure the flash programmer target task.

**Figure 3.25 Flash Programmer Task Editor Window**

- Flash Devices — Lists the devices added in the current task.
- Target RAM — Enables you to specify the settings for Target RAM.
- Flash Program Actions — Displays the programmer actions to be performed on the flash devices.

### 3.11.2 Configure the Flash Programmer Target Task

To configure a flash programmer target task, you need to perform the following actions:

- **3.11.2.1 Add a Flash Device**
- **3.11.2.2 Specify Target RAM Settings**
- **3.11.2.3 Add Flash Programmer Actions**

**NOTE** Click the Save button or press CTRL + s to save the task settings.
3.11.2.1 Add a Flash Device

You can add flash devices in the Flash Devices table.
To add a flash device to the Flash Devices table:
1. Click the Add Device button.
   The Add Device dialog box appears.

Figure 3.26 Add Device Dialog Box

```
2. Select a flash device from the device list.
3. Click the Add Device button.
   The flash device is added to the Flash Devices table in the Flash Programmer Task editor window.
```

**NOTE**  You can select multiple flash devices to add to the Flash Devices table. To select multiple devices, hold the CTRL key while selecting the devices.

4. Click Done.
   The Add Device dialog box closes and the flash device appears in the Flash Devices table in the Flash Programmer Task editor window.
NOTE For NOR flashes, the base address indicates the location where the flash is mapped in the memory. For SPI and NAND flashes, the base address is usually 0x0.

3.11.2.2 Specify Target RAM Settings

The Target RAM is used by Flash Programmer to download its algorithms.

NOTE The Target RAM memory area is not restored by flash programmer. If you are using flash programmer with Active Debug Context, it will impact your debug session.

The Target RAM (Figure 3.27) group contains fields to specify settings for the Target RAM.

- **Address** text box — Enables you to specify the address from the target memory. The **Address** text box should contain the first address from target memory used by the flash algorithm running on a target board.
- **Size** text box — Enables you to specify the size of the target memory. The flash programmer does not modify any memory location other than the target memory buffer and the flash memory.
- **Verify Target Memory Writes** check box — Check this check box to verify all write operations to the hardware RAM during flash programming.
3.11.2.3 Add Flash Programmer Actions

In the Flash Programmer Actions group in the Flash Programmer Task editor window (Figure 3.25), you can add following actions on the flash device.

- 3.11.2.3.1 Erase/Blank Check Actions
- 3.11.2.3.2 Program/Verify Actions
- 3.11.2.3.3 Checksum Actions
- 3.11.2.3.4 Diagnostics Actions
- 3.11.2.3.5 Dump Flash Actions
- 3.11.2.3.6 Protect/Unprotect Actions
- 3.11.2.3.7 Secure/Unsecure Actions

The Flash Programmer Actions group contains the following list to work with flash programmer actions.

1. Add Action drop-down list
   - Erase / Blank Check Action — Enables you to add erase or blank check actions for a flash device.
   - Program / Verify Action — Enables you to add program or verify flash actions for a flash device.
   - Checksum Action — Enables you to add checksum actions for a flash device.
   - Diagnostics Action — Enables you to add a diagnostics action.
   - Dump Flash Action — Enables you to add a dump flash action.
   - Protect / Unprotect Action — Enables you to add protect or unprotect action.
   - Secure/Unsecure Action — Enables you to add secure or unsecure action.

2. Duplicate Action button — Enables you to duplicate a flash program action in the Flash Programmer Actions table.

3. Remove Action button — Enables you to remove a flash program action from the Flash Programmer Actions table.

4. Move Up button — Enables you to move up the selected flash action in the Flash Programmer Actions table.

5. Move Down button — Enables you to move down the selected flash action in the Flash Programmer Actions table.

**NOTE** Actions can also be enabled or disabled using the Enabled column. The Description column contains the default description for the flash programmer actions. You can also edit the default description.
3.11.2.3.1 Erase/Blank Check Actions

The erase action enables you erase sectors from the flash device. You can also use the erase action to erase the entire flash memory without selecting sectors. The blank check action verifies if the specified areas have been erased from the flash device.

NOTE  Flash Programmer will not erase a bad sector in the NAND flash. After the erase action a list of bad sectors is reported (if any).

To add an erase/blank check action:

1. Select Erase/Blank Check Action from the Add Action drop-down list.
   The Add Erase/Blank Check Action dialog box appears.

2. Select a sector from the Sectors table and click the Add Erase Action button to add an erase operation on the selected sector.

   NOTE  Press CTRL or SHIFT keys for selecting multiple sectors from the Sectors table.

3. Click the Add Blank Check button to add a blank check operation on the selected sector.
4. Check the **Erase All Sectors Using Chip Erase Command** check box to erase the entire flash memory.

**NOTE** After checking the **Erase All Sectors Using Chip Erase Command** check box, you need to add either erase or blank check action to erase all sectors.

5. Click **Done**.

The **Add Erase / Blank Check Action** dialog box closes and the added erase / blank check actions appear in the **Flash Programmer Actions** table in the **Flash Programmer Task** editor window.

### 3.11.2.3.2 Program/Verify Actions

The program action enables you to program the flash device and the verify action verifies the programmed flash device.

**NOTE** The program action will abort and fail if it is performed in a bad block for NAND flashes.

To add a program/verify action:

1. Select **Program/Verify Action** from the **Add Action** drop-down list.

   The **Add Program/Verify** dialog box appears.

   **Figure 3.29 Add Program/Verify Action Dialog Box**

2. Select the file to be written to the flash device.
   
   - Check the **Use File from Launch Configuration** check box to use the file from the launch (run) configuration associated with the task.
Debugger
Flash Programmer

- Specify the file name in the File text box. You can use Workspace, File System, or Variables buttons to select the desired file.

3. Select the file type from the File Type drop-down list. You can select any one of the following file types:
   - Auto — Detects the file type automatically.
   - Elf — Specifies executable in ELF format.
   - Srec — Specifies files in Motorola S-record format.
   - Binary — Specifies binary files.

4. Check the Erase sectors before program checkbox to erase sectors before program.
5. [Optional] Check the Verify after program checkbox to verify after the program.

**NOTE** The Verify after Program option is available only with processors supporting it.

6. Check the Restricted To Address in the Range check box to specify a memory range. The write action is permitted only in the specified address range. In the Start text box, specify the start address of the memory range sector and in the End text box, specify the end address of the memory range.

7. Check the Apply Address Offset check box and set the memory address in the Address text box. Value is added to the start address of the file to be programmed or verified.

8. Click the Add Program Action button to add a program action on the flash device.

9. Click the Add Verify Action button to add a verify action on the flash device.

10. Click Done.

The Add Program/Verify Action dialog box closes and the added program / verify actions appear in the Flash Programmer Actions table in the Flash Programmer Task editor window.

**3.11.2.3.3 Checksum Actions**

The checksum can be computed over host file, target file, memory range or entire flash memory. To add a checksum action:

1. Select Checksum Action from the Add Action drop-down list.

   The Add Checksum dialog box appears.
2. Select the file for checksum action.
   • Check the Use File from Launch Configuration check box to use the file from the launch (run) configuration associated with the task.
   • Specify the filename in the File text box. You can use the Workspace, File System, or Variables buttons to select the desired file.

3. Select the file type from the File Type drop-down list.

4. Select an option from the Compute Checksum Over options. The checksum can be computed over the host file, the target file, the memory range, or the entire flash memory.

5. Specify the memory range in the Restricted To Addresses in the Range group. The checksum action is permitted only in the specified address range. In the Start text box, specify the start address of the memory range sector and in the End text box, specify the end address of the memory range.

6. Check the Apply Address Offset check box and set the memory address in the Address text box. Value is added to the start address of the file to be programmed or verified.

7. Click the Add Checksum Action button.

8. Click Done.


3.11.2.3.4 Diagnostics Actions

The diagnostics action generates the diagnostic information for a selected flash device.

**NOTE** Flash Programmer will report bad blocks, if they are present in the NAND flash.

To add a diagnostics action:
1. Select Diagnostics from the Add Action drop-down list.

   The Add Diagnostics dialog box appears.
2. Select a device to perform the diagnostics action.
3. Click the Add Diagnostics Action button to add diagnostic action on the selected flash device.

**NOTE** Check the Perform Full Diagnostics check box to perform full diagnostics on a flash device.

4. Click Done.

   The Add Diagnostics Action dialog box closes and the added diagnostics action appears in the Flash Programmer Actions table in the Flash Programmer Task editor window.

### 3.11.2.3.5 Dump Flash Actions

The dump flash action enables you to dump selected sectors of a flash device or the entire flash device.

To add a dump flash action:
1. Select Dump Flash Action from the Add Action drop-down list.
   
   The Add Dump Flash Action dialog box appears.

**Figure 3.32 Add Dump Flash Action Dialog Box**

2. Specify the file name in the File text box. The flash is dumped in this selected file.
3. Select the file type from the **File Type** drop-down list. You can select any one of the following file types:
   - Srec — Saves files in Motorola S-record format.
   - Binary — Saves files in binary file format.

4. Specify the memory range for which you want to add dump flash action.
   - Type the start address of the range in the **Start** text box.
   - Type the end address of the range in the **End** text box.

5. Click the **Add Dump Flash Action** button to add a dump flash action.

6. Click **Done**.

   The **Add Dump Flash Action** dialog box closes and the added dump flash action appear in the **Flash Programmer Actions** table in the **Flash Programmer Task** editor window.

### 3.11.2.3.6 Protect/Unprotect Actions

The protect/unprotect actions enable you to change the protection of a sector in the flash device.

To add a protect/unprotect action:

1. Select the **Protect/Unprotect Action** from the **Add Action** drop-down list.

   The **Add Protect/Unprotect Action** dialog box appears.
2. Select a sector from the Sectors table and click the Add Protect Action button to add a protect operation on the selected sector.

**NOTE** Press CTRL or SHIFT keys for selecting multiple sectors from the Sectors table.

3. Click the Add Unprotect Action button to add an unprotect action on the selected sector.
4. Check the All Device check box to add action on full device.
5. Click Done.

The Add Protect/Unprotect Action dialog box closes and the added protect or unprotect actions appear in the Flash Programmer Actions table in the Flash Programmer Task editor window.

### 3.11.2.3.7 Secure/Unsecure Actions

The secure/unsecure actions enable you to change the security of a flash device.

**NOTE** The Secure/Unsecure flash actions are not supported for StarCore devices.

To add a secure/unsecure action:

Figure 3.34 Add Secure/UnSecure Action Dialog Box

2. Select a device from the Flash Devices table.
3. Click the Add Secure Action button to add Secure action on the selected flash device.
   a. Type a password in the Password text box.
   b. Select the password format from the Format drop-down list box.
4. Click the Add Unsecure Action button to add an unprotect action on the selected sector.
5. Click Done.
   The Add Secure/UnSecure Action dialog box closes and the added secure or unsecure action appears in the Flash Programmer Actions table in the Flash Programmer Task editor window.

3.11.2.3.8 Duplicate an Action

To duplicate a flash programmer action from the Flash Programmer Actions table:
1. Select the action in the Flash Programmer Actions table.
2. Click the Duplicate Action button.
   The selected action is copied in the Flash Programmer Action table.

3.11.2.3.9 Remove an Action

To remove a flash programmer action from the Flash Programmer Actions table:
1. Select the action in the Flash Programmer Actions table.
2. Click the Remove Action button.
   The selected action is removed from the Flash Programmer Action table.

### 3.11.3 Run Flash Programmer Target Task

To execute the configured flash programmer target task, select a target task and click the Execute button in the Target Tasks view toolbar. Alternatively, right-click on a target task and select Execute from the context menu.

**Figure 3.35 Run Target Task**

![Target Task](image)

**NOTE** You can use predefined target tasks for supported boards. To load a predefined target task, right-click in the Target Tasks view and select Import Target Task from the context menu. To save your custom tasks, right-click in the Target Tasks view and then select Export Target Task from the context menu.

You can check the results of flash batch actions in the Console view. The green color indicates the success and the red color indicates the failure of the task.
3.12 Flash File to Target

This CodeWarrior feature enables you to perform flash operation. You don’t need any project for using Flash File to Target feature, only a valid Remote System is required. This feature enables you to perform these basic flash operations:

- **3.12.1 Erasing Flash Device**
- **3.12.2 Programming a File**

To open the Flash File to Target dialog box, click the Flash Programmer menu button on the IDE toolbar.

The following figure shows the Flash File to Target dialog box.
Figure 3.37 Flash File to Target Dialog Box

- **Remote system Connection** drop-down list — Lists all run configurations defined in Eclipse. If a connection to the target has already been made the control becomes inactive and contains the text Active Debug Configuration.

- **Flash Configuration File** drop-down list — Lists predefined target tasks for the processor selected in the Launch Configuration and tasks added by user with the **Browse** button. The values in this drop-down list are updated based on the processor selected in the launch configuration. For more information on launch configurations, see <product> targeting manual.
  - **Unprotect flash memory before erase** checkbox - Check to unprotect flash memory before erasing the flash device. This feature allows you to unprotect the flash memory from Flash File To Target dialog box.

- **File to Flash** group — Allows selecting the file to be programmed on the flash device and the location.
  - **File** text box — Enables you to specify the filename. You can use the **Workspace**, **File System**, or **Variables** buttons to select the desired file.
  - **Offset:0x** text box — Enables you to specify offset location for a file. If no offset is specified the default value of zero is used. The offset is always added to the start address of the file. If the file doesn't contain address information then zero is considered as start address.

- **Save as Target Task** — Check to enable **Task Name** text box.
  - **Task Name** text box — Enables you to save the specified settings as a Flash target task. Use the text box to specify the name of the target task.
• **Erase Whole Device** button — Erases the flash device. In case you have multiple flash blocks on the device, all blocks are erased. If you want to selectively erase or program blocks, use the 3.11 Flash Programmer feature.

• **Erase and Program** button — Erases the sectors that are occupied with data and then programs the file. If the flash device can not be accessed at sector level then the flash device is completely erased.

### 3.12.1 Erasing Flash Device

To erase a flash device, follow these steps:

1. Click the **Flash Programmer** menu button from the IDE toolbar
   
   The **Flash File to Target** dialog box appears.

2. Select a connection from the **Remote system Connection** drop-down list.

   **NOTE**  
   If a connection is already established with the target, this control is disabled.

   The **Flash Configuration** drop-down list is updated with the supported configurations for the processor from the launch configuration.

3. Select a flash configuration from the **Flash Configuration** drop-down list.

4. Check the **Unprotect flash memory before erase** checkbox to unprotect flash memory before erasing the flash device.

5. Click the **Erase Whole Device** button.

### 3.12.2 Programming a File

1. Click the **Flash Programmer** menu button from the IDE toolbar
   
   The **Flash File to Target** dialog box appears.

2. Select a connection from the **Remote system Connection** drop-down list.

   **NOTE**  
   If a connection is already established with the target, this control is disabled.

   The **Flash Configuration** drop-down list is updated with the supported configurations for the processor from the launch configuration.

3. Select a flash configuration from the **Flash Configuration** drop-down list.

4. Check the **Unprotect flash memory before erase** checkbox to unprotect flash memory before erasing the flash device.

5. Type the file name in the **File** text box. You can use the **Workspace, File System, or Variables** buttons to select the desired file.
6. Type the offset location in the Offset text box. Click the Erase and Program button.

### 3.13 Hardware Diagnostics

The **Hardware Diagnostics** utility lets you run a series of diagnostic tests that determine if the basic hardware is functional. These tests include:

- **Memory read/write** — This test only makes a read or write access to the memory in order to read or write a byte, word (2 bytes) and long word (4 bytes) to or from the memory. For this task, the user needs to set the options in the **Memory Access** group.

- **Scope loop** — This test makes read and write accesses to memory in a loop at the target address. The time between accesses is given by the loop speed settings. The loop can only be stopped by the user, which cancels the test. For this type of test, the user needs to set the memory access settings and the loop speed.

- **Memory tests** — This test requires the user to set the access size and target address from the access settings group and the settings present in the **Memory Tests** group.

The Hardware Diagnostics has following topics:

- 3.13.1 Creating Hardware Diagnostics Task
- 3.13.2 Working with Hardware Diagnostic Action Editor
- 3.13.3 Memory Test Use Cases

### 3.13.1 Creating Hardware Diagnostics Task

To create a task for Hardware Diagnostics:

1. Select **Window > Show View > Other** from the IDE menu bar.
   
   The **Show View** dialog box appears.

2. Expand the **Debug** group and select **Target Tasks**.

3. Click **OK**.

4. Click the Create a new Target Task button on the **Target Tasks** view toolbar. Alternatively, right-click on the **Target Tasks** view select New Task from the context menu.

   The Create a New Target Task wizard appears.
5. Type name for the new task in the Task Name text box.
6. Select a launch configuration from the Run Configuration drop-down list.

**NOTE** If the task does not successfully launch the configuration that you specify, the Execute button on the Target Tasks view toolbar stays disabled.

7. Select Hardware Diagnostic from the Task Type drop-down list.
8. Click Finish.

A new hardware diagnostic task is created in the Target Tasks view.

**NOTE** You can perform various actions on a hardware diagnostic task, such as renaming, deleting, or executing the task, using the context menu that appears on right-clicking the task in the Target tasks view.

### 3.13.2 Working with Hardware Diagnostic Action Editor

The Hardware Diagnostic Action editor is used to configure a hardware diagnostic task. To open the Hardware Diagnostic Action editor for a particular task, double-click the task in the Target Tasks view.

The Hardware Diagnostics Action editor window includes the following groups:

- [3.13.2.1 Action Type](#)
Debugger

Hardware Diagnostics

- 3.13.2.2 Memory Access
- 3.13.2.3 Loop Speed
- 3.13.2.4 Memory Tests

The following figure shows the Hardware Diagnostics Action editor window.

Figure 3.39 Hardware Diagnostics Action Editor Window

3.13.2.1 Action Type

The Action Type group in the Hardware Diagnostics Action editor window is used for selecting the action type. You can select any one of the following actions:

- Memory read/write — Enables the options in the Memory Access group.
- Scope loop — Enables the options in the Memory Access and the Loop Speed groups.
- Memory test — Enables the access size and target address from the access settings group and the settings present in the Memory Tests group.
3.13.2.2 Memory Access

The Memory Access pane (Figure 3.39) configures diagnostic tests for performing memory reads and writes over the remote connection interface. The table below lists and describes the items in the pane.

Table 3.6 Memory Access Pane Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</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>1 Byte</td>
<td>Select to have the hardware diagnostic tools perform byte-size operations.</td>
</tr>
<tr>
<td>2 Bytes</td>
<td>Select to have the hardware diagnostic tools perform word-size operations.</td>
</tr>
<tr>
<td>4 Bytes</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>
<tr>
<td>Value</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>Verify Memory Writes</td>
<td>Check the option to verify success of each data write to the memory.</td>
</tr>
</tbody>
</table>

3.13.2.3 Loop Speed

The Loop Speed pane (Figure 3.39) 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. After the first 1000 operations, the Status shows the estimated time between operations.

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. Following table lists and describes the items in Loop Speed pane.

### Table 3.7  Loop Speed Pane Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Loop Speed</td>
<td>Enter a numeric value between 0 to 1000 in the text box to adjust the speed. You can also move the slider to adjust the speed at which the hardware diagnostic tools repeat successive read and write operations. Lower speeds increase the delay between successive operations. Higher speeds decrease the delay between successive operations.</td>
</tr>
</tbody>
</table>

### 3.13.2.4 Memory Tests

The **Memory Tests** pane ([Figure 3.39](#)) lets you perform three hardware tests:

- 3.13.2.4.1 Walking Ones
- 3.13.2.4.2 Address
- 3.13.2.4.3 Bus Noise

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 compare 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.

The following table explains the items in the **Memory Tests** pane.
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 sub-tests:
- 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 are 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.

### 3.13.2.4.2 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.

### 3.13.2.4.3 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.

#### 3.13.2.4.3.1 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 + number and - number refer to the next item location (the specific increment or decrement depends on byte, word, or long word address mode):
  - the lowest address
  - the highest address
  - (the lowest address) + 1
  - (the highest address) - 1
  - (the lowest address) + 2
  - (the highest address) - 2

- **Maximum Invert Convergence**—This approach uses calculated end point addresses to maximize the number of bits flipping from one access to the next. This approach involves identifying address end points such that the values have the maximum inverted bits relative to one another. Specifically, the test identifies the lowest address with all 0x5 values in the least significant nibbles and the highest address with all 0xA values in the least significant nibbles. After the test identifies these end points, memory access alternates between low address and high address, working towards the center of the memory under test. Accessing memory in this manner, the test achieves the maximum number of bits flips from one access to the next.

#### 3.13.2.4.3.2 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 sub-tests 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 sub-tests:

- 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

3.13.3 Memory Test Use Cases

The memory read/write and scope loop tests are host based tests. The host machine issues read and write action to the memory through the connection protocol. For example CCS.

Memory tests are the complex tests that can be run in two modes: Host based and Target based depending upon the selection made for Use Target CPU check box.

- Checked: Target Based
- Unchecked: Host Based

The Host Based tests are slower than the Target Based tests.

3.13.3.1 Use Case 1: Run Host based Scope Loop on the Target

You need to perform the following action to run the host based scope loop on the target:

1. Select Scope loop in the Action Type.
2. Set Memory Access settings from the Memory Access section.
3. Set the speed used for the scope loop diagnostic from the Loop Speed Section.
4. Save the settings.
5. Press **Execute** to execute the action.

### 3.13.3.2 Use Case 2: Run Target based Memory Tests on the Target

You need to perform the following action to run the target based memory test on the target:

1. Select **Memory Test** in the **Action Type**.
2. Specify **Target Address** and **Access Size** settings from the **Memory Access** section.
3. Specify the following settings for **Memory Tests** section:
   - **Test Area Size**: The tested memory region is computed from **Target Address** until **Target Address + Test Area Size**.
   - **Tests to Run**: Select tests to run on the target.
   - **Number of passes**: Specify number of times a test will be executed.
   - **Use Target CPU**: set the Address to which the test driver (algorithm) is to be downloaded.

4. Save the settings.
5. Press **Execute** to execute the action.

### 3.14 Import/Export/Fill Memory

The **Import/Export/Fill Memory** utility lets you export memory contents to a file and import data from a file into memory. The utility also supports filling memory with a user provided data pattern.

#### 3.14.1 Creating a Task for Import/Export/Fill Memory

Use the **Import/Export/Fill Memory** utility to perform various tasks on memory. The utility can be accessed from the **Target Tasks** view.

To open the **Target Tasks** view:

1. Select **Window > Show View > Other** from the IDE menu bar.
   - The **Show View** dialog box appears.
2. Expand the **Debug** group.
3. Select **Target Tasks**.
4. Click **Ok**.

The first time it opens, the **Target Tasks** view contains no tasks. You must create a task in order to run the **Import/Export/Fill Memory** utility.

To create a task:
Debugger
Import/Export/Fill Memory

1. Click the Create a new Target Task toolbar button of the Target Tasks view. Alternatively, right-click the left-hand list of tasks and select New Task from the context menu that appears.

   The Create a New Target Task page appears.

   **Figure 3.40  Create New target Task Window**

2. In the Task Name text box, enter a name for the new task.

3. Use the Run Configuration drop-down list to specify the configuration that the task launches and uses to connect to the target.

   **NOTE** If the task does not successfully launch the configuration that you specify, the Execute button of the Target Tasks view toolbar stays disabled.

4. Use the Task Type drop-down list to specify Import/Export/Fill Memory.

5. Click Finish.

   The Import/Export/Fill Memory target task is created and it appears in the Import/Export/Fill Memory Action editor window appears.
3.14.2 Importing Data from a File into Memory

Select the Import memory option from the Import/Export/Fill Memory Action editor window to import the encoded data from a user specified file, decode it, and copy it into a user specified memory range.
The following table explains the import memory options.

**Table 3.9 Import Data from a File into Memory Window items**

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory space and address</td>
<td>Enter the literal address and memory space on which the data transfer is performed. The Literal address field allows only decimal and hexadecimal values.</td>
</tr>
<tr>
<td>Expression</td>
<td>Enter the memory address or expression at which the data transfer starts.</td>
</tr>
<tr>
<td>Access Size</td>
<td>Denotes the number of addressable units of memory that the debugger accesses in transferring one data element. The default values shown are 1, 2, and 4 units. When target information is available, this list shall be filtered to display the access sizes that are supported by the target.</td>
</tr>
</tbody>
</table>
3.14.3 Exporting Memory Contents to a File

Select the Export memory option from the Import/Export/Fill Memory Action editor window to read data from a user specified memory range, encode it in a user specified format, and store this encoded data in a user specified output file.

Table 3.9 Import Data from a File into Memory Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Type</td>
<td>Defines the format in which the imported data is encoded. By default, the following file types are supported:</td>
</tr>
<tr>
<td></td>
<td>• Signed decimal Text</td>
</tr>
<tr>
<td></td>
<td>• Unsigned decimal Text</td>
</tr>
<tr>
<td></td>
<td>• Motorola S-Record format</td>
</tr>
<tr>
<td></td>
<td>• Hex Text</td>
</tr>
<tr>
<td></td>
<td>• Annotated Hex Text</td>
</tr>
<tr>
<td></td>
<td>• Raw Binary</td>
</tr>
<tr>
<td>Select file</td>
<td>Enter the path to the file that contains the data to be imported. Click the Workspace button to select a file from the current project workspace.</td>
</tr>
<tr>
<td></td>
<td>Click the System button to select a file from the file system the standard File Open dialog box. Click the Variables button to select a build variable.</td>
</tr>
<tr>
<td>Number of Elements</td>
<td>Enter the total number of elements to be transferred.</td>
</tr>
<tr>
<td>Verify Memory Writes</td>
<td>Check the option to verify success of each data write to the memory.</td>
</tr>
</tbody>
</table>
The following table explains the export memory options.

### Table 3.10 Export Data from Memory into a File Window Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory space and address</td>
<td>Enter the literal address and memory space on which the data transfer is performed. The Literal address field allows only decimal and hexadecimal values.</td>
</tr>
<tr>
<td>Expression</td>
<td>Enter the memory address or expression at which the data transfer starts.</td>
</tr>
<tr>
<td>Access Size</td>
<td>Denotes the number of addressable units of memory that the debugger accesses in transferring one data element. The default values shown are 1, 2, and 4 units. When target information is available, this list shall be filtered to display the access sizes that are supported by the target.</td>
</tr>
</tbody>
</table>
3.14.4 Fill Memory with a Data Pattern

Select the Fill memory option from the Import/Export/Fill Memory Action editor window to fill a user specified memory range with a user specified data pattern.
The following table explains the fill memory options.

Table 3.11 Fill Memory With a Data Pattern Window items

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| Memory space and address | Enter the literal address and memory space on which the fill operation is performed.  
The Literal address field allows only decimal and hexadecimal values. |
| Expression            | Enter the memory address or expression at which the fill operation starts.                                                                     |
| Access Size           | Denotes the number of addressable units of memory that the debugger accesses in modifying one data element.  
The default values shown are 1, 2, and 4 units. When target information is available, this list shall be filtered to display the access sizes that are supported by the target. |
| Fill Pattern          | Denotes the sequence of bytes, ordered from low to high memory mirrored in the target.  
The field accept only hexadecimal values. If the width of the pattern exceeds the access size, an error message. |
3.15 Launch Group

A launch group is a launch configuration that contains other launch configurations. You can add any number of existing launch configurations to the launch group and order them. In addition, you can attach an action to each launch configuration.

You can also specify the mode in which the launch configuration should be launched. For example, run mode or debug mode.

3.15.1 Creating a Launch Group

To create a launch group:

1. Select Run > Debug Configurations.
   The Debug Configurations dialog box appears.
2. Select Launch Group from the left panel.
3. Click the New launch configuration button.
   A new launch configuration of launch group type is created and shown on the left panel of the Debug Configurations dialog box.

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Elements</td>
<td>Enter the total number of elements to be modified.</td>
</tr>
<tr>
<td>Verify Memory Writes</td>
<td>Check the option to verify success of each data write to the memory.</td>
</tr>
</tbody>
</table>
Figure 3.45  Launch Group Configuration Panel Controls

Table 3.12  Launch Group Configuration Panel Controls

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Specify a name for the launch group</td>
</tr>
<tr>
<td>Up button</td>
<td>Click to move up the selected launch configuration</td>
</tr>
<tr>
<td>Down button</td>
<td>Click to move down the selected launch configuration</td>
</tr>
<tr>
<td>Edit button</td>
<td>Click to edit the selected entry in the launch group</td>
</tr>
<tr>
<td>Add button</td>
<td>Click to add a launch configuration to the launch group</td>
</tr>
<tr>
<td>Remove button</td>
<td>Click to remove a launch configuration from the launch group</td>
</tr>
</tbody>
</table>

4. Specify a name for the launch group configuration in the Name text box.
5. Click **Add**.

The **Add Launch Configuration** dialog box appears.

**Figure 3.46 Add Launch Configuration Dialog Box**
6. To add a launch configuration to the launch group:
   a. Select one or more launch configurations from the tree control.
   b. Select an action from the Post launch action list.
   c. Click OK.
      The launch configuration is added to the launch group and the Add Launch Configuration dialog box closes.

7. Click Apply.
   The launch configurations are added to the launch group.

Table 3.13 Add Launch Configuration Dialog Box Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch Mode</td>
<td>Enables you to specify launch mode for the selected launch configuration. This can also be used to filter launch configurations. debug — specifies that the launch configuration will be launched in debug mode. run — specifies that the launch configuration will be launched in run mode. profile — specifies that the launch configuration will be launched in profile mode.</td>
</tr>
<tr>
<td>Use default mode when launching</td>
<td>Checking this option indicates that the child launch configuration should be launched in the mode used to initiate the launch group launch.</td>
</tr>
<tr>
<td>Post launch action</td>
<td>Enables you to specify a post launch action for the selected launch configuration. None — the debugger immediately moves on to launch the next launch configuration. Wait until terminated — the debugger waits indefinitely until the debug session spawned by the last launch terminates and then it moves on to the next launch configuration. Delay — the debugger waits for specified number of seconds before moving on to the next launch configuration.</td>
</tr>
</tbody>
</table>
3.15.2 Launching the Launch Group

When launched, the debugger iterates through the launch configurations contained in the launch group and launches each enabled configuration sequentially, in the same order as they are configured in the launch group.

The following figure shows the result of a launch group launch in the Debug view.

Figure 3.47 Launch Group in Debug View

3.16 Load Multiple Binaries

The CodeWarrior debugger supports loading multiple binaries (.elf), to enable the availability of symbols and source code of other executable, within a debugging session.

To load multiple binary files within a debugging session:

1. Click Run > Open Debug Dialog.

   The Debug Configurations dialog box appears. The left side of this window has a list of debug configurations that apply to the current application.

2. Expand the CodeWarrior configuration.
3. From the expanded list, select the debug configuration that you want to modify.

The following figure shows the **Debug Configurations** dialog box with the settings for the debug configuration you selected.

**Figure 3.48 Debug Configurations Dialog Box**

4. Click the **Debugger** tab to view the corresponding debugger settings page.

5. Click the **Other Executables** tab under the **Debugger Options** panel on the page.

6. Click **Add** to open the **Debug Other Executable** dialog box.

The **Debug Other Executables** dialog box enables you to specify additional ELF files to download or debug in addition to the main executable file associated with the launch configuration.

**Figure 3.49 Debug Other Executable**
7. Enter the path to the additional executable file that the debugger controls in addition to the current project’s executable file. Alternatively, click the Browse button to specify the file path.

8. Check the Load Symbols option to have the debugger load symbols for the specified file. Clear to prevent the debugger from loading the symbols. The Debug column of the File list corresponds to this setting.

9. Check the Download to Device option to have the debugger download the specified file to the target device. Clear this option to prevent the debugger from downloading the file to the device. The Download column of the File list corresponds to this setting.

10. Click OK to add the additional executable to the Other Executables file list.

11. Click Debug to launch a debug session with multiple binaries.

Multiple binary files within a debugging session are now available.

3.16.1 Viewing Binaries

The Modules view shows the application executable and all shared libraries loaded by the application during a debug session. In addition to the current project’s executable file, the Modules view shows the other executables listed in the Other Executables panel (refer to 3.16 Load Multiple Binaries).

To view the executables loaded during a debug session:

1. Select Window > Show View > Modules from the IDE menu bar.

   The Modules view appears.

   **Figure 3.50 Modules View**

2. Click on the application executable to view its details.

   An executable can also be expanded in the modules view (to shows its symbols) regardless of whether the executable has been targeted or not in the Debug Other Executables panel.
3. An executable that is not marked to be targeted at launch time can be forced to be targeted at any time during the debug session by selecting Load Symbols from the context menu that appears. The menu item will be disabled if the executable is already targeted.

NOTE All executables listed in the Other Executables pane are added to the Modules view whether or not they are marked to be targeted or downloaded.

3.17 Memory View

The Memory view lets you monitor and modify your process memory. The process memory is presented as a list called memory monitors. Each monitor represents a section of memory specified by its location called base address. Each memory monitor can be displayed in different predefined data formats known as memory renderings.

The debugger supports the following rendering types:

- Disassembly
- Hexadecimal (default)
- ASCII
- Signed integer
- Unsigned integer
- Mixed Source
- Traditional

The default rendering is displayed automatically when a monitor is created.

The Memory view contains these two panes:

- Monitors panel — Displays the list of memory monitors added to the debug session currently selected in the Debug view
- Renderings panel — Display memory renderings.

The content of the Renderings panel is controlled by the selection in the Monitors panel. The Renderings panel can be configured to display two renderings simultaneously.

3.17.1 Opening Memory View

To open the Memory view, click the Memory tab of the Debug perspective. Alternatively, from the IDE menu bar, select Window > Show View > Memory.

Figure 3.51 Memory View
3.17.2 Adding Memory Monitor

To add a memory monitor to Memory view:

1. Start a debugging session.
2. Select the Memory tab.
   The Memory view comes forward.
3. In the Monitors pane toolbar, click the plus-sign (+) icon. Alternatively, right-click a blank area in the Monitors pane and select Add Memory Monitor.
   The Monitor Memory dialog box appears.

4. Specify information about the memory monitor:
   • To enter a memory space and literal address, simply enter an address.
   • To enter an expression, type in the expression. If you enter a literal address as the expression, use the prefix 0x to indicate hexadecimal notation, or use no prefix to indicate decimal notation. You can use the drop-down list to select a previously specified expression.

   **NOTE** If you do not select a memory space and the expression does not contain a memory space then the memory space is set to default data memory space that is specific for each architecture.
5. If you want to translate the memory address or the expression to another memory space, check the Memory space check box. The Memory space drop-down list is enabled.

6. Select one of the following values from the Memory space drop-down list.
   - Physical — Indicates that the specified address or expression refers to physical memory space.
   - Data — Indicates that the specified address or expression refers to data memory space.
   - Program — Indicates that the specified address or expression refers to program memory space.

7. Click OK.

The memory monitor is added to the Monitors panel and the default rendering is displayed in the Renderings panel.

Figure 3.53  Added Memory Monitor

3.17.3 Adding Memory Renderings

When you add a variable, expression, or register to the Memory view, you can do so multiple times, each time adding a new (or the same) rendering. Alternatively, once you have added a memory monitor and rendering, you can go to the Renderings pane and click Add Rendering(s). This will prompt you with a dialog to select the rendering that you want to add to the view. In this dialog, you can select more than one rendering by using the keyboard Shift or Ctrl keys - doing this will cause a rendering to be opened for each rendering format that is selected. When you add multiple memory renderings, they are separated by tabs.

For more details on opening the Memory view and adding memory monitors, refer 3.17.2 Adding Memory Monitor.

You can also split the Memory Renderings pane by selecting Toggle Split Pane. When the Renderings pane is split, you can view two renderings side-by-side.

When you have multiple memory renderings for a memory monitor, you can set the renderings to be linked with one another. To do this, select the Link Memory Rendering Panes. When renderings are
linked, they are synchronized with each other (for example, if you change the column size in one rendering, the column size in the other rendering will also change - or if you scroll or move the cursor in one rendering, the other rendering will scroll or follow the same cursor movement). Linking memory renderings only applies to the current Memory view. If you have multiple Memory views open, they do not link to each other.

To remove a rendering, select it in the Renderings pane and click Remove Rendering. When you remove all memory renderings for a monitored expression, variable, or register, the Renderings pane will be populated with the memory rendering selection list. From this list, you can select the data format that you want to use for the memory rendering and then click Add Rendering(s).

Figure 3.54 Added Rendering

3.17.4 Mixed Source Rendering

The mixed source rendering enables you to view memory with instructions in C correspondence or mixed modes.

To add mixed source rendering in the Memory view:
1. Open the Memory view, see 3.17.1 Opening Memory View.
2. Add a memory monitor, see 3.17.2 Adding Memory Monitor
3. Click the New Renderings tab in the Renderings panel.
   The New Renderings tab displays the different rendering types that can be added in the Renderings panel.
4. Select Mixed Source from the Select rendering(s) to create list.
5. Click the Add Rendering(s) button.
   The mixed source rendering is added to the Renderings panel in the Memory view.
3.17.5 Setting Memory Access Size

To set memory access size:

1. Open the Memory view, see 3.17.1 Opening Memory View.
2. Right-click on a Memory Rendering.
   The context menu appears.
3. Select Format from the context menu.
   The Format dialog box appears.
4. Select a row size from the **Row Size** drop-down list to change the number of rows displayed in the **Renderings** panel of the **Memory** view.

5. Select a column size from the **Column Size** drop-down list to change the number of columns.

**NOTE** The default value for the **Column size** depends on the architecture being debugged. For example, for 32 bit architectures the default value for **Column size** is 4 and for 8 bit architectures the default value is 1. To save the newly selected values as default values, click the **Save as Defaults** button.

6. Click **OK**.

Now, the memory access size is set.

### 3.17.6 Exporting Memory

To export memory data:

1. Open the **Memory** view, see 3.17.1 Opening Memory View.
2. Click the **Export** button in the **Memory** view toolbar.

   The **Export Memory** dialog box appears.
Figure 3.57 Export Memory Dialog Box

- **Format** drop-down list — Enables you to select the format in which the memory data is exported.
  - SRecord — Exports memory data in Motorola S-record format.
  - Plain Text — Exports memory data in ASCII format.
  - RAW Binary — Exports memory data in binary format.
- **Start address** text box — Enables you to specify the start address of memory range to be exported.
- **End address** text box — Enables you to specify the end address of the memory range to be exported.
- **Length** text box — Displays the length of the memory range.
- **File name** text box — Enables you to specify the file name to save the exported memory. Click the **Browse** button to select a file on your system.

3. Select memory format from the **Format** drop-down list.
4. Specify the start address of the memory range to be exported in the **Start address** text box.
5. Specify the end address of the memory range to be exported in the **End address** text box.
6. Type a file name in the **File name** text box. Click **Browse** to select a file on your system.
7. Click **OK**.

Memory data is now exported.

### 3.17.7 Importing Memory

To import memory data:

1. Open the **Memory** view, see 3.17.1 Opening Memory View.
2. Click the **Import** button in the **Memory** view toolbar.

The **Import Memory** dialog box appears.
3. Select memory format from the **Format** drop-down list.
4. Select **Restore to address specified in the file** to restore the memory to location specified in the file.
5. Select **Restore to this address** option to store the memory data at the specified memory location. Type the memory location in the adjacent text box.
6. Type a file name in the **File name** text box. Click **Browse** to select a file from your file system.
7. Check the **Scroll to restore address** check box to scroll to restore point in memory view after the export operation is completed.
8. Click **OK**.

The memory data is now imported.

### 3.17.8 Setting Watchpoint in Memory View

To set a watchpoint using the **Memory** view:
### 3.17.9 Clearing Watchpoints from the Memory View

To clear a watchpoint from the Memory view:

1. Select the watchpoint expression in the Breakpoint view.
2. Click the Remove Selected Breakpoints button.

To clear all watchpoints from the Memory view:

1. Open the Breakpoint view.
2. Choose Run > Remove all Breakpoints or click the Remove All Breakpoints button in the Breakpoints view.

**NOTE**  All watchpoints clear automatically when the target program terminates or the debugger terminates the program. Your watchpoints are reset the next time the program runs.

The watchpoint is cleared from memory view.

### 3.18 Memory Browser View

The Memory Browser view lets you monitor your process memory. This view also enables you to browse through the memory rendering. You can also specify a memory space along the address to browse for.

To open the Memory Browser view, click the Memory Browser tab of the Debug perspective. Alternatively, from the IDE menu bar, select Window > Show View > Memory Browser.
To browse to a desired memory location, type the memory address in the Memory Address text box and click the Go button. The memory location is highlighted in the Memory Browser view.

NOTE If you do not select a memory space and the expression does not contain a memory space then the memory space is set to default data memory space that is specific for each architecture.

If you want to translate the memory address or the expression to another memory space, select one of the following values from the Memory space drop-down list.

- Physical - Indicates that the specified address or expression refers to physical memory space.
- Data - Indicates that the specified address or expression refers to data memory space.
- Program - Indicates that the specified address or expression refers to program memory space.

### 3.19 Memory Management Unit Configurator

The CodeWarrior Memory Management Unit (MMU) Configurator allows different user tasks or programs (usually in the context of an RTOS) to use the same areas of memory. To use the MMU configurator, you set up a mapping for data and instruction addresses, then enable address translation. The mapping links virtual addresses to physical addresses. Translation occurs before software acts on the addresses.

The MMU configurator simplifies peripheral-register initialization of the MMU registers. You can use the tool to generate code that you can insert into a program. The inserted code initializes an MMU configuration or writes to the registers on-the-fly. Also, you can use the MMU configurator to examine the status of the current MMU configuration.

Use the MMU configurator to:

- configure MMU general control registers
3.19.1 Creating an MMU Configuration

In order to use the MMU configurator, you must create an MMU configuration. To create the configuration:

1. From the main menu bar, select File > New > Other.

   The New window appears.
2. Expand the **Peripheral Configurators** group.
3. From the expanded group, select **MMU Configuration File**.
4. Click **Next**.

   The **MMU Configurator File** page appears.
5. Enter in the **Container** text box the path to the directory in which you want to store the MMU configuration. Alternatively, click **Browse**, then use the resulting dialog box to specify the directory.

6. Enter in the **File name** text box a name for the MMU configuration. Alternatively, leave the default name intact.

**NOTE** If you enter a new name, make sure to preserve the `.mmu` filename extension.

7. Expand the **Device Number** list.

8. From the expanded list, select the target hardware for which you are creating the MMU configuration. *(SC3x50)*

9. Click **Finish**.

The New window closes. The IDE generates the MMU configuration file in the specified container directory, then opens the **MMU Configuration File Editor** view.
3.19.2 Saving MMU Configurator Settings

Each time you change a setting on a page of the MMU configurator, you create a pending (unsaved) change. In order to commit those pending changes, you must save the MMU configurator settings to a file. An asterisk (*) appears to the left of the MMU Configuration File Editor tab text to indicate that the MMU configurator still has pending changes among its pages.

To save to a file the current settings on each page of the MMU Configuration File Editor view:

1. Click the MMU Configuration File Editor tab.
   The corresponding view becomes active.
2. From the main menu bar, select File > Save.
3. The IDE saves to a file the pending changes to each page of the MMU configurator.

3.19.3 MMU Configurator Toolbar

The MMU configurator has an associated toolbar. Depending on how you open the MMU configurator, this toolbar appears either in the main IDE toolbar, or in the MMU configurator view toolbar. The following table explains each toolbar button.
3.19.4 MMU Configurator Pages

This section explains each MMU configurator page. You use these pages to configure MMU mapping and translation properties. The MMU configurator's tabbed interface displays pages for configuration options and pages for generated code.

3.19.4.1 General Page

Use the **General** page to configure the overall MMU properties.

---

**Table 3.14 MMU Configurator Toolbar Buttons**

<table>
<thead>
<tr>
<th>Name</th>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save C Source</td>
<td><img src="image" alt="C file icon" /></td>
<td>Save the generated C code to a new .c file.</td>
</tr>
<tr>
<td>Save ASM Source</td>
<td><img src="image" alt="ASM file icon" /></td>
<td>Save the generated Assembly code to a new .asm file.</td>
</tr>
<tr>
<td>Save TCL Source</td>
<td><img src="image" alt="TCL file icon" /></td>
<td>Save the generated TCL script to a new .tcl file.</td>
</tr>
<tr>
<td>Read Target Registers</td>
<td><img src="image" alt="Target registers icon" /></td>
<td>Updates the content of the MMU Configuration File Editor pages to reflect the current values of the target hardware registers.</td>
</tr>
<tr>
<td>Write Target Registers</td>
<td><img src="image" alt="Target registers icon" /></td>
<td>Writes the modified content of all the MMU Configuration File Editor pages. You must click this button, or use the corresponding toolbar menu command to write the MMU Configurator modifications to the target hardware registers.</td>
</tr>
</tbody>
</table>
The following table explains options on the **General** page.

**Table 3.15 General Page Settings**

<table>
<thead>
<tr>
<th>Page Item</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| Address Translation  | **Checked**—Enables address translation. For example, translation occurs from a virtual address to a physical address.  
**Cleared**—Disables address translation. For example, translation does not occur from a virtual address to a physical address.  
This option corresponds to the Address Translation Enable (ATE) bit of the MMU Control Register (M_CR). |
| Memory Protection    | **Checked**—Enables protection checking for all enabled segment descriptors. With this option checked, the system consumes more power.  
**Cleared**—Disables protection checking for all enabled segment descriptors. With this option cleared, the system consumes less power.  
This option corresponds to the Memory Protection Enable (MPE) bit of the MMU Control Register (M_CR). |
### 3.19.4.2 Program MATT Page

Use the Program MATT page (Figure 3.64) to define and display program memory-space mappings (virtual-to-physical address mappings) for the StarCore DSP. The MMU configurator generates the appropriate descriptors for the program memory-address translation table (MATT).

Each memory-space mapping has a corresponding entry in the list on the left-hand side of the Program MATT page. Each entry shows an abbreviated expression which summarizes the settings on the right-hand side of the page. A plus sign to the left of an entry indicates an enabled mapping, and a minus sign indicates a disabled mapping.

To change an entry, select it from the left-hand side of the page, then use the Address, Size, and Properties settings to specify options that the MMU configurator verifies as a group. Click the Change button to assign the specified options to the selected entry. To cancel your changes, select another entry from the left-hand side of the page, without clicking the Change button.
The following table explains each option on the Program MATT page.

### Table 3.16 Program MATT Page Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| Virtual | Enter the virtual base address of the program segment.  
This option corresponds to the Program Segment Virtual Base Address and Size (PSVBAS) bits of the Program Segment Descriptor Registers A (M_PSDAx) that configure the virtual base address. |
| Physical | Enter the most-significant part of the physical address to use for translation. This option corresponds to the Data Segment Physical Base Address (DSPBA) bits of the Data Segment Descriptor Registers B (M_DSDBx). |
| Size    | Specify the PMATT Units number in **Number** box.  
Select the PMATT Units type from the drop-down list: **B**, **KB**, **MB**, **GB** |
| Permissions | Specify whether to share the program segment.  
This option corresponds to the System/Shared Virtual Program Memory (SSVPM) bit of the Program Segment Descriptor Registers A (M_PSDAx). |
### Debugger

**Memory Management Unit Configurator**

#### Table 3.16 Program MATT Page Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| **Burst**       | Specify the number of transactions (beats) on the bus that the bus controller cannot interrupt. This burst size applies in the region to a cacheable segment.  
This option corresponds to the Program Burst Size (PBS) bits of the Program Segment Descriptor Registers B (M_PSDBx). |
| **L2 Cache Policy** | Determines the cache policy for the L2 cache for accesses from the core through L1 Instruction cache: **Cacheable**, **NonCacheable**, and **Reserved**.  
The drop-down list has two **Reserved** values. This is because the L2 Cache Policy Values is stored on 2 bits so they are 4 possible values (2 valid and 2 reserved). Every entry in the combo box corresponds to a combination of bits. |
| **Cacheable**   | **Checked**—Enables caching of the segment in instruction cache.  
**Cleared**—Disables caching of the segment in instruction cache.  
This option corresponds to the Instruction Cacheability (IC) bit of the Program Segment Descriptor Registers A (M_PSDAx). |
| **PAPS**        | **Checked**—The segment has supervisor-level fetch permission for program accesses. If you check the PAPU option as well, you disable program-protection checks for this segment.  
**Cleared**—The segment does not have supervisor-level fetch permission for program accesses.  
This option corresponds to the Program Access Permission in Supervisor Level (PAPS) bit of the Program Segment Descriptor Registers A (M_PSDAx). |
| **Entry Enabled** | **Checked**—The MMU enables this mapping entry.  
**Cleared**—The MMU disables this mapping entry. |
| **PAPU**        | **Checked**—The segment has user-level fetch permission for program accesses. If you check the PAPS option as well, you disable program-protection checks for this segment.  
**Cleared**—The segment does not have user-level fetch permission for program accesses.  
This option corresponds to the Program Access Permission in User Level (PAPU) bit of the Program Segment Descriptor Registers A (M_PSDAx). |
The **PMATT Table** page (Figure 3.65) shows an alternate, tabular rendering of the settings that you specify on the **Program MATT** page. Use this page to view the configuration of all Program MATT mappings. The MMU configurator uses the settings that you specify on the **Program MATT** page to generate the column headers of this page. The table data shows the validated records for each Program MATT entry. You can resize the table columns to hide columns or view the larger data fields. A plus sign (+) in a table cell represents a checked check box in the associated Program MATT configuration page.

**NOTE**  The PMATT Table page shows just a tabular summary of the settings that you specify on the Program MATT page. To make changes, use the Program MATT page.

### Table 3.16 Program MATT Page Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefetch Line</td>
<td><strong>Checked</strong>—Enables the fetch unit’s program-line pre-fetch to a segment cacheable in instruction cache.</td>
</tr>
<tr>
<td></td>
<td><strong>Cleared</strong>—Disables the fetch unit’s program-line prefetch to a segment cacheable in instruction cache.</td>
</tr>
<tr>
<td></td>
<td>This option corresponds to the Program Pre-fetch Line Enable (PPFE) bit of the Program Segment Descriptor Registers B (M_PSDBx).</td>
</tr>
<tr>
<td>Program Next Line Prefetch</td>
<td><strong>Checked</strong>—Enables the fetch unit’s program next line pre-fetch mechanism to an ICache cacheable segment.</td>
</tr>
<tr>
<td></td>
<td><strong>Cleared</strong>—Enables the fetch unit's program next line pre-fetch mechanism to an ICache cacheable segment.</td>
</tr>
</tbody>
</table>
Use the **Data MATT** page to define and display data memory-space mappings (virtual-to physical address mappings) for the **StarCore DSP**. The MMU configurator generates the appropriate descriptors for the data memory-address translation table (MATT).

Each memory-space mapping has a corresponding entry in the list on the left-hand side of the **Data MATT** page. Each entry shows an abbreviated expression which summarizes the settings on the right-hand side of the page. A plus sign to the left of an entry indicates an enabled mapping, and a minus sign indicates a disabled mapping.

To change an entry, select it from the left-hand side of the page, then use the Address, Size, and Properties settings to specify options that the MMU configurator verifies as a group. Click the Change button to assign the specified options to the selected entry. To cancel your changes, select another entry from the left-hand side of the page, without clicking the Change button.

The following figure shows the **Data MATT** page.
The following table explains each option on the Data MATT page

**Table 3.17  Data MATT Page Settings**

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual</td>
<td>Enter the virtual base address of the data segment. This option corresponds to the Data Segment Virtual Base Address and Size (DSVBAS) bits of the Data Segment Descriptor Registers A (M_DSDAx) that configure the virtual base address.</td>
</tr>
<tr>
<td>Physical</td>
<td>Enter the most-significant part of the physical address to use for translation. The value that you specify with the Range drop-down list determines the size of the most-significant part. This option corresponds to the Data Segment Physical Base Address (DSPBA) bits of the Data Segment Descriptor Registers B (M_DSDBx).</td>
</tr>
<tr>
<td>Size</td>
<td>Specify the PMATT Units number in <strong>Number</strong> box. Select the PMATT Units type from the drop-down list: <strong>B</strong>, <strong>KB</strong>, <strong>MB</strong>, <strong>GB</strong></td>
</tr>
</tbody>
</table>
Table 3.17 Data MATT Page Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissions</td>
<td>Specify whether to share the data segment: <strong>shared</strong> and <strong>non-shared</strong>&lt;br&gt;This option corresponds to the Supervisor/Shared Virtual Data Memory (SSVDM) bit of the Data Segment Descriptor Registers A (M_DSDAx).</td>
</tr>
<tr>
<td>Burst</td>
<td>Specify the number of transactions (beats) on the bus that the bus controller cannot interrupt. This burst size applies in the region to a cacheable segment.&lt;br&gt;This option corresponds to the Data Burst Size (DBS) bits of the Data Segment Descriptor Registers B (M_DSDBx).</td>
</tr>
<tr>
<td>L2 Cache Policy</td>
<td>Determines the cache policy for the L2 cache for accesses from the core through L1 Data Cache: <strong>Cacheable write through</strong>, <strong>Cacheable write-back</strong>, <strong>Non-cacheable</strong>, and <strong>Adaptive write</strong>.</td>
</tr>
<tr>
<td>DAPU</td>
<td>Specify whether to allow user-level read (r-), write (-w), both (rw), or neither (-- types of data access.&lt;br&gt;This option corresponds to the Data Access Permission in User Level (DAPU) bits of the Data Segment Descriptor Registers A (M_DSDAx).</td>
</tr>
<tr>
<td>DAPS</td>
<td>Specify whether to allow supervisor-level read (r-), write (-w), both (rw), or neither (-- types of data access.&lt;br&gt;This option corresponds to the Data Access Permission in Supervisor Level (DAPS) bits of the Data Segment Descriptor Registers A (M_DSDAx).</td>
</tr>
</tbody>
</table>
Write Policy

Specify the policy to use for data writes and cache:

- **Cacheable write through**—Writes are buffered in the write queue (WRQ) and go to both the cache and the higher-level memory. The write-through is a non-write allocate, and a cacheable write-through access is not updated in the cache unless there is a hit.

- **Cacheable write back**—Writes are buffered in the write queue (WRQ) and go through the DCache and the write back buffer (WBB). The information is written to the VBR in the cache only. The modified cache VBR is written to higher-level memory only when it is replaced. The resulting WBB is combined with a write-allocate write-miss policy in which the required VBR is loaded to cache when a write-miss occurs.

- **Non Cacheable write through**—Writes are buffered in the WRQ and go through the write through buffer (WTB) to the higher-level memory.

- **Non-cacheable write-through destructive area**—Writes are buffered in the WRQ and go through the write through buffer (WTB) to the higher-level memory. Speculative read accesses are blocked in the platform level and does not go to a higher level memory.

Prefetch Line

- **Checked**—Enables the fetch unit’s data-line prefetch to a segment cacheable in data cache.

- **Cleared**—Disables the fetch unit’s data-line prefetch to a segment cacheable in data cache.

This option corresponds to Data Pre-fetch Line Enable (DPFE) bit of the Data Segment Descriptor Registers B (M_DSDBx).

Entry Enabled

- **Checked**—The MMU enables this mapping entry.

- **Cleared**—The MMU disables this mapping entry.

### Table 3.17 Data MATT Page Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Policy</td>
<td>Specify the policy to use for data writes and cache:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Cacheable write through</strong>—Writes are buffered in the write queue (WRQ)</td>
</tr>
<tr>
<td></td>
<td>and go to both the cache and the higher-level memory. The write-through is</td>
</tr>
<tr>
<td></td>
<td>a non-write allocate, and a cacheable write-through access is not updated</td>
</tr>
<tr>
<td></td>
<td>in the cache unless there is a hit.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Cacheable write back</strong>—Writes are buffered in the write queue (WRQ)</td>
</tr>
<tr>
<td></td>
<td>and go through the DCache and the write back buffer (WBB). The information</td>
</tr>
<tr>
<td></td>
<td>is written to the VBR in the cache only. The modified cache VBR is written</td>
</tr>
<tr>
<td></td>
<td>to higher-level memory only when it is replaced. The resulting WBB is</td>
</tr>
<tr>
<td></td>
<td>combined with a write-allocate write-miss policy in which the required VBR</td>
</tr>
<tr>
<td></td>
<td>is loaded to cache when a write-miss occurs.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Non Cacheable write through</strong>—Writes are buffered in the WRQ and go</td>
</tr>
<tr>
<td></td>
<td>through the write through buffer (WTB) to the higher-level memory.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Non-cacheable write-through destructive area</strong>—Writes are buffered in</td>
</tr>
<tr>
<td></td>
<td>the WRQ and go through the write through buffer (WTB) to the higher-level</td>
</tr>
<tr>
<td></td>
<td>memory. Speculative read accesses are blocked in the platform level and</td>
</tr>
<tr>
<td></td>
<td>does not go to a higher level memory.</td>
</tr>
<tr>
<td>Prefetch Line</td>
<td><strong>Checked</strong>—Enables the fetch unit’s data-line prefetch to a segment</td>
</tr>
<tr>
<td></td>
<td>cacheable in data cache.</td>
</tr>
<tr>
<td></td>
<td><strong>Cleared</strong>—Disables the fetch unit’s data-line prefetch to a segment</td>
</tr>
<tr>
<td></td>
<td>cacheable in data cache.</td>
</tr>
<tr>
<td></td>
<td>This option corresponds to Data Pre-fetch Line Enable (DPFE) bit of the</td>
</tr>
<tr>
<td></td>
<td>Data Segment Descriptor Registers B (M_DSDBx).</td>
</tr>
<tr>
<td>Entry Enabled</td>
<td><strong>Checked</strong>—The MMU enables this mapping entry.</td>
</tr>
<tr>
<td></td>
<td><strong>Cleared</strong>—The MMU disables this mapping entry.</td>
</tr>
</tbody>
</table>
The DMATT Table page (Figure 3.67) shows an alternate, tabular rendering of the settings that you specify on the Data MATT page. Use this page to view the configuration of all Data MATT mappings. The MMU configurator uses the settings that you specify on the Data MATT page to generate the column headers of this page. The table data shows the validated records for each Data MATT entry. You can resize the table columns to hide columns or view the larger data fields. A plus sign (+) in a table cell represents a checked check box in the associated Data MATT configuration page.

**NOTE** The DMATT Table page shows the summary of the settings that you specify on the Data MATT page in the tabular format. To change these settings, use the Data MATT page.

### 3.19.4.4 Saving MMU Configurator Generated Code

In this section:

- 3.19.4.4.1 Saving Generated C Code
- 3.19.4.4.2 Saving Generated Assembly Code
- 3.19.4.4.3 Saving Generated TCL Script

#### 3.19.4.4.1 Saving Generated C Code

The generated C code is unique for different targets. Follow these steps to save the C code generated by the MMU Configuration File Editor:
1. From the CodeWarrior IDE menu bar, select **MMU Editor** > **Save C** to save the generated C code. Alternatively, click the corresponding toolbar buttons in the MMU Configurator toolbar. A standard Save dialog box appears.

2. Specify the filename in the **File name** text box and click **Save** to save the generated code as a new file.

**NOTE**  The MMU Configuration File Editor regenerates the C code when you change settings in the configuration pages or when you click Change on the **Program MATT** or **Data MATT** pages.

### 3.19.4.4.2 Saving Generated Assembly Code

The generated Assembly (ASM) code is unique for different targets. Follow these steps to save the ASM code generated by the MMU Configuration File Editor:

1. From the **CodeWarrior IDE** menu bar, select **MMU Editor** > **Save ASM** to save the generated Assembly code. Alternatively, click the corresponding toolbar buttons in the MMU Configurator toolbar. A standard Save dialog box appears.

2. Specify the filename in the **File name** text box and click **Save** to save the generated code as a new file.

**NOTE**  The MMU Configuration File Editor regenerates the Assembly code when you change settings in the configuration pages or when you click Change on the **Program MATT** or **Data MATT** pages.

### 3.19.4.4.3 Saving Generated TCL Script

The generated TCL script can be executed within the Debugger Shell view, or the Debugger Shell can execute the generated TCL script as an initialization script for the target hardware. The generated TCL script is unique for different targets.

Follow these steps to save the TCL script generated by the MMU Configuration File Editor:

1. From the **CodeWarrior IDE** menu bar, select **MMU Editor** > **Save TCL** to save the generated TCL script. Alternatively, click the corresponding toolbar buttons in the MMU Configurator toolbar. A standard Save dialog box appears.

2. Specify the filename in the **File name** text box and click **Save** to save the generated code as a new file.
NOTE  The MMU Configuration File Editor regenerates the TCL script when you change settings in the configuration pages or when you click Change on the Program MATT or Data MATT pages.

### 3.19.5 Opening the MMU Configurator View

Using the New window to create an MMU configuration is just one way to work with MMU. Alternatively, you can open the MMU configurator view, such as during a debugging session. You can use this view to examine the current state of a thread's MMU configuration during the course of the debugging session. Also, you can detach the MMU configurator view into its own floating window and reposition that window into other collections of views.

NOTE  Because the MMU configurator view does not have an associated configuration file initially, the MMU tab appears in place of the tab that shows the name of the configuration file. Saving the MMU configurator view settings to a file (by selecting File > Save) replaces the MMU tab with the name of the saved configuration file.

To open the MMU Configurator view:

1. Start a debugging session.
2. In the Debug view of the Debug perspective, select the process for which you want to work with MMU.
3. Select Window > Show View > Other from the IDE menu bar.
   The Show View dialog box appears.
4. Expand the **Debug** group.
5. Select **MMU Configurator**.
6. Click **OK**.

The Show View dialog box closes. The **MMU Configurator** view appears, attached to an existing collection of views in the current perspective.

You just finished opening the **MMU Configurator** view. You can right-click the **MMU Configurator** tab to select the menu command that detaches the view into a floating window. Also, you can drag the **MMU Configurator** tab to a different collection of view tabs.

### 3.20 Multicore Debugging

The debugger 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 debugger can debug multiple cores by debugging multiple projects.
NOTE  CodeWarrior for Microcontrollers v10.x does not support debugging multiple projects on multiple cores in the same multi-core target. CodeWarrior for Microcontrollers v10.x supports creating and/or debugging Single Multi Processing (SMP) projects.

Configuring multi-core debugging involves these tasks:

- creating a project for each core
- configuring specific target settings for each project
- for some cores, specifying a configuration file for initializing multi-core debugging

You can use either the user interface or the Debugger Shell to perform multicore operations. In the user interface, you can access multicore operations from these locations in the Debug perspective:

- Run menu
- Debug view context menu
- Debug view toolbar
- Debug view toolbar pull-down menu

### 3.20.1 Multicore Suspend

To suspend execution of a core:

1. Enable multicore groups for multicore operations (see 3.21 Multicore Groups).
2. In the Debug view, select a thread that corresponds to a core for bareboard debugging.
3. Click Multicore Suspend.

Alternatively, in the 4.4 Command-Line Debugger Shell, select a thread using the switchtarget command and then use the mc::stop command to suspend execution of a core during a debugging session.

NOTE  If Use all cores is enabled, then all cores in the processor are suspended. Otherwise, if the core is in a multicore group, then all cores in the multicore group are suspended. In either case, cores that are not being debugged can still be affected by the command. If this is not the desired behavior, then reconfigure your multicore groups.

### 3.20.2 Multicore Resume

To resume execution of a core:

1. Enable multicore groups for multicore operations (see 3.21 Multicore Groups).
2. In the Debug view, select a thread that corresponds to a core for bareboard debugging.
3. Click **Multicore Resume**.

Alternatively, in the [4.4 Command-Line Debugger Shell](#), select a thread using the `switchtarget` command and then use the `mc::go` command to resume execution of a core during a debugging session.

**NOTE**

If **Use all cores** is enabled, then all cores in the processor are resumed. Otherwise, if the core is in a multicore group, then all cores in the Multicore Group are resumed. In either case, please note that cores that are not being debugged can still be affected by the command. If this is not the desired behavior, then reconfigure your multicore groups.

### 3.20.3 Multicore Terminate

To terminate execution of a core:

1. Enable multicore groups for multicore operations (see [3.21 Multicore Groups](#)).
2. In the **Debug** view, select a thread that corresponds to a core for bareboard debugging.
3. Click **Multicore Terminate**.

Alternatively, in the [4.4 Command-Line Debugger Shell](#), select a thread using the `switchtarget` command and then use the `mc::kill` command to terminate execution of a core during a debugging session.

**NOTE**

If **Use all cores** is enabled, then all Debug Threads for the processor will be terminated. Otherwise, if the core is in a multicore group then all threads corresponding to the cores in the multicore group will be terminated.

### 3.20.4 Multicore Restart

To restart execution of a core:

1. Enable multicore groups for multicore operations (see [3.21 Multicore Groups](#)).
2. In the **Debug** view, select a thread that corresponds to a core for bareboard debugging.
3. Click **Multicore Restart**.

Alternatively, in the [4.4 Command-Line Debugger Shell](#), select a thread using the `switchtarget` command and then use the `mc::restart` command to restart execution of a core during a debugging session.

### 3.21 Multicore Groups

The multicore grouping feature enables you to define multiple arbitrary groupings of cores and then perform multicore operations on the groups. Clicking the **Multicore Groups** button in the **Debug** view
toolbar enables you to create new multicore groups, see 3.21.1 Creating a Multicore Group. For more information on multicore debugging, see 3.20 Multicore Debugging.

Figure 3.69 Multicore Groups

The Multicore Groups drop-down menu provides the following options:

- **Use All Cores** — If the selected debug context is a multicore system, then all cores are used for multicore operations.
- **Disable Halt Groups** — Disables breakpoint halt groups, see 3.21.5 Multicore Breakpoint Halt Groups.
- **Limit new breakpoints to current group** — If selected, all new breakpoints set during a debug session are reproduced only on cores belonging to the group of the core on which the breakpoint is set.
- **Edit Target Types** — Opens the TargetTypes dialog box to add or remove target types, see 3.21.3 Editing a Multicore Group.
- **Edit Multicore Groups** — Opens the Multicore Groups dialog box to create multicore groups, see 3.21.1 Creating a Multicore Group. You can also use this option to modify existing multicore groups, see 3.21.2 Modifying a Multicore Group.

The Multicore Groups drop-down menu also shows the list of groups that are shown in the Multicore Groups dialog box.

### 3.21.1 Creating a Multicore Group

To create a multicore group:

1. Click the Multicore Groups button from the Debug view toolbar.

   The Multicore Groups dialog box appears.
Figure 3.70 Multicore Groups

- **New** button — Creates a new group using the **New Multicore Group** dialog box. The initial name of the group is the name unless the name is already in use. If the name is already in use then an index is appended to the group name. The initial enablement of the group and its descendants will be non-cores enabled, cores disabled. This guarantees an initial state with no error due to overlap.

- **Remove** button — Removes a selected group.

- **Remove All** button — Remove all groups.

- **Use all cores** check box — If checked, all cores are used for multicore operations irrespective of multicore groups.

- **Limit new breakpoints to current group** check box — If checked, all new breakpoints set during a debug session are reproduced only on cores belonging to the group of the core on which the breakpoint is set. When the **Use all cores** check box is checked, this check box is grayed and is not used on breakpoints filtering, as all cores are considered on the same group for multicore operations.

2. Click the **New** button.

   The **New Multicore Group** dialog box appears.
3. Select a target type from the list.
4. Click OK.

The group appears in the Multicore Groups dialog box.

5. Repeat Steps 2 - 4 to add more core groups for multicore operations.
6. Click OK.
You have just created multicore group.

### 3.21.2 Modifying a Multicore Group

You can also modify an existing multicore group.

**NOTE** You are not allowed to enable a group that overlaps with another group.

To modify a multicore group:

1. Select the **Edit Multicore Groups** option from the **Multicore Groups** drop-down menu in the **Debug** view toolbar to open **Multicore Groups** dialog box.
2. Check the cores you want to add to the multicore group.
3. Uncheck the cores you want to remove from the multicore group.
4. Click **OK**.
You have just modified multicore group.

### 3.21.3 Editing a Multicore Group

You can add custom target types by importing files from:

- JTAG configuration files
- Device Tree Blob files (for Power Architecture)

To add and remove system types from multicore groups:

1. Click the **Edit Target Type** option from the **Multicore Groups** pull-down menu.
   - The **Target Type** dialog box appears.
Figure 3.73 Multicore Target Types

- **Import** — Creates a custom target type by importing it from a configuration file.
- **Remove** — Removes a target type from the list.
- **Remove All** — Removes all target types from the list.

2. Click **Import**. The **Import Target Type** dialog box appears.

Figure 3.74 Import Target Type
3. Select a multicore configuration file, then click **Open**.

   The multicore appears in the **Target Types** dialog box.

   **Figure 3.75 Added Target Types**

4. Click **OK**.

   You just added custom target types.

### 3.21.4 Using Multicore Group Debugging Commands

Multicore Group features can also be accessed from the Debugger Shell command line. The following table lists and defines different multicore group debugging commands.
## Debugger

### Multicore Groups

Table 3.18  Multicore Group Debugging Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| mc::type         | **Syntax**
|                  | mc::type
|                  | Lists the available target types.                                                                                                          |
| mc::type import  | **Syntax**
|                  | mc::type import <filename>
|                  | Imports a new specified using the filename.                                                                                                 |
| mc::type remove  | **Syntax**
|                  | mc::type remove <filename>|<type-index> ...  
|                  | Removes the specified imported or types. Built-in target types cannot be removed and will return an error.                                  |
| mc::type removeall | **Syntax**
|                   | mc::type removeall
|                   | Removes all imported target types.                                                                                                          |
| mc::group        | **Syntax**
|                  | mc::group
|                  | Lists the defined groups.                                                                                                                   |
| mc::group new    | **Syntax**
|                  | mc::group new <type-name>|<type-index> |<name>  
|                  | Creates a new group for the system specified using the type-name or type-index. If no name is specified, then a unique default name is assigned to the group. |
### 3.21.5 Multicore Breakpoint Halt Groups

A halt group is a group of cores that will stop execution simultaneously whenever any one of the cores in the group hits a breakpoint. In multicore groups, each group can be configured as a run control group, a breakpoint halt group, or both. Figure 3.70 shows Multicore Groups dialog box.

The halt groups are configured on any applicable debug target. Similarly, whenever a debug session is launched, all applicable halt groups are applied to the debug target.

NOTE Multicore breakpoint halt groups are supported by P4080 processor only.

---

**Table 3.18 Multicore Group Debugging Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc::group rename</td>
<td>Syntax</td>
</tr>
<tr>
<td></td>
<td>mc::group rename &lt;name&gt;</td>
</tr>
<tr>
<td></td>
<td>Renames an existing group. Specifying a duplicate name results in an error.</td>
</tr>
<tr>
<td>mc::group remove</td>
<td>Syntax</td>
</tr>
<tr>
<td></td>
<td>mc::group remove &lt;name&gt;</td>
</tr>
<tr>
<td></td>
<td>Removes the specified group or groups.</td>
</tr>
<tr>
<td>mc::group removeall</td>
<td>Syntax</td>
</tr>
<tr>
<td></td>
<td>mc::group removeall</td>
</tr>
<tr>
<td></td>
<td>Removes all groups.</td>
</tr>
<tr>
<td>mc::group enable</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td>Syntax</td>
</tr>
<tr>
<td></td>
<td>mc::group enable</td>
</tr>
<tr>
<td></td>
<td>Enables or disables nodes in the group tree.</td>
</tr>
</tbody>
</table>
3.22 Multicore Reset

This CodeWarrior debugger feature enables you to configure reset and run out of reset action for your target system. It also enables you to configure your target system to perform system reset action.

**NOTE**  The system reset action is applicable for initial launch only.

To specify reset setting for cores in a multicore environment:
1. Go to 2.17.5 Remote Systems View.
2. Right-click a remote system and select **Properties** from the context menu.
   The Properties for <Remote System> dialog box appears.

**Figure 3.76 Properties for <Remote System> Dialog Box**

3. Click the **Initialization** tab.
   The initialization settings page appears.
**Remote System Explorer Configurator initial launch** — The initial launch session for the remote system, after all other launched sessions were terminated.

**System Reset** — Resets the entire Remote System. This option is available only if the processor supports system reset. Reset system is executed only for the initial launch.

**Core reset** — Independently reset one or more cores from the Remote System. This option is available only if the processor supports core reset. Use this option in RSE configuration if you want to independently reset the core on launch or restart. Initial launches with system reset and core reset options will execute only the system reset.

**NOTE** In StarCore, the **Core reset** column is referred as **Processor reset**.

**Run out of reset** — Puts a core in run mode after reset. This option is enabled only if system reset or core reset is checked.

**Initialize target** — Enables **initialize target script** configuration

**Initialize target script** — Script to initialize the target. This option is enabled only if initialize target is checked. Target initialization scripts and reset cores are applied to cores being launched.

4. Check the **Execute system reset** to perform system reset. The system reset applies only to initial launch.

5. Check the **Core reset** check box adjacent to the core on which you want to perform a reset action.

6. Check the **Run out of reset** check box adjacent to the core on which you want to perform run out of reset action.

7. Click **OK**.
NOTE Initialization files are executed only for cores selected for debug.

You have just specified reset setting for cores in a multicore environment.

### 3.22.1 On Demand Reset

The on demand settings are serialized when the user performs the reset action. The **Reload** button allows user to load the settings from the remote system configuration. The on demand reset configurations apply to the whole system, these configurations are not filtered to the active debug context. The initialization files are executed only for cores under debug.

You can access the **Reset** command from the **Run** menu in the debug view.

#### Figure 3.78  On Demand Reset

![On Demand Reset](image)

### 3.23  Path Mappings

The Path Mapping settings are used in IDE to resolve a partial or absolute path from a binary executable during debugging to effectively locate a source file. A binary executable used for debugging typically contains a list of source files in its debugger that were used to build the executable. The source file list is used by the debugger to provide source level debugging. The CodeWarrior IDE supports automatic as well as manual path mapping.

In this section:

- [3.23.1 Automatic Path Mappings](#)
- [3.23.2 Manual Path Mappings](#)
3.23.1 Automatic Path Mappings

The Automatic Path Mapping feature focuses on reducing as much as possible the manual steps required by the user to setup the path mapping settings in order to support source level debugging.

For automatic path mapping:
1. In the CodeWarrior Projects view, expand Binaries folder and right-click on the *.eld file.
2. Select Properties from the shortcut menu that appears.
   - The Properties <Properties for *.eld> dialog box appears.
3. Select Path Mappings from the list.
   - The Path Mappings page appears. The Path Mapping Configuration page displays every path mapping settings for the launch configurations associated with a project.

Figure 3.79 Automatic Path Mapping

You can edit either a single set of settings for all launch configurations associated with a project or the settings for a given launch configuration by selecting the appropriate value from the launch configuration combo box.

Under each path mapping, the table displays a list of source files that exist in the binary executable that share the same source mapping prefix. In the Local Path column, a green (✓) is displayed if the
file exists after being mapped by the destination path or a red (x) if it does not. Also, the local path itself is displayed in red if it does not exist on the local file system.

A default folder named **Files Not Mapped** is created if the user explicitly removes existing mappings. All unmapped files that are not found on the file system are automatically shown under this folder.

The following table describes various options available in the Path Mappings page.

**Table 3.19 Automatic Path Mappings Options**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Correct</td>
<td>The Auto Correct button automatically iterate through all the files not found on the file system and attempt to group them with their common prefix. This action often generates satisfactory results from the source files listed in the binaries so that the manual steps required by the user are kept at a minimum.</td>
</tr>
<tr>
<td>Add</td>
<td>The Add button allows you to create a new Path Mapping entry. If any paths are selected, the dialog will be pre-initialized with their common prefix.</td>
</tr>
<tr>
<td>Remove</td>
<td>The Remove button allows you to remove any path mapping or default entry.</td>
</tr>
<tr>
<td>Edit</td>
<td>The Edit button allows you to change the values of the selected path mapping entry. Editing non-path mapping entry is not supported.</td>
</tr>
<tr>
<td>Up</td>
<td>The Up button allows the user to reorder the entries by moving the selected entry up in the list. Note that path mappings need always to be grouped together, and as such moving up the top most path mapping will always move its siblings above the preceding entry as well.</td>
</tr>
</tbody>
</table>
If you create a new path mappings manually from the source lookup path, the source files are automatically resorted to their most likely path mapping parent.

4. Click **OK**.

The Path Mappings dialog box closes.

### 3.23.2 Manual Path Mappings

You need to specify the source lookup path in terms of the compilation path and the local file-system path for the newly imported executable file. The CodeWarrior debugger uses both of these paths to debug the executable file. The compilation path is the path to the original project that built the executable file. If the original project is from an IDE on a different computer, you specify the compilation path in terms of the file system on that computer. The local file-system path is the path to the project that the CodeWarrior IDE creates in order to debug the executable file. Path mappings can be added per launch configuration or global, per workspace. In the latest case the mapping will be valid for all the projects within the workspace.

To add a path mapping to a launch configuration:

1. Click the **Source** tab of the **Debug Configurations** dialog box.

   The corresponding page appears.

---

### Table 3.19 Automatic Path Mappings Options

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Down</td>
<td>The Down button allows the user to reorder the entries by moving the selected entry down in the list. Note that path mappings need always to be grouped together, and as such moving down the bottom most path mapping will always move its siblings below the following entry as well.</td>
</tr>
<tr>
<td>Restore Defaults</td>
<td>The Restore Defaults button resets the launch configuration path mappings settings to their previous values, including the library path mapping automatically generated by the APM plug-in.</td>
</tr>
</tbody>
</table>
2. Click **Add**.
   The Add Source dialog box appears.
3. Select **Path Mapping** (Figure 3.81).
4. Click OK.

The Path Mappings dialog box appears.

5. Specify the Path mappings name in Name text box.
6. Click Add.
7. In the **Compilation path** text box, enter the path to the parent project of the executable file, relative to
the computer that generated the file.

For example, the computer on which you debug the executable file is not the same computer that
generated that executable file. On the computer that generated the executable file, the path to the
parent project is \D:\workspace\originalproject. Enter this path in the Compilation path text box.

**TIP** You can use the IDE to discover the path to the parent project of the executable file, relative to
the computer that generated the file. In the C/C++ Projects view of the C/C++ perspective, expand
the project that contains the executable file that you want to debug. Next, expand the group that
has the name of the executable file itself. A list of paths appears, relative to the computer that
generated the file. Search this list for the names of source files used to build the executable file.
The path to the parent project of one of these source files is the path you should enter in the
Compilation path text box.

8. In the **Local file system path** text box, enter the path to the parent project of the executable file,
relative to your computer. Alternatively, click the Browse button to specify the parent project.

Suppose the computer on which you debug the executable file is not the same computer that generated
that executable file. On your current computer, the path to the parent project of the executable file is
\C:\projects\thisproject. Enter this path in the Local file system path text box.

9. Click **OK**.

    The Path Mappings dialog box closes. The mapping information now appears under the path mapping
shown in the **Source Lookup Path** list of the Source page.

10. If needed, change the order in which the IDE searches the paths.

    The IDE searches the paths in the order shown in the Source Lookup Path list, stopping at the first
match. To change this order, select a path, then click the Up or Down button to change its position in
the list.

11. Click **Apply**.

    The IDE saves your changes.

### 3.23.2.1 Adding Path Mapping to Workspace

To add a path mapping to the workspace:

1. Select **Window > Preferences** from the CodeWarrior IDE menu bar.

    The Preferences dialog box appears.

2. Expand **C/C++ > Debug > Common Source Lookup Path**.
3. Repeat step 2, from the previous section for adding a path mapping for a single launch configuration.

### 3.24 Redirecting Standard Output Streams to Socket

This CodeWarrior feature enables a user to redirect standard output (stdout, stderr) of a process being debugged to a user specified socket.

To specify the initial connection redirection settings:

1. In the CodeWarrior project window, right click on the project folder to display the context menu.
2. Select **Debug As > Debug Configurations** from the context menu.

   The **Debug Configurations** dialog box appears. The left panel of the **Debug Configurations** dialog box has a list of debug configurations that apply to the current application.
3. Expand the **CodeWarrior** configuration tree.

4. Click the name of the debug configuration, from the expanded list, for which you want to modify debugger settings.

   The right panel of the **Debug Configurations** dialog box shows the settings for the configuration that you selected.

5. Click the **Common** tab.

   The common settings are displayed in the right panel of the **Debug Configurations** dialog box.

### Figure 3.84  Debug Configurations Dialog Box

6. Check the **Port** Check box.

   The **Act as Server** or **Hostname/IP address** options are enabled.

7. Type the port number in the **Port** text box.

8. Select **Act as Server** to redirect the output from this process to a local server socket bound to the specified port.

9. Select **Hostname/IP address** to redirect the output from this process to a server socket located on the specified host and bound to the specified port. The debugger will connect and write to this server socket via a client socket created on an ephemeral port.
10. Click **Apply**.
   The changes are applied to the selected debug configuration.

**NOTE** You can also use the `redirect` command in a debugger shell to redirect standard output streams to a socket.

### 3.25 Refreshing Data During Run Time

This debugger feature refreshes the memory and registers data non-intrusively during run time. The data is automatically refreshed after a specified interval during run time.

You can also refresh data by clicking the **Refresh** button from a view toolbar. If you select the **Refresh While Running** option from the pull-down menu, the data is refreshed automatically after the interval specified in debug configurations settings.

The data can be refreshed for the following views:

- Memory view
- Variable view
- Registers view

To specify a time interval to automatically refresh view data during run time:

1. In the CodeWarrior project window, right click on the project folder to display the context menu.
2. Select **Debug As > Debug Configurations** from the context menu.
   The **Debug Configurations** dialog box appears. The left panel of the **Debug Configurations** dialog box lists debug configurations that apply to the current project.
3. Expand the **CodeWarrior** tree.
4. Click the name of the debug configuration, from the expanded list, for which you want to modify debugger settings.
   The right panel of the **Debug Configurations** dialog box shows the settings for the configuration that you selected.
5. Click the **Debugger** tab.
6. Click the **Debug** tab from the **Debugger Options** group.
7. Type the refresh interval in the **Refresh while running period (seconds)** text box.

8. Click **Apply**.

   The changes are applied to the selected debug configuration.

### 3.26 Registers View

The **Registers** view lists information about the registers in a selected stack frame. Values that have changed are highlighted in the **Registers** view when your program stops.

You can use the **Registers** view to:

- add, edit, or remove groups of registers
- view register details, such as explanations of a register’s bit fields and values
- change register values
- import/export register data

You can also change the number system in which the debugger displays register values. These number systems are supported:
• Binary
• Decimal
• Hexadecimal
• Natural

**NOTE** *Natural* represents the register’s default format, which is defined by the debugger implementation. *Natural* displays the register in the format that is optionally specified in the debug database for each register (by default *hexadecimal*).

The Registers view also allows you to cast existing data type to complex data types that may or may not exist in the debugged executable. For more information on casting a data type, see [Cast to Type](#).

**NOTE** Casting a register to a type requires that the size of the register must match the size of the type, otherwise the cast will fail. Therefore, if the type is a complex one (e.g. structure, union), it should be declared first to avoid padding done by compilers.

**Figure 3.86 Registers View**

![Registers View](image)

### 3.26.1 Displaying the Registers View

To display the Registers view:
1. Switch to the **Debug** perspective.
2. Select *Window > Show View > Registers* from the IDE menu bar.

### 3.26.2 Viewing Registers

To view registers content:
1. Open the **Registers** view (Figure 3.86).
2. Expand a register group.
   
   Expanding a group shows its content by register name and the content of each register in the group.

### 3.26.3 Changing Register Values

To change the value of a register:
1. Open the **Registers** view (Figure 3.86).
2. Expand the hierarchical list to reveal the register whose value you want to modify.
3. Right-click the register value that you want to change and choose **Change Value** from the context menu that appears.
   
   The **Set Value** dialog box appears.

![Figure 3.87 Set Value Dialog Box](image)

4. Type a new value in the **Enter a new value for ALTCAR** text box.
5. Click **OK**.

The debugger assigns the specified value to the selected register.

**TIP** Alternatively, you can click on the value and edit it to change the Registers value.
3.26.4 Exporting Registers

The export operation generates two files:

- a *.regs file that contains the registers information in XML format which is also used by the import operation.
- a *.csv file that contains the registers information in plain text CSV (comma-separated values) format that can be used for easy visual inspection in an external text editor or MS Excel/Open Office.

To export register data to a file:

1. Open the Registers view (Figure 3.86).
2. Click the Export registers button in the Registers view toolbar.

   The Export Registers dialog box appears.

   Figure 3.88 Export Registers Dialog Box

   - Registers group — Controls the scope of export operation. Selecting the All option exports all registers in the Register view. Selecting the Selected option exports selected registers. If a register group is selected in the Register view then the entire register tree, starting at the selected node, is exported.
NOTE   The Selected option is disabled if no register is selected in the Registers view.

- **File** text box — Specifies the name of the file to store the exported register information.
- **Include register information** check box — Check this check box to export the location information for registers.
- **Overwrite existing** check box — Check this check box to overwrite an existing file.
- **Cancel on error** check box — Check this check box to stop the export operation upon encountering any error.

3. Click **Finish**.

### 3.26.5 Importing Registers

To import register data from a file:

1. Open the **Registers** view (**Figure 3.86**).
2. Click the **Import registers** button in the **Registers** view toolbar.

   The **Import Registers** dialog box appears.

**Figure 3.89 Import Registers Dialog Box**
File drop-down list — Specifies the name of the register data file to import register information.

Import all registers — Selecting this option allows you to import all registers from the register data file.

Import selected registers — Selecting this option allows you to select registers you want to import.

Verify check box — When checked, a register write to the target is followed by a read and a comparison against the written value. This ensures that the import operation on the register is successful.

Cancel on error check box — Check this check box to stop the import operation upon encountering any error.

3. Click Finish.

### 3.26.6 Changing Register Data Display Format

You can change the format in which the debugger displays the contents of registers. For example, you can specify that a register's contents be displayed in hexadecimal, rather than binary. The debugger provides these data formats:

- Binary
- Natural
- Decimal
- Hexadecimal

To change register display format:

1. Open the Registers view.
2. Expand the hierarchical list to reveal the register for which you want to change the display format.
3. Select the register value that you want to view in a different format.
   - The value highlights.
4. Right-click and choose Format > dataformat from the context menu that appears, where dataformat is the data format in which you want to view the register value.
   - The register value changes format.

### 3.27 Register Details View

The Register Details view (Figure 3.90) shows detailed information for a selected register during a debug session. The Register Details pane shows the following information for a register:

- Bit Fields — Shows a graphical representation of the selected register's bit values. This graphical representation shows how the register organizes bits. You can use this representation to select and
Debugger

Register Details View

change the register's bit values. Hover the cursor over each part of the graphical representation to see additional information.

- Actions — Enables you to perform various operations on the selected register's bit-field values.
- Description — Shows explanatory information for the selected register. The information includes name, current value, description, and bit-field explanations and values of the selected register.

NOTE The default display of the Registers view shows register details, such as Bit Fields, Description, and Actions. To see more register contents, use Registers view pull-down to select Layout > Registers View Only. To restore the register details, use the view's toolbar menu to select a different menu command.

To open the Register Details view, right-click on a register name in the Registers view and select Show Details As > Register Details pane from the context menu that appears.

Figure 3.90 Register Details View

NOTE If the Registers view loses focus, all pending changes are discarded. For more information, see the <Product> Targeting Manual.

Following sections will help you with more details on the Register Details view:
3.27.1 Viewing Register Details Offline

The Register Details view also allows you to browse and edit registers without an active debug session. Select File > Open File and select the previously exported registry dump (.reg) file, to launch the Register Details view for viewing and editing registry details offline. For details on how to create a .reg file, refer 3.26.4 Exporting Registers.

Figure 3.91  Viewing Register Details Offline - Visual Editor

The following table provides details of the various options that help you load binary or plain hex text register dump files in the Register Details view.
Upon loading the register dump file, you can view and edit the register details in the Visual Editor mode and click Write to save the updated details to the .reg file. Alternatively, you can click the Text Editor tab to edit the registry dump file.

Figure 3.92 Viewing Register Details Offline - Text Editor

### 3.27.2 Customizing Register Details Pane

You can customize background color, fonts, and foreground color for Register Details pane. To customize Register Details pane:
1. Open the Registers view.
2. Select Window > Preferences from the IDE menu bar.
   The Preferences dialog box appears.
3. Select General > Appearance > Colors and Fonts from the left pane of the Preferences dialog box.
   The color and fonts preferences appear in the right pane of the Preferences dialog box.

**Figure 3.93 Preferences Dialog Box**

4. Expand Debug > Register Details tree controls.
5. Modify colors and fonts settings to suit your needs.
6. Click Apply.
7. Click OK.
   You have just customized Register Details pane.

### 3.28 Remote Launch

The remote launch feature of CodeWarrior allows launch configurations to be executed remotely. A Jython script is used to declare which launch configuration to use as a basis and provides points of interaction with the executing launch configuration if desired.

The launch scripts can be submitted to CodeWarrior in these ways:

- The submissions web page
Debugger
Remote Launch

- Java and/or Python Clients

CodeWarrior requires a launch configuration to be set up on the host CodeWarrior instance in order to execute. The remote launch script will make a copy of that launch configuration, execute it, and then delete the configuration.

3.28.1 Remote Launch View

The Remote Launch view (Figure 3.95) displays the remote launch configurations for the project. The Enable Remote Launch option in the pull-down menu is a toggle button to enable or disable remote launch view. The Open Remote Launch Web Page opens the CodeWarrior Remote Launch web page where you can submit remote launch scripts.

NOTE  Click the Help/Examples link in the CodeWarrior Remote Launch web page for remote launch examples.

To display the Remote Launch view:
1. Select Window > Show View > Others from the IDE menu bar.
   The Show View dialog box appears.

Figure 3.94  Show View Dialog Box

2. Expand the Debug tree control.
4. Click **OK**.

   The Remote Launch view appears.

**Figure 3.95 Remote Launch View**

The **Jython Consoles** view is a scripting view where you can work with Jython scripts. You can use this view to test remote launches.

**Figure 3.96 Jython Consoles View**

### 3.29 Stack Crawls

CodeWarrior allows you to limit the depth of stack crawls in the debugger. You can limit the stack crawl depth in two ways:

- **One Frame Mode**
- **Global Preference**
3.29.1 One Frame Mode

In the one frame mode, only the topmost frame is retrieved by the debugger engine and displayed in the Debug view and in the debugger shell. The following figure shows selecting the one-frame mode from the Debug view.

Figure 3.97 Selecting One Frame Mode

The Show Only One Frame menu option is a two-state menu item which uses a checkmark to indicate the state. If the Show Only One Frame option is selected then a checkmark appears before the option and only one frame is displayed. The following figure shows the stack crawl in a one frame mode.

Figure 3.98 Stack Crawls in One Frame Mode

The decorator 1 (shown at left) in the stack frame element indicates that the stack crawl is limited to one.

3.29.2 Global Preference

CodeWarrior exposes a global preference that allows you to specify the maximum number of frames that will be displayed in the debug view. This limit is merely a display limit and does not restrict the depth of
the stack crawl calculated by the debugger engine. This mode allows you to manage the amount of content in the **Debug** view.

To specify the maximum frames in the global preference window:

1. Select **Window > Preferences** from the IDE menu bar.
   
   The **Preferences** dialog box appears.

2. Expand the C/C++ group and select **Debug** group.
   
   General C/C++ debug settings appears in the left-panel of the **Preferences** dialog box.

**Figure 3.99 Preferences Dialog Box**

3. Type the maximum frame depth in the Maximum stack crawl depth text box.

**NOTE**  The upper limit for maximum frame depth is 100.

4. Click **Apply**.

5. Click **OK**.

Changing the stack crawl preference does not have an immediate effect on stack crawls currently displayed in the **Debug** view. The limit takes effect the next time the stack crawl is constructed, which happens either on the next suspended event, or after toggling in or out of the one frame mode.
When the actual stack crawl depth of a core exceeds the number of frames specified in the global preference, the stack crawl contains a final frame that is labeled ... (Figure 3.100). This label indicates that frames are being omitted from display.

**Figure 3.100 Exceeding Stack Crawl Depth**

### 3.30 Symbolics

Use the Symbolics page to specify whether the debugger keeps symbolics in memory. Symbolics represent an application's debugging and symbolic information. Keeping symbolics in memory, known as caching symbolics, helps when you debug a large application.

Suppose that the debugger loads symbolics for a large application, but does not download program code and data to a hardware device. Also, suppose that the debugger uses custom makefiles with several build steps in order to generate the large application. In this situation, caching symbolics helps speed up the debugging process. The debugger uses the cached symbolics during subsequent debugging sessions. Otherwise, the debugger spends significant time creating an in-memory representation of symbolics during subsequent debugging sessions.

**NOTE** Caching symbolics provides the most benefit for large applications because doing so speeds up application-launch times. If you debug a small application, caching symbolics does not significantly improve launch times.

To open the Symbolics page:

1. Select **Run > Debug Configurations** from the IDE menu bar.
   
   The Debug Configurations dialog box appears. The left side of this dialog box has a list of debug configurations that apply to the current application.

2. Expand the CodeWarrior configuration.
3. From the expanded list, select the debug configuration that you want to modify.
4. Select the **Debugger** tab to view the corresponding debugger settings page.

5. Select the **Symbolics** tab in the **Debugger Options** group on the page.
   
   The **Symbolics** page appears.

**Figure 3.101 Symbolics Page**

![Symbolics Page](image)

---

### 3.31 System Browser View

The **System Browser** view is a framework for displaying embedded operating system (OS) information. A CodeWarrior user working with a target running an embedded OS can use the **System Browser** view to gather information about the OS during a debug session.

The **System Browser** view enables the user to debug specific threads, tasks, and processes running in the OS.

#### 3.31.1 Opening the System Browser View

To open the **System Browser** view:

1. Start a debugging session.

2. Select **Window > Show View > Other** from the IDE menu bar.
   
   The **Show View** dialog box appears.
3. Expand the **Debug** group.
4. Select **System Browser**.
5. Click **OK**.

The **System Browser** view appears.

**NOTE** The **System Browser** view shows information only when there is an OS running on the target being debugged.
3.32 Target Connection Lost

This feature enables you to configure the debugger’s behavior when connection to the target is lost, such as low power modes, target power switched off, target changed communication speed, or disconnected run control. This feature enables you to configure the debugger to close the connection or automatically reconnect with a specified time-out value.

To configure target connection lost settings for debugger:

2. Right-click a remote system name and select Properties from the context menu.
   The Properties for <Remote System> dialog box appears.
3. In the right-pane, select Advanced tab.
   The advanced connection settings appear under the Advanced tab.

Figure 3.104 Advanced Settings

4. Specify the target connection lost settings to suit your needs.
   - **Try to reconnect** — Whenever target connection is lost, the debugger doesn’t close the debug session but waits for the connection to be restored. A time-out may be specified to limit the waiting time. When the time-out expires, the debugger closes the debug session.
   - **Terminate the debug session** — Select this option to terminate the debug session when target connection is lost.
   - **Ask me** — Select this option to prompt the user for an action when target connection is lost.
5. Click OK.

You have just configured target connection lost settings for debugger.
3.33 Target Initialization Files

A target initialization file contains commands that initialize registers, memory locations, and other components on a target board. The most common use case is to have the CodeWarrior debugger execute a target initialization file immediately before the debugger downloads a bare board binary to a target board. The commands in a target initialization file put a board in the state required to debug a bare board program.

**NOTE** The target board can be initialized either by the debugger (by using an initialization file), or by an external bootloader or OS (U-Boot, Linux). In both cases, the extra use of an initialization file is necessary for debugger-specific settings (for example, silicon workarounds needed for the debug features).

3.33.1 Selecting Target Initialization File

A target initialization file is a command file that the CodeWarrior debugger executes each time the launch configuration to which the initialization file is assigned is debugged. You can use the target initialization file for all launch configuration types (Attach, Connect and Download). The target initialization file is executed after the connection to the target is established, but before the download operation takes place.

The debugger executes the commands in the target initialization file using the target connection protocol, such as a JTAG run-control device.

**NOTE** You do not need to use an initialization file if you debug using the CodeWarrior TRK debug protocol.

To select a target initialization file, follow these steps:

1. Go to the 2.17.5 Remote Systems View.
2. Right-click a remote system name and select Properties from the context menu.
   - The Properties for <Remote System> dialog box appears.
3. Click Edit next to the drop-down list.
   - The Properties for <remote system> window appears.
4. Select initialization tab.
5. Click the ellipsis button in the Initialize target script column corresponding to the core for which you want to select the target initialization file.

**TIP** Single-click in the specified cell of the Initialize target script column for the Ellipsis button to appear.

The Target Initialization File dialog box appears.

**Figure 3.106 Target Initialization File Dialog Box**

a. Check the File check box to activate the text box.

b. Type the target initialization file path in the File text box. You can use the Workspace, File System, or Variables buttons to select the desired file.
c. Click OK.
   The target initialization file path appears in the Initialize target column.
6. Click OK.
You have just selected a target initialization file.

3.34 Target Tasks View

In order to run a hardware-diagnostic or memory operation, you must first open the Target Tasks view.
To open the Target Tasks view:
1. Select Window > Show View > Other from the IDE menu bar.
   The Show View dialog box appears.
2. Expand the Debug group.
3. Select Target Tasks.
4. Click OK.
The Target Tasks View appears in the Debug perspective.

Figure 3.107 Target Task View

3.34.1 Exporting Target Tasks

You can export a target task to an external file. The exported task is stored in XML format.
To export a target task:
1. Select the target task in the Target Task view.
2. Click the Export button from the Target Task view toolbar. Alternatively, right-click the target task and select Export from the context menu.
   The Save As dialog box appears.
3. Type a file name in the File name drop-down list.
4. Click Save.
3.34.2 Importing Target Tasks

You can import a target task from an external file. To import a target task:

1. Click the Import button in the Target Task view toolbar. Alternatively, right-click in the Target Task view and select Import from the context menu.
   - The Open dialog box appears.
2. Select a target task file.
3. Click Open.

3.35 Variables

The Variables view shows all global and static variables for each process that you debug. Use the view to observe changes in variable values as the program executes.

Figure 3.108 Variables View

![Variables View](image)

3.35.1 Opening the Variables View

Use the Variables view to display information about the variables in the currently-selected stack frame.

To open the Variables view:

1. Select Window > Show View > Other from the IDE menu bar.
   - The Show View dialog box appears.
2. Expand the Debug group.
3. Select Variables.
4. Click OK.
The Variable view appears.

### 3.35.2 Adding Variable Location to the View

A variable location can be a memory address or a register. This can change from one execution point to another in the target application. The return value will be a hexadecimal (“0x...”) value if the variable is in memory; if it is in a register, $<\text{register-name}>$ will be returned.

To add the variable location column in the **Variables** view:
1. Click the **Variables** view pull-down menu button.
2. Click **Layout**.
3. Click **Select Columns**.
   - The **Select Columns** dialog box appears.

**Figure 3.109  Select Columns**

![Select Columns dialog box](image)

NOTE In **Variables** view, Freescale CDT (C/C++ Development toolkit) does not support the **Actual Type** column. This column is relevant for C++ only when RTTI (Run-time type information)
is used. Check the **Window > Preferences... > C/C++ > Debug > CodeWarrior Debugger > Attempt to show the dynamic runtime type of objects** option to get declared types displaying the Actual types.

4. Click the **Location** check box.

5. Click **OK**.

**TIP** You can use the **Select Columns** dialog box to enable or disable different columns displayed in the **Variables** view.

The variable location column appears in the **Variables** view.

### 3.35.3 Manipulating Variable Values

You can change the way the **Variables** view displays a variable value. To manipulate the format of a variable value, click **Format** from the context menu and select any of the following formats:

- Binary
- Natural
- Decimal
- Hexadecimal
- Fractional

### 3.35.3.1 Fractional Variable Formats

In addition to the Natural, Binary, Decimal, and Hexadecimal variable formats, CodeWarrior supports an additional class of custom fractional formats called **Qn**. **Qn** is a floating point representation of a fractional or fixed point number where n signifies the number of fractional bits (the number of bits to the right of the binary point).

CodeWarrior supports fractional formats ranging from Q0 to Q31.

To change the variable display format to fractional format:

1. Open the **Variable** view, see 3.35.1 Opening the Variables View.

2. Right-click a variable in the **Variable** view.

   The context menu appears.
3. Select **Format > Fractional > Qn** (where n = 0 to 31).

   The variable value will be displayed in the specified Qn format.

**NOTE**  
The Qn formats are enabled or disabled (grayed) depending on the size of the variable.
- Q0 - Q7 available for 1 byte variables
- Q0 - Q15 available for 2 byte variables
- Q0 - Q31 available for 4 byte variables

### 3.35.4 Adding Global Variables

You can add global variables to the **Variables** view.

To add a global variable:

1. Select **Project > Debug** from the IDE menu bar.
   
   A debugging session starts.

2. In the **Variables view** toolbar, click the **Add Global Variables** button.
   
   The Add Globals dialog box appears.
TIP You can also add a global variable using the Add Global Variable command from the context menu.

Figure 3.111 Add Globals Dialog Box

3. Specify a search criteria in the Search Textbox to filter the list of variables.
4. Select the global variable to be added.

NOTE Global variables of other executables (other than the main one) are also listed in the Global Variables dialog box.

5. Click OK.
   You have just added global variable.

3.35.5 Cast to Type

This feature allows the user to cast the type of a variable to a particular type. The Cast to Type dialog box enables the user to filter the type list based on a search pattern specified in the search text box.

To cast a variable to a selected type:
1. Open the Variable view, see 3.35.1 Opening the Variables View.
2. Right-click a variable in the Variable view.
   The context menu appears.
3. Select **Cast to Type**.

   The **Cast to Type** dialog box appears.

**Figure 3.112  Cast to Type Dialog Box**

![Cast to Type Dialog Box](image)

4. Specify a search pattern in the Search text box.

   The matching types appear in the **Matching Items** list box.

5. Select a type from the **Matching Items** list box.

6. Click **OK**.

You have just cast a variable to a selected type.

### 3.36 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 variable values and alter the flow of normal program execution.

You can set a watchpoint from the:
- Breakpoints view
- Memory view
The debugger handles both watchpoints and breakpoints in a similar way. You use the Breakpoints view to manage both types. For example, you use the Breakpoints view to add, remove, enable, and disable both watchpoints and breakpoints.

The debugger attempts to set the watchpoint if a session is in progress based on the active debugging context (the active context is the selected project in the Debug view). If the debugger sets the watchpoint when no debugging session is in progress, or when re-starting a debugging session, the debugger attempts to set the watchpoint at startup as it does for breakpoints.

The Problems view displays error messages when the debugger fails to set a watchpoint. For example, if you set watchpoints on overlapping memory ranges, or if a watchpoint falls out of execution scope, an error message appears in the Problems view. You can use this view to see additional information about the error.

### 3.36.1 Setting a Watchpoint

Use the Add Watchpoint dialog box (Figure 3.113) to set a watchpoint. When the value at the memory address on which you set a watchpoint changes, your program's execution halts and the debugger takes control.

To set a watchpoint:
1. Open the Debug perspective.
2. Click one of these tabs:
   - Breakpoints
   - Memory
   - Variables

   The selected view comes forward.
3. Right-click within the selected view.

   The part of the view in which to right-click varies depending upon the type of view:
   - Breakpoints — click the Breakpoints View pull-down menu button and select Add Watchpoint.
   - Memory — the addressable unit or range of units on which you want to set the watchpoint.
   - Variables — a global variable.
4. Select Add Watchpoint (C/C++) from the context menu that appears.

   The Add Watchpoint dialog box appears.
The **Breakpoints** view shows information about the newly set watchpoint and the number of addressable units that the watchpoint monitors.

The **Problems** view shows error messages if the debugger fails to set a watchpoint.

### 3.36.2 Creating Watchpoint

Use the **Add Watchpoint** dialog box to create a watchpoint. The debugger sets the watchpoint according to the settings that you specify in this dialog box.

**Figure 3.113  Add Watchpoint Dialog Box**

![Add Watchpoint Dialog Box](image)

The following table describes each option.
Table 3.21 Add Watchpoint Dialog Box Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| Expression to Watch | Enter an expression that evaluates to an address on the target device. The debugger displays an error message when the specified expression evaluates to an invalid address.  
You can enter these types of expressions:  
- An r-value, such as &variable  
- A register-based expression. Use the $ character to denote register names. For example, enter $SP-12 to have the debugger set a watchpoint on the stack-pointer address minus 12 bytes.  
The Add Watchpoints dialog box does not support entering expressions that evaluate to registers. |
| Memory Space   | Check this box if you want to specify the memory space in which the watchpoint is set.  
The dropdown menu to the right of the check box lists each memory space available for the active debug context.  
If no debug session is active, the dropdown menu is empty and lets you enter text. This feature lets you set a memory-space-qualified watchpoint before starting a debug session. |
| Units          | Checked—Enter the number of addressable units that the watchpoint monitors.  
Cleared—Set the watchpoint on the entire range of memory occupied by the variable. |
| Write          | Checked—The watchpoint monitors write activity on the specified memory space and address range.  
Cleared—The watchpoint does not monitor write activity. |
| Read           | Checked—The watchpoint monitors read activity on the specified memory space and address range.  
Cleared—The watchpoint does not monitor read activity. |

3.36.3 Viewing Watchpoint Properties

After you set a watchpoint, you can view its properties. To view properties for a watchpoint
1. Right-click the watchpoint in the Breakpoints view.
2. Select Properties from the context menu that appears.
   
   The Properties for dialog box appears.

**Figure 3.114 Properties for Dialog Box**

### 3.36.4 Modifying Watchpoint Properties

After you set a watchpoint, you can modify its properties. To modify properties for a watchpoint:

1. Right-click the watchpoint in the Breakpoints view.
2. Select the Breakpoint Properties from the context menu that appears.
   
   The Properties for C/C++ Watchpoint window appears.
3. Select Common from the left pane of the Properties for C/C++ Watchpoint window.
   
   The right pane displays the common properties for the watchpoint.
4. Edit the values in the fields.
5. Click OK.
You have just modified properties for a watchpoint.

3.36.5 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, right-click and select Disable from the context menu that appears.

3.36.6 Enabling a Watchpoint

Enable a watchpoint to have it halt program execution when its associated memory location changes value. Enabling a watchpoint that you previously disabled is easier than clearing it and re-creating it from scratch.

To enable a watchpoint, select its name in the Breakpoints window, right-click and select Enable from the context menu that appears.
3.36.7 Remove a Watchpoint

To remove a watchpoint in the Breakpoints view, select its name from the list, right-click and select Remove from the context menu that appears.

3.36.8 Remove All Watchpoints

To remove all watchpoints, right-click inside the Breakpoints view and select Remove All from the context menu that appears. The Breakpoints view reflects your changes.
Debugger Shell

CodeWarrior supports a command-line interface to some of its features including the debugger. You can use the command-line interface together with TCL scripting engine. You can even issue a command that saves the command-line activity to a log file.

The Debugger Shell view is used to issue command lines to the IDE. For example, you enter the command debug in this window to start a debugging session. The window lists the standard output and standard error streams of command-line activity.

**Figure 4.1 Debugger Shell View**

To open the Debugger Shell view, perform these steps.

1. Switch the IDE to the Debug perspective and start a debugging session.
2. Select Window > Show View > Debugger Shell.

The Debugger Shell view appears (Figure 4.1).

**NOTE** Alternatively, select Window > Show View > Other. Expand the Debug tree control in the Show View dialog box, select Debugger Shell, and click OK.
To issue a command-line command, type the desired command at the command prompt (`%`) in the *Debugger Shell* view, then press Enter or Return. The command-line debugger executes the specified command.

If you work with hardware as part of your project, you can use the command-line debugger to issue commands to the debugger while the hardware is running.

**NOTE** To list the commands the command-line debugger supports, type `help` at the command prompt and press Enter. The `help` command lists each supported command along with a brief description of each command.

**TIP** To view page-wise listing of the debugger shell commands, right-click in the *Debugger Shell* view and select `Paging` from the context menu. Alternatively, click the `Enable Paging` icon.

### 4.1 Executing Previously Issued Commands

The debugger shell maintains a history of previously executed commands. Instead of re-typing these commands, you can recall them from the history.

- To recall a command from the history, press the Up arrow key.

Each press of the Up arrow key shows the preceding issued command. Each press of the Down arrow key shows the succeeding issued command.
4.2 Using Code Hints

You can have the debugger shell complete the name of a command as you enter it on the command-line. As you continue typing characters, the debugger shell refines the list of possible commands. For example, you can use this technique as a shortcut to entering help to see a full list of commands.

To use code hints in the debugger shell:

1. Open the Debugger Shell view.
2. Type Ctrl + Space.
   Code hints appear. As you enter additional characters, the debugger shell refines the commands that appear in the code hints. Use the arrow keys or the mouse to scroll through the command names that appear in the list. The debugger shell shows additional information for the highlighted command name.
3. Highlight the name of the command that you want to have the debugger shell complete for you.
4. Press the Enter key, or double-click the name of the command.
   The remaining characters of the command name appear in the debugger shell.

**NOTE** Press the Esc key to exit code hints and return to the debugger shell.

4.3 Using Auto-Completion

The Debugger Shell supports auto-completion of these items:

- Debugger Shell Commands
- Arguments (such as file path)
- TCL commands (both built-in and those created with proc)

To use auto-completion:

1. Open the Debugger Shell view.
2. Type the initial characters of an item that the debugger shell can auto-complete.
3. Press the Tab key.
   The remaining characters of the item appear in the debugger shell.

**TIP** If you enter the abbreviated form of an IDE command name, press the spacebar instead of the Tab key to have the IDE auto-complete the command name
4.4 Command-Line Debugger Shell

Use the debugger shell to execute commands in a command-line environment. The command-line debugger engine executes the commands that you enter in the debugger shell, then displays the results. For example, the `launch`, `debug`, and `run` commands let you list or run launch configurations from the command line.

4.4.1 Setting Hardware Breakpoints

Use the debugger shell to set hardware breakpoints that control program execution. You use the `bp` command to set the hardware breakpoints.

To use the debugger shell to set a hardware breakpoint:

1. Open the debugger shell.
2. Begin the command line with the text: `bp -hw`
3. Complete the command line by specifying the function, address, or file at which you want to set the hardware breakpoint.
4. Press the Enter key.
   
   The debugger shell executes the command and sets the hardware breakpoint.

**TIP** Enter `help bp` at the command-line prompt to see examples of the `bp` command syntax and usage.

4.5 Debugger Shell Commands

This topic lists and defines each command-line debugger command.

---

**about**

Lists the version information.

**Syntax**

```
about
```
alias

Creates an alias for a debug command, removes such an alias, or lists all current aliases.

Syntax

```
alias [<alias> [command]]
```

Parameters

**alias**
- Lists current aliases.

Examples

The following table lists and defines examples of the `alias` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Lists current aliases.</td>
</tr>
<tr>
<td>alias ls</td>
<td>Remove the alias <code>ls</code>.</td>
</tr>
<tr>
<td>alias ls dir</td>
<td>Issue the <code>dir</code> command when <code>ls</code> is typed.</td>
</tr>
</tbody>
</table>

bp

Sets a breakpoint, removes a breakpoint, or lists the current breakpoints.

Syntax

```
bp [-{hw|sw|auto}] {<func>[:<ms>]<addr>|<file> <line> [<column>]}  
bp [-{hw|sw|auto}] {<file> <line> [<function>] [column]} 
bp all|#<id> [#<id.instance>]<func>|<addr> off|enable|disable|ignore
```
Debugger Shell

Debugger Shell Commands

```plaintext
<count>}
bp #<id> cond <c-expr>
```

Examples

The following table lists and defines examples of the `bp` command.

**Table 4.2 bp Command-Line Debugger Command — Examples**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bp</code></td>
<td>Lists all breakpoints.</td>
</tr>
<tr>
<td><code>bp -hw fn</code></td>
<td>Set hardware breakpoint at function fn().</td>
</tr>
<tr>
<td><code>bp -auto file.cpp 101 1</code></td>
<td>Set an auto breakpoint on file file.cpp at line 101, column 1.</td>
</tr>
<tr>
<td><code>bp -auto file.cpp 101 &quot;int foo&lt;int&gt;()&quot; 1</code></td>
<td>Set an auto breakpoint on file file.cpp at line 101 on function template instance &quot;int foo&lt;int&gt;()&quot;, column 1.</td>
</tr>
<tr>
<td><code>bp fn off</code></td>
<td>Remove the breakpoint at function fn().</td>
</tr>
<tr>
<td><code>bp 10343</code></td>
<td>Set a breakpoint at memory address 10343.</td>
</tr>
<tr>
<td><code>bp #4 off</code></td>
<td>Remove the breakpoint number 4.</td>
</tr>
<tr>
<td><code>bp #4 disable</code></td>
<td>Disable the breakpoint number 4.</td>
</tr>
<tr>
<td><code>bp #4 ignore 3</code></td>
<td>Set ignore count to 3 for breakpoint number 4.</td>
</tr>
<tr>
<td><code>bp #4 cond x == 3</code></td>
<td>Set the condition for breakpoint number 4 to fire only if x == 3.</td>
</tr>
<tr>
<td><code>bp #4.1 off</code></td>
<td>Remove the breakpoint instance number 4.1.</td>
</tr>
<tr>
<td><code>bp #4.1 ignore 3</code></td>
<td>Set ignore count to 3 for breakpoint instance number 4.1.</td>
</tr>
<tr>
<td><code>bp #4.1 cond x == 3</code></td>
<td>Set the condition for breakpoint instance number 4.1 to fire only if x == 3.</td>
</tr>
</tbody>
</table>
cd
Changes to a different directory or lists the current directory. Pressing the Tab key completes the directory name automatically.

Syntax
```
cd [path]
```

Parameter
path
- Directory pathname; accepts asterisks and wildcards.

Examples
The following table lists and defines examples of the `cd` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>Display current directory.</td>
</tr>
<tr>
<td>cd c:</td>
<td>Changes to the C: drive root directory.</td>
</tr>
<tr>
<td>cd d:/mw/0622/test</td>
<td>Changes to the specified D: drive directory</td>
</tr>
<tr>
<td>cd c:p*s</td>
<td>Changes to any C: drive directory whose name starts with <code>p</code> and ends with <code>s</code>.</td>
</tr>
</tbody>
</table>

change
Changes the contents of register, memory location, block of registers, or memory locations.

Syntax
```
change <addr-spec> [range] [-s|-ns] [%<conv>] <value>
  change <addr-spec>{..<addr|#<n> [range]} [-s|-ns] [%<conv>]
```

---

*CodeWarrior Common Features Guide* 251
Debugger Shell

Debugger Shell Commands

change <reg-spec> [n] [-s|-ns] [%<conv>] <value>
change <reg-spec>{..<reg>|#<n>} [-s|-ns] [%<conv>] <value>
change <var-spec> [-s|-ns] [%<conv>] <value>
change v <var> [-s|-ns] [%<conv>] <value>

Parameter

The following table lists and defines parameters of the change command.

Table 4.4 change Command-Line Debugger Command — Parameters

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ms&gt;</td>
<td>On architectures supporting multiple memory spaces, specifies the memory space in which &lt;addr&gt; is to be found. Refer the help for the option -ms of display or mem for more information on memory spaces. If unspecified, the setting &quot;config MemIdentifier&quot; is used.</td>
</tr>
<tr>
<td>&lt;addr&gt;</td>
<td>Target address in hex format.</td>
</tr>
<tr>
<td>&lt;count&gt;</td>
<td>Number of memory cells.</td>
</tr>
<tr>
<td>x&lt;cell-size&gt;</td>
<td>Memory is displayed in units called cells, where each cell consists of &lt;cell-size&gt; bytes. If unspecified, the setting &quot;config MemWidth&quot; is used.</td>
</tr>
<tr>
<td>h&lt;access-size&gt;</td>
<td>Memory is accessed with a hardware access size of &lt;access-size&gt; bytes. If unspecified, the setting &quot;config MemAccess&quot; is used.</td>
</tr>
<tr>
<td>{8,16,32,64}bit</td>
<td>Sets both &lt;cell-size&gt; and &lt;access-size&gt;.</td>
</tr>
<tr>
<td>&lt;a1&gt;{..&lt;a2&gt;</td>
<td>#&lt;n&gt;}</td>
</tr>
</tbody>
</table>
### Debugger Shell

### Debugger Shell Commands

#### Table 4.4 change Command-Line Debugger Command — Parameters

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{r</td>
<td>nr}</td>
</tr>
<tr>
<td>&lt;reg&gt;</td>
<td>A register name or a register group name.</td>
</tr>
<tr>
<td>..&lt;reg&gt;</td>
<td>The end point for a range of registers to access.</td>
</tr>
<tr>
<td>&lt;n&gt;</td>
<td>Number of registers.</td>
</tr>
<tr>
<td>all</td>
<td>Specifies all registers.</td>
</tr>
<tr>
<td>v:</td>
<td>If this option appears with no &lt;var&gt; following it, then all variables pertinent to the current scope are printed.</td>
</tr>
<tr>
<td>&lt;var&gt;</td>
<td>Symbolic name of the variable to print. Can be a C expression as well.</td>
</tr>
<tr>
<td>v</td>
<td>This alternate syntax is provided mainly for backward compatibility.</td>
</tr>
<tr>
<td>-s</td>
<td>-ns</td>
</tr>
<tr>
<td>%-&lt;conv&gt;</td>
<td>Specifies the type of the data. Possible values for &lt;conv&gt; are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal.</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal.</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal.</td>
</tr>
</tbody>
</table>
Examples

The examples assume the following settings:

- radix x
- config MemIdentifier 0
- config MemWidth 32
- config MemAccess 32
- config MemSwap off

The following table lists and defines Memory examples of the change command.

Table 4.5 change Command-Line Debugger Command — Memory Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>change 10000 10</td>
<td>Change memory range 0x10000-3 to 0x10 (because radix is hex).</td>
</tr>
<tr>
<td>change 1:10000 20</td>
<td>Change memory range 0x10000-3, memory space 1, to 0x20.</td>
</tr>
<tr>
<td>change 10000 16 20</td>
<td>Change each of 16 cells in the memory range 0x10000-3f to 0x20.</td>
</tr>
<tr>
<td>change 10000 16x1h8 31</td>
<td>Change each of 16, 1-byte cells to 0x31, using a hardware access size of 8-bytes per write.</td>
</tr>
<tr>
<td>change 10000 -s %d 200</td>
<td>Change memory range 0x10000-3 to c8000000.</td>
</tr>
</tbody>
</table>

The following table lists and defines Register examples of the change command.
The following table lists and defines Variable examples of the change command.

**Table 4.7 change Command-Line Debugger Command — Variable Examples**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>change myVar 10</td>
<td>Change the value of variable myVar to 16 (0x10)</td>
</tr>
</tbody>
</table>

**cls**

Clears the command line debugger window.

**Syntax**

`cls`

**cmdwin::ca**

Manages global cache operations. That is, they affect the operation of the entire cache. For multi-core processors, these commands operate on a specific cache if an optional ID number is provided. If the ID
number is absent, the command operates on the cache that was assigned as the default by the last 
cmdwin::ca::default command. The following table summarizes cache commands.

### Table 4.8 Global Cache Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdwin::ca::default</td>
<td>Set specified cache as default</td>
</tr>
<tr>
<td>cmdwin::ca::enable</td>
<td>Enable/disable cache</td>
</tr>
<tr>
<td>cmdwin::ca::flush</td>
<td>Flushes cache</td>
</tr>
<tr>
<td>cmdwin::ca:: inval</td>
<td>Invalidates cache</td>
</tr>
<tr>
<td>cmdwin::ca::lock</td>
<td>Lock/Unlock cache</td>
</tr>
<tr>
<td>cmdwin::ca::show</td>
<td>Show the architecture of the cache</td>
</tr>
</tbody>
</table>

### Syntax

command [<cache ID>] [on | off]

### Parameters

- `<cache ID>`
  - Selects the cache that the command affects.
- [on | off]
  - Changes a cache state.

---

**cmdwin::caln**

The cmdwin::caln commands manage cache line operations. They affect memory elements within a designated cache. The following table summarizes these commands.

### Table 4.9 Cache Line Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdwin::caln::get</td>
<td>Display cache line</td>
</tr>
<tr>
<td>cmdwin::caln::flush</td>
<td>Flush cache line</td>
</tr>
<tr>
<td>cmdwin::caln::inval</td>
<td>Invalidate cache line</td>
</tr>
</tbody>
</table>
Syntax
command [<cache ID>] <line> [<count>]

Parameters
<cache ID>
   Optional. Specifies the cache that the command affects, otherwise it affects the default cache.

<line>
   Specifies the cache line to affect.

<count>
   Optional. Specifies the number of cache lines the command affect.

Examples
The following table lists and defines examples of the cmdwin::caln commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdwin::caln::lock</td>
<td>Locks/unlocks cache line</td>
</tr>
<tr>
<td>cmdwin::caln::set</td>
<td>Writes specified data to cache line</td>
</tr>
</tbody>
</table>

Table 4.9  Cache Line Commands

Table 4.10  copy Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdwin::caln::get 2</td>
<td>Displays the second cache line.</td>
</tr>
<tr>
<td>cmdwin::caln::flush 2</td>
<td>flushes line 2 of the default cache.</td>
</tr>
<tr>
<td>cmdwin::caln::set 2 = 0 1 1 2 3 5 8 13</td>
<td>Sets the contents of cache line two, where the first word has a value of 0, the second word has a value of 1, the third word has a value of 1, the fourth word has a value of 2, and so on.</td>
</tr>
</tbody>
</table>

config
Configures the command window.
Debugger Shell

Debugger Shell Commands

Syntax

config <option> [<sub-option>] <value> [-np]
config

Options

<none>

With no options, config displays the current configuration settings.

- np

Do not print anything to the display, only return the data.

The table below lists and defines Display options of the config command.

Table 4.11 config Command-Line Debugger Command — Display Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>echoCmd on</td>
<td>off</td>
</tr>
<tr>
<td>h/exPrefix &lt;prefix&gt;</td>
<td>Sets the string to be used as the prefix for hex values.</td>
</tr>
<tr>
<td>binPrefix &lt;prefix&gt; 3</td>
<td>Sets the string to be used as the prefix for binary values.</td>
</tr>
<tr>
<td>showCommas off</td>
<td>on</td>
</tr>
<tr>
<td>hexPadding on</td>
<td>off</td>
</tr>
<tr>
<td>decPadding on</td>
<td>off</td>
</tr>
<tr>
<td>mem/Identifier &lt;mem-space-id&gt;</td>
<td>Sets the string to be used for the main memory space prefix.</td>
</tr>
<tr>
<td>memCache off</td>
<td>on</td>
</tr>
</tbody>
</table>
### Table 4.11  config Command-Line Debugger Command — Display Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memReadMax &lt;max-bytes&gt;</td>
<td>Limits the amount of memory to be read in a single command. This prevents the Command Window from locking up on abnormally large memory read requests.</td>
</tr>
<tr>
<td>memSwap off</td>
<td>on</td>
</tr>
<tr>
<td>memWidth &lt;bits&gt;</td>
<td>factory</td>
</tr>
<tr>
<td>memAccess &lt;bits&gt;</td>
<td>Specifies the default hardware access size for target memory. A setting of 0 allows the hardware access size to match the display width of the command.</td>
</tr>
<tr>
<td>memFixedIntBits &lt;bits&gt;</td>
<td>For fixed point formatting, sets the range to the specified number of bits. For example, a value of 8 will set the range to [-2^8,2^8), or [-256,256).</td>
</tr>
</tbody>
</table>
The following table lists and defines Run Control options of the `config` command.

### Table 4.11 `config` Command-Line Debugger Command — Display Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`variable &lt;sub-option&gt; [on</td>
<td>Enables or disables certain fields in the output of the &quot;evaluate&quot; off]<code>command. If neither on nor off are specified, then the field is enabled. Possible values for</code>&lt;sub-option&gt;` are:</td>
</tr>
<tr>
<td></td>
<td>• echo - the variable name</td>
</tr>
<tr>
<td></td>
<td>• location - the address of the variable</td>
</tr>
<tr>
<td></td>
<td>• size - the size of the variable is bytes</td>
</tr>
<tr>
<td></td>
<td>• type - the variable type</td>
</tr>
<tr>
<td><code>variable format &lt;format&gt;</code></td>
<td>Controls the output format of the &quot;evaluate&quot; command. Possible values for <code>&lt;format&gt;</code> are:</td>
</tr>
<tr>
<td></td>
<td>• -</td>
</tr>
<tr>
<td></td>
<td>• d</td>
</tr>
<tr>
<td></td>
<td>• u</td>
</tr>
<tr>
<td></td>
<td>• x</td>
</tr>
<tr>
<td></td>
<td>• c</td>
</tr>
<tr>
<td></td>
<td>• s</td>
</tr>
<tr>
<td></td>
<td>• p</td>
</tr>
<tr>
<td></td>
<td>• f</td>
</tr>
<tr>
<td></td>
<td>• e</td>
</tr>
<tr>
<td></td>
<td>• i</td>
</tr>
<tr>
<td></td>
<td>• b</td>
</tr>
<tr>
<td></td>
<td>• o</td>
</tr>
<tr>
<td></td>
<td>• o</td>
</tr>
</tbody>
</table>
Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoThreadSwitch</td>
<td>Allows the user to control whether the Command Window will perform automatic thread-switching. Possible settings are always on, always off, and on when running interactively, i.e. not from a script. If enabled, automatic thread switching is done in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• If no thread is currently selected or if the current thread exits, then the first one detected will become the current.</td>
</tr>
<tr>
<td></td>
<td>• If the current thread is running and another thread stops, then the current thread will switch to the stopped thread.</td>
</tr>
<tr>
<td>debugTimeout</td>
<td>The maximum amount of time to wait for a debug command to finish. You can also hit ESC to stop waiting.</td>
</tr>
<tr>
<td>runControlSync</td>
<td>Sets how to synchronize run control commands. If set to &quot;on&quot;, then all run control commands will wait until a thread stopped event. If set to &quot;off&quot;, then all run control commands will return immediately. If set to &quot;script-only&quot;, then all run control commands will wait while running a script but will return immediately while running interactively.</td>
</tr>
<tr>
<td>setPCNextValidLine</td>
<td>Controls the behavior of the setpc command in the case that the specified file line number has no source code. If set to &quot;on&quot;, the PC is set to the next line number containing source. If set to &quot;off&quot;, an error is shown.</td>
</tr>
</tbody>
</table>

config

Display the current config status.

The following table lists and defines Display examples of the config command.
Debugger Shell
Debugger Shell Commands

Table 4.13  Conf Command-Line Debugger Command — Display Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| config echoCmd on | If "reg D1" is a command executed from a script, the output will have on top the command itself:  
  
 .cmdwin::reg D1  
  + General Purpose Registers  
  D1=$ffffffff |
| config hexPrefix 0x | Show hexadecimal numbers with "0x" prefix. |
| config ShowCommas on | Show hexadecimal and binary numbers with a colon, as in  
  $0000:0000, and show decimal numbers with a comma, as in  
  1,000,000.00. |
| config HexPadding off | Show hex and binary numbers with leading zeros, as in 0x0000.  
  config MemIdentifier 0. Use "0" as the default memory space for memory commands. |
| config MemCache off | Turn off caching of target memory. AFFECTS COMMAND WINDOW ONLY. |
| config MemReadMax 2048 | Limit memory commands to 2048 (decimal) bytes. |
| config MemSwap on | Swap memory on cell boundaries before accessing the target. |
| config MemWidth 16 | Displays and writes 16bit values. |
| config MemWidth factory | Reset the MemWidth to factory settings. |
| config MemAccess 8 | Uses an 8-bit access size for reading and writing target memory. |
| config MemFixedIntBits 8 | Sets the fixed point range to [-2^8,2^8), or [-256,256). |
| config var echo on | Include the variable name in the output of the "evaluate" command. |
The following table lists and defines Run Control examples of the `config` command.

### Table 4.14 Conf Command-Line Debugger Command — Run Control Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>config AutoThreadSwitch</code></td>
<td>If commands are being entered interactively, i.e. not from a script, automatic thread switching will be performed. If no thread is currently selected or if the current thread exits, then the first one detected will become the current. If the current thread is running and another thread stops, then the current thread will switch to the stopped thread.</td>
</tr>
<tr>
<td><code>config DebugTimeout 10</code></td>
<td>Wait up to 10 seconds for debug command to finish.</td>
</tr>
<tr>
<td><code>config RunControlSync on</code></td>
<td>Run control commands will wait for thread-stopped event.</td>
</tr>
<tr>
<td><code>config SetPCNextValid Line on</code></td>
<td>If setpc is called for a file line number with no source code, the line number is automatically increased to the next line with source code.</td>
</tr>
</tbody>
</table>
copy
Copies contents of a memory address or address block to another memory location.

Syntax
copy [\ms:]<addr>[..<addr>|\#<bytes>] [\ms:]<addr>

Parameter
<addr>
One of these memory-address specifications:

- A single address
- First address of the destination memory block.

Examples
The following table lists and defines examples of the copy command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy 00..1f 30</td>
<td>Copy memory addresses 00 through 1f to address 30.</td>
</tr>
<tr>
<td>copy 20#10 50</td>
<td>Copy 10 memory locations beginning at memory location 20 to memory beginning at location 50.</td>
</tr>
</tbody>
</table>

debug
Launches a debug session.

Syntax
debug [[-index] <index> | [-name] <debug-config-name>]

Examples
The following table lists and defines examples of the debug command.
**Debugger Shell**

**Debugger Shell Commands**

### Table 4.16 debug Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>Start debugging using the default launch configuration, which is the last debugged configuration if one exists and index 0 otherwise.</td>
</tr>
<tr>
<td>debug -index 3</td>
<td>Start debugging using the launch configuration at index 3. Type ‘launch’ for the current set of launch configurations.</td>
</tr>
<tr>
<td>debug -name 3</td>
<td>Start debugging using the launch configuration named ‘3’. Type ‘launch’ for the current set of launch configurations.</td>
</tr>
<tr>
<td>debug 3</td>
<td>Start debugging using the launch configuration named ‘3’. If ‘3’ does not exist then launch configuration with index 3 will be launched. Type ‘launch’ for the current set of launch configurations.</td>
</tr>
<tr>
<td>debug {My Launch Config}</td>
<td>Start debugging using the launch configuration named ‘My Launch Config’. Type ‘launch’ for the current set of launch configurations.</td>
</tr>
</tbody>
</table>

### dir

Lists directory contents.

**Syntax**

dir [path|files]

**Examples**

The following table lists and defines examples of the dir command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir</td>
<td>Lists all files of the current directory.</td>
</tr>
<tr>
<td>di *.txt</td>
<td>Lists all current-directory files that have the .txt file name extension.</td>
</tr>
</tbody>
</table>
Debugger Shell
Debugger Shell Commands

Table 4.17  dir Command-Line Debugger Command—Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir c:/tmp</td>
<td>Lists all files in the tmp directory on the C: drive.</td>
</tr>
<tr>
<td>dir /ad</td>
<td>Lists only the subdirectories of the current directory.</td>
</tr>
</tbody>
</table>

disassemble

Disassembles the instructions of the specified memory block.

Syntax

disassemble

disassemble pc|[<ms>:]<addr> [count]

reset

disassemble [ms:]<a1>{..<a2>|#<n>}

Parameter

[none]

With no options, the next block of instructions is listed. After a target stop event, the next block starts at the PC.

[ms:]<addr>

Target address in hex. On targets with multiple memory spaces, a memory space id can be specified.

pc

The current program counter.

<count>

Number of instructions to be listed.

reset

Reset the next block to the PC and the instruction count to one screen.

<a1>{..<a2>|#<n>}

Specifies a range of memory either by two endpoints, \(<a1>\) and \(<a2>\), or by a startpoint and a count, \(<a1>\) and \(<n>\).
Examples

The following table lists and defines examples of the `disassemble` command.

Table 4.18 disassemble Command-Line Debugger Command—Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disassemble</td>
<td>Lists the next block of instructions.</td>
</tr>
<tr>
<td>disassemble reset</td>
<td>Reset the next block to the PC and the instruction count to one screenful.</td>
</tr>
<tr>
<td>disassemble pc</td>
<td>Lists instructions starting at the PC.</td>
</tr>
<tr>
<td>disassemble pc 4</td>
<td>Lists 4 instructions starting at the PC. Sets the instruction count to 4.</td>
</tr>
<tr>
<td>disassemble 1000</td>
<td>Lists instructions starting at address <code>0x1000</code>.</td>
</tr>
<tr>
<td>disassemble p:1000 4</td>
<td>Lists 4 instructions from memory space p, address 1000. Sets the instruction count to 4.</td>
</tr>
</tbody>
</table>

**display**

Lists the contents of a register or memory location; lists all register sets of a target; adds register sets, registers, or memory locations; or removes register sets, registers, or memory locations.
Debugger Shell
Debugger Shell Commands

Syntax

display <addr-spec> [range] [-s|-ns] [%<conv>] [-np]
display -ms
display <addr-spec>{..<addr}|#<n>} [range] [-s|-ns] [%<conv>] [-np]
display <reg-spec> [<n}] [{d|nr|nv|np} ...] [-s|-ns] [%<conv>]}
display <reg-spec>{..<reg}|#<n>} [{d|nr|nv|np} ...] [-s|-ns] [%<conv>]}
display all|r:|nr: [{d|nr|nv|np} ...] [-s|-ns] [%<conv>]}
display [-]regset
display <var-spec> [-np] [-s|-ns] [%<conv>]}
display v: [-np] [-s|-ns] [%<conv>]

Options

The following table lists and defines parameters of the display command.

Table 4.19 display Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ms&gt;</td>
<td>On architectures supporting multiple memory spaces, specifies the memory space in which &lt;addr&gt; is to be found. Refer the help for the option -ms of display or mem for more information on memory spaces. If unspecified, the setting &quot;config MemIdentifier&quot; is used.</td>
</tr>
<tr>
<td>&lt;addr&gt;</td>
<td>Target address in hex format.</td>
</tr>
<tr>
<td>&lt;count&gt;</td>
<td>Number of memory cells.</td>
</tr>
<tr>
<td>x&lt;cell-size&gt;</td>
<td>Memory is displayed in units called cells, where each cell consists of &lt;cell-size&gt; bytes. If unspecified, the setting &quot;config MemWidth&quot; is used.</td>
</tr>
<tr>
<td>h&lt;access-size&gt;</td>
<td>Memory is accessed with a hardware access size of &lt;access-size&gt; bytes. If unspecified, the setting &quot;config MemAccess&quot; is used.</td>
</tr>
<tr>
<td>{8,16,32,64}bit</td>
<td>Sets both &lt;cell-size&gt; and &lt;access-size&gt;.</td>
</tr>
<tr>
<td>-np</td>
<td>Don’t print anything to the display, only return the data.</td>
</tr>
</tbody>
</table>
### Debugger Shell Commands

#### Table 4.19 display Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ms</td>
<td>On architectures supporting multiple memory spaces, displays the list of available memory spaces including a mnemonic and/or an integer index which may be used when specifying a target address.</td>
</tr>
<tr>
<td>`&lt;a1&gt;[..&lt;a2&gt;</td>
<td>#&lt;n&gt;]`</td>
</tr>
<tr>
<td>`{r</td>
<td>nr}`</td>
</tr>
<tr>
<td><code>&lt;reg&gt;</code></td>
<td>A register name or a register group name.</td>
</tr>
<tr>
<td><code>..&lt;reg&gt;</code></td>
<td>The end point for a range of registers to access.</td>
</tr>
<tr>
<td><code>&lt;n&gt;</code></td>
<td>Number of registers.</td>
</tr>
<tr>
<td>all</td>
<td>Specifies all registers.</td>
</tr>
<tr>
<td>-d</td>
<td>Print detailed data book information.</td>
</tr>
<tr>
<td>-nr</td>
<td>Print only register groups, i.e. no registers.</td>
</tr>
<tr>
<td>-nv</td>
<td>Print only register groups and register names, i.e. no values.</td>
</tr>
<tr>
<td>-np</td>
<td>Don't print anything to the display, only return the data.</td>
</tr>
<tr>
<td>regset</td>
<td>Display the register group hierarchy.</td>
</tr>
<tr>
<td>v:</td>
<td>If this option appears with no <code>&lt;var&gt;</code> following it, then all variables pertinent to the current scope are printed.</td>
</tr>
</tbody>
</table>
Debugger Shell
Debugger Shell Commands

Table 4.19 display Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;var&gt;</td>
<td>Symbolic name of the variable to print. Can be a C expression as well.</td>
</tr>
<tr>
<td>-s</td>
<td>-ns</td>
</tr>
<tr>
<td>%&lt;conv&gt;</td>
<td>Specifies the type of the data. Possible values for &lt;conv&gt; are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal.</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal.</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal.</td>
</tr>
<tr>
<td>%f</td>
<td>Floating point.</td>
</tr>
<tr>
<td>%[Q&lt;n&gt;]F</td>
<td>Fixed or Fractional. The range of a fixed point value depends on the (fixed) location of the decimal point. The default location is set by the config command option &quot;MemFixedIntBits&quot;.</td>
</tr>
<tr>
<td>%s</td>
<td>ASCII.</td>
</tr>
</tbody>
</table>

Examples

The examples assume the following settings:
- radix x
- config MemIdentifier 0
- config MemWidth 32
- config MemAccess 32
- config MemSwap off

The following table lists and defines examples of the display command.
### Table 4.20 display Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display 10000</td>
<td>Display memory range 0x10000-3 as one cell.</td>
</tr>
<tr>
<td>display 1:10000</td>
<td>Display memory range 0x10000-3, memory space 1, as one cell.</td>
</tr>
<tr>
<td>display 10000 16</td>
<td>Display memory range 0x10000-3f as 16 cells.</td>
</tr>
<tr>
<td>display 10000 16x1h8</td>
<td>Display 16, 1-byte cells, with a hardware access size of 8-bytes per read.</td>
</tr>
<tr>
<td>display 10000 8bit</td>
<td>Display one byte, with a hardware access size of one byte.</td>
</tr>
<tr>
<td>display 10000 -np</td>
<td>Return one cell, but don't print it to the Command Window.</td>
</tr>
<tr>
<td>display 10000 -s</td>
<td>Display one cell with the data endian-swapped.</td>
</tr>
<tr>
<td>display 10000 %d</td>
<td>Display one cell in decimal format.</td>
</tr>
<tr>
<td>display -ms</td>
<td>Display the available memory spaces, if any.</td>
</tr>
<tr>
<td>display -regset</td>
<td>List all the available register sets on the target chip.</td>
</tr>
<tr>
<td>display R1</td>
<td>Display the value of register R1.</td>
</tr>
<tr>
<td>display &quot;General Purpose Registers/R1&quot;</td>
<td>Display the value of register R1 in the General Purpose Register group.</td>
</tr>
<tr>
<td>display R1 -d</td>
<td>Display detailed &quot;data book&quot; contents of R1, including bitfields and definitions.</td>
</tr>
<tr>
<td>display &quot;nr:General Purpose Registers/R1&quot; 25</td>
<td>Beginning with register R1, display the next 25 registers. Register groups are not recursively searched.</td>
</tr>
<tr>
<td>display myVar-s %d</td>
<td>Display the endian-swapped contents of variable myVar in decimal.</td>
</tr>
</tbody>
</table>
Debugger Shell
Debugger Shell Commands

**evaluate**

Display variable or expression.

**Syntax**

```
evaluate [#<format>] [-l] [<var|expr>]
```

**Parameter**

- `<format>`
  
  Output format and possible values:
  
  `-`, #Default
  
  `d`, #Signed
  
  `u`, #Unsigned
  
  `h`, #x, #Hex
  
  `c`, #Char
  
  `s`, #CString
  
  `p`, #PascalString
  
  `f`, #Float
  
  `e`, #Enum
  
  `i`, #Fixed
  
  `o`, #w, #Unicode
  
  `b`, #Binary
  
  `<none>`, #Fract
  
  `<none>`, #Boolean
  
  `<none>`, #SignedFixed

**Examples**

The following table lists and defines examples of the `evaluate` command.
finish

Execute until the current function returns.

Syntax

finish
Debugger Shell
Debugger Shell Commands

fl::blankcheck
Test that the flash device is in the blank state.

Syntax
blankcheck all | list

Parameters
all
  Check that all sectors are in the blank state.
list
  Check that specific sectors are in the blank state. The sector list is set with the "device" command.

Examples
The following table lists and defines examples of the fl::blankcheck command.

Table 4.22 fl::blankcheck Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blankcheck all</td>
<td>Test if the flash device is in the blank state. All sectors will be tested regardless of the enabled list maintained by the &quot;device&quot; command.</td>
</tr>
<tr>
<td>blankcheck list</td>
<td>Test whether the sectors in the enabled list are in the blank state.</td>
</tr>
</tbody>
</table>

fl::checksum
Calculate a checksum.

Syntax
checksum [-host | -range <addr> <size> | -dev]

Options
The following table lists and defines options of the fl::checksum command.
Debugger Shell
Debugger Shell Commands

**Example**

```markdown
checksum

Calculate a checksum.
```

### `fl::device`

Define the flash device.

**Syntax**

```markdown
device
device <setting> ...
device ls
device ls org [<dev>]
device ls sect [[<dev>] <org>]
```

**Options**

The following table lists and defines options of the `fl::device` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>When no options are specified, calculate the checksum for the target memory contents corresponding to the settings of the <code>fl::image</code> command. The target is defined by the <code>fl::target</code> command.</td>
</tr>
<tr>
<td>-host</td>
<td>Calculate the checksum for the host image file contents corresponding to the settings of the <code>fl::image</code> command.</td>
</tr>
<tr>
<td>-range</td>
<td><code>&lt;addr&gt;</code> <code>&lt;size&gt;</code> Calculate the checksum for the target memory contents specified by a beginning address <code>&lt;addr&gt;</code> and number of bytes <code>&lt;size&gt;</code>, both given in hex. The target is defined by the <code>fl::target</code> command.</td>
</tr>
<tr>
<td>-dev</td>
<td>Calculate the checksum for the entire flash contents. The flash is defined by the <code>fl::device</code> command. The target is defined by the <code>fl::target</code> command.</td>
</tr>
</tbody>
</table>
Debugger Shell

Debugger Shell Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>With no options, lists the current settings.</td>
</tr>
<tr>
<td>&lt;setting&gt;</td>
<td>Used to set a configuration setting. Possible values are:</td>
</tr>
<tr>
<td>-d &lt;dev&gt;, -o &lt;org&gt;, -a &lt;addr&gt; [&lt;end]&gt;</td>
<td>-se all</td>
</tr>
<tr>
<td>-d &lt;dev&gt;</td>
<td>Set the device to &lt;dev&gt;.</td>
</tr>
<tr>
<td>-o &lt;org&gt;</td>
<td>Set the organization to &lt;org&gt;.</td>
</tr>
<tr>
<td>-a &lt;addr&gt; [&lt;end&gt;]</td>
<td>Set the start &lt;addr&gt; and optional end &lt;end&gt; address for the device, both given in hex.</td>
</tr>
<tr>
<td>-se all</td>
<td>&lt;index&gt; ...</td>
</tr>
<tr>
<td>-sd all</td>
<td>&lt;index&gt; ...</td>
</tr>
<tr>
<td>ls</td>
<td>Lists all the supported devices.</td>
</tr>
<tr>
<td>ls org [&lt;dev&gt;]</td>
<td>List the organizations for a particular device. The device may be specified with &lt;dev&gt;, otherwise the current device is used.</td>
</tr>
<tr>
<td>ls sect [[&lt;dev&gt;] &lt;org&gt;]]</td>
<td>List the sectors for a particular device and organization. The organization may be specified with &lt;org&gt;, otherwise the current device and organization is used. If &lt;org&gt; is specified, the device may be specified with &lt;dev&gt;, otherwise the current device is used. If &lt;dev&gt; is specified, then 0 is used for the starting address; otherwise, the current device start address is used.</td>
</tr>
</tbody>
</table>

Examples

device

Lists the current settings.

fl::diagnose

Dump flash information.
Syntax

diagnose [full]

Options

full

Also dump sector status (programmed/erased). This could take a few minutes for large flashes.

Examples

The following table lists and defines examples of the `fl::diagnose` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diagnose</td>
<td>Dump flash information like ID, sector map, sector factory protect status. <code>fl::device</code> command needs to be called prior to this command in order to set the device.</td>
</tr>
</tbody>
</table>

---

`fl::disconnect`

Close the connection to the target.

Syntax

disconnect

Examples

disconnect

Close the connection to the target. The first flash command that needs to access the target opens a connection to the target that remains open for further flash operations.

---

`fl::dump`

Dumps the content of entire flash to the specified file.

Syntax

`fl::dump [all | -range start_addr end_addr] -o <file>`
Debugger Shell

Debugger Shell Commands

**Parameter**

- **-all**
  Dumps content of entire flash to the specified file.

- **-range** `<start_addr> <end_addr>`
  Sets the range of flash region to be dumped.

- **-t** `<type>`
  Sets the type of flash region to be dumped.

- **-o** `<file>`
  Dumps the flash to the specified file. This is mandatory.

**Examples**

dump -all -t "Binary/Raw Format" -o "myfile"

Dumps all flash or flash region from `<start_addr>` to `<end_addr>` to the file specified with `-o` argument.

---

**fl::erase**

Erase the flash device.

**Syntax**

erase all | list | image

**Parameters**

- **all**
  Erase all sectors using an all-chip erase function.

- **list**
  Erase specific sectors as set with the "device" command.

- **image**
  Erase all sectors occupied by the file specified with `fl::image`.

**Examples**

The following table lists and defines examples of the `fl::erase` command.
Define the flash image settings.

**Syntax**

```shell
image
image <setting> ...
image ls
```

**Options**

The following table lists and defines options of the `fl::image` command.

---

### Table 4.26 fl::image Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>erase all</td>
<td>Erase the device using the all-chip erase operation. This is not supported by all flash devices. All sectors will be erased regardless of the enabled list maintained by the &quot;device&quot; command.</td>
</tr>
<tr>
<td></td>
<td>Erase the device one sector at a time. All sectors will be erased regardless of the enabled list maintained by the &quot;device&quot; command.</td>
</tr>
<tr>
<td>erase list</td>
<td>Erase the sectors in enabled list.</td>
</tr>
<tr>
<td>erase image</td>
<td>Erase the sectors occupied by the file defined with <code>fl::image</code>.</td>
</tr>
</tbody>
</table>

---

### Table 4.27 fl::erase Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>With no options, lists the current settings.</td>
</tr>
<tr>
<td>&lt;setting&gt;</td>
<td>Used to set a configuration setting. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>-f &lt;file&gt;, -t &lt;type&gt;, -re on</td>
</tr>
<tr>
<td>-f &lt;file&gt;</td>
<td>Set the image file.</td>
</tr>
</tbody>
</table>
Debugger Shell
Debugger Shell Commands

Table 4.27 fl::erase Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t &lt;type&gt;</td>
<td>Set the type of the image file. Possible values are shown by &quot;image ls&quot;.</td>
</tr>
<tr>
<td>-re on/off</td>
<td>If -re is set to on, the range settings of this command will be used to restrict all flash commands to a particular address range. Otherwise no restriction is made.</td>
</tr>
<tr>
<td>-r &lt;addr&gt; [&lt;end&gt;]</td>
<td>Set the start &lt;addr&gt; and optional end &lt;end&gt; address for the restricting flash access, both given in hex. The range must also be enabled by the option &quot;-re&quot;.</td>
</tr>
<tr>
<td>-oe all</td>
<td>If -oe is set to on, the offset setting of this command will be used.</td>
</tr>
<tr>
<td>-e</td>
<td>If -oe is set to on, then the value of this setting is added to all addresses in the image file. The value is given in hex.</td>
</tr>
</tbody>
</table>

Examples

image

List the current settings.

fl::protect

Protects the sectors.

Syntax

fl::protect [on | off]

Parameter

[on | off]

Enable or disable protection of sectors.

fl::secure

Secure/unsecure the device.
Syntax
fl::secure [on | off] [password <pass>]

Parameter
[on | off]
Secure or unsecure a device.
password <pass>
Password used to secure the device.

fl::target
Define the target configuration settings.

Syntax
target
target <setting> ...
target ls [p|c]

Options
The following table lists and defines options of the fl::target command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>With no options, lists the current settings.</td>
</tr>
<tr>
<td>&lt;setting&gt;</td>
<td>Used to set a configuration setting. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>-c &lt;conn&gt;, -p &lt;proc&gt;, -ie on</td>
</tr>
<tr>
<td></td>
<td>&lt;initfile&gt;, -b &lt;addr&gt; [&lt;size&gt;] , -v on</td>
</tr>
<tr>
<td>-lc &lt;launch configuration name&gt;</td>
<td>Set the launch configuration that will be used.</td>
</tr>
<tr>
<td>-c &lt;conn&gt;</td>
<td>Set the connection to &lt;conn&gt;.</td>
</tr>
<tr>
<td>-p &lt;proc&gt;</td>
<td>Set the processor to &lt;proc&gt;.</td>
</tr>
</tbody>
</table>
Debugger Shell

Debugger Shell Commands

Table 4.28 fl::target Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ie on</td>
<td>off</td>
</tr>
<tr>
<td>-i &lt;initfile&gt;</td>
<td>Set the target initialization file to &lt;initfile&gt;. Only used if -ie is on.</td>
</tr>
<tr>
<td>-b &lt;addr&gt; [&lt;size&gt;]</td>
<td>Set the target RAM buffer for downloading image data to begin at &lt;addr&gt; with &lt;size&gt; bytes, both given in hex.</td>
</tr>
<tr>
<td>-v on</td>
<td>off</td>
</tr>
<tr>
<td>-l on</td>
<td>off</td>
</tr>
<tr>
<td>ls [p</td>
<td>c]</td>
</tr>
</tbody>
</table>

Examples

target
List the current settings.

fl::verify
Verify the flash device.

Syntax
verify

fl::write
Write the flash device.

Syntax
write
funcs
Displays information about functions.

**Syntax**
```bash
funcs [-all] <file> <line>
```

**Parameter**

- `-all`
  - Displays information about the functions using all debug contexts.

- `<file>`
  - Specifies the file name.

- `<line>`
  - Specifies the line number.

getpid
List the ID of the process being debugged.

**Syntax**
```bash
getpid
```

go
Starts to debug your program from the current instruction.

**Syntax**
```bash
go [nowait | <timeout_s>]
```

**Parameter**

- `<none>`
Debugger Shell

Debugger Shell Commands

Run the default thread. The command may wait for a thread break event before returning, depending on the settings `config runControlSync` and `config autoThreadSwitch`.

`nowait`

Return immediately without waiting for a thread break event.

`<timeout_s>`

Maximum number of seconds to wait for a thread break event. Can be set to `nowait`.

Examples

The following table lists and defines examples of the `go` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>go</code></td>
<td>Run the default thread.</td>
</tr>
<tr>
<td><code>go nowait</code></td>
<td>Run the default thread without waiting for a thread break event.</td>
</tr>
<tr>
<td><code>go 5</code></td>
<td>Run the default thread. If config runControlSync is enabled, then the command will wait for a thread break event for a maximum of 5 seconds.</td>
</tr>
</tbody>
</table>

help

Lists debug command help in the command-line debugger window.

Syntax

`help [-sort | -tree | <cmd>]`

Parameter

`command`

Name or short-cut name of a command.

Examples

The following table lists and defines examples of the `help` command.
### history

Lists the history of the commands entered during the current debug session.

**Syntax**

`history`

### jtagclock

Read or update the current JTAG clock speed.

**Syntax**

`jtagclock`

`jtagclock <chain-position> [<speed-in-kHz>]`

**Examples**

The following table lists and defines examples of the `jtagclock` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jtagclock 3</code></td>
<td>Read the current JTAG clock speed for chain position 3.</td>
</tr>
<tr>
<td><code>jtagclock 3 1000</code></td>
<td>Update the JTAG clock speed to 1000kHz for chain position 3.</td>
</tr>
</tbody>
</table>
Debugger Shell

Debugger Shell Commands

kill

Close the specified debug session.

Syntax

\texttt{kill [all | \textless index\textgreater{} ...]}

Examples

The following table lists and defines examples of the kill command.

Table 4.32  kill Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kill</td>
<td>Kills the debug session for the current process.</td>
</tr>
<tr>
<td>kill all</td>
<td>Kills all active debug sessions.</td>
</tr>
<tr>
<td>kill 0 1</td>
<td>Kills debug sessions 0 and 1.</td>
</tr>
</tbody>
</table>

launch

Lists the launch configurations.

Syntax

\texttt{launch}

Examples

\texttt{launch}

List the launch configurations. The last debugged configuration is denoted with an asterisk '*', last run with a greater than '>'.

linux::displaylinuxlist

Lists the expression for each element of a Linux list.
**Syntax**

```
displaylinuxlist -list <listName> -function
   <functionWhereListIsVisible> -address <listAddress> -type
   <elementTypeName> [-next <nextPath>]
```

**Options**

The following table lists and defines options of the `linux::displaylinuxlist` command.

**Table 4.33 linux::displaylinuxlist Command-Line Debugger Command — Options**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l[ist] &lt;listName&gt;</td>
<td>The name of the list (must be global).</td>
</tr>
</tbody>
</table>
| -f[unction]
   <functionWhereListIsVisible> | Some function where the list is visible, optional.                       |
| -a[ddress]
   <listAddress> | The address of the list, in hexa, only in case when
   (listName, functionWhereListIsVisible) are not specified, to be
   used with local lists.                                               |
| -t[type]
   <elementTypeName> | The type of the list elements.                                             |
| -n[next]
   <nextPath> | Specify in order all the structure member names needed to reach
   the next element.                                                   |

**Examples**

The following table lists and defines examples of the `linux::displaylinuxlist` command.
**Debugger Shell**

*Debugger Shell Commands*

---

### Table 4.34  `linux::displaylinuxlist` Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>linux::displaylinuxlist -list workqueues -function __create_workqueue -type workqueue_struct -next list next</code></td>
<td>Lists the current workqueues.</td>
</tr>
<tr>
<td><code>linux::displaylinuxlist -address 0xC00703fc -type workqueue_struct -next list next</code></td>
<td>List the current workqueues, 0xC00703fc should be the address of workqueues. Available only if the kernel is stopped.</td>
</tr>
</tbody>
</table>

---

**`linux::loadsymbolics`**

Load the symbolics for the selected module.

**Syntax**

`loadsymbolics <absolute_file_path>`

---

**`linux::refreshmodules`**

Lists loaded modules.

**Syntax**

`refreshmodules`

---

**`linux::selectmodule`**

Sets the current module.

**Syntax**

`selectmodules <index>`
**linux::unloadsymbolics**

Unloads the symbolics for the specified module.

**Syntax**

```
unloadsymbolics
```

---

**loadsym**

Load a symbolic file.

**Syntax**

```
loadsym <filename> [PIC load addr (hex)]
```

**Examples**

The following table lists and defines examples of the `loadsym` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>loadsym</code> myapp.elf</td>
<td>Loads the debug information in myapp.elf into the debugger.</td>
</tr>
<tr>
<td><code>loadsym</code> mypicapp.elf 0x40000</td>
<td>Loads the debug information in mypicapp.elf into the debugger; symbolics addresses are adjusted based on the alternate load address of 0x40000.</td>
</tr>
</tbody>
</table>

---

**log**

Logs the commands or lists entries of a debug session. If issued with no parameters, the command lists all open log files.
Debugger Shell

Debugger Shell Commands

Syntax

log c|s <filename>
log off [c|s] [all]
log

Parameter

c
Command specifier.
s
Lists entry specifier.
<filename>
Name of a file.

Examples
The following table lists and defines examples of the log command.

Table 4.36 log Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>Lists currently opened log files.</td>
</tr>
<tr>
<td>log s</td>
<td>Log all display entries to file session.log.</td>
</tr>
<tr>
<td>session.log</td>
<td></td>
</tr>
<tr>
<td>log off c</td>
<td>Close current command log file.</td>
</tr>
<tr>
<td>log off</td>
<td>Close current command and log file.</td>
</tr>
<tr>
<td>log off all</td>
<td>Close all log files.</td>
</tr>
</tbody>
</table>

mc::config

Display or edit multicore groups options.

Syntax

cconfig [useAllCores|haltGroups|smartSelect [enable|disable]]
Examples

The following table lists and defines examples of the `mc::config` command.

Table 4.37 mc::config Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mc::config</code></td>
<td>Shows the current configuration settings.</td>
</tr>
<tr>
<td><code>mc::config useAllCores disable</code></td>
<td>Disables Use All Cores mode.</td>
</tr>
<tr>
<td><code>mc::config haltGroups enable</code></td>
<td>Enables halt groups (where supported)</td>
</tr>
<tr>
<td><code>mc::config smartSelect disable</code></td>
<td>Disables smart select mode so that the `mc::group enable</td>
</tr>
</tbody>
</table>

**mc::go**

Resumes multiple cores.

**Syntax**

`mc::go`

**Remarks**

`mc::go` resumes the selected cores associated with the current thread context (see “switchtarget”).

**mc::group**

Display or edit multicore groups.
Debugger Shell

Debugger Shell Commands

Syntax

```
group

    group new <type-name> [<name>]
    group rename <name>|<group-index> <new-name>
    group remove <name>|<group-index> ...
    group removeall
    group enable|disable [-context ops|swbps|hwbps|wps ...] <group-index> ...
```

Examples

The following table lists and defines examples of the `mc::group` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mc::group</code></td>
<td>Shows the defined groups, including indices for use in the <code>mc::group rename enable remove</code> set of commands.</td>
</tr>
<tr>
<td><code>mc::group new 8572</code></td>
<td>Creates a new group for target type 8572. The group name will be based on the target name and will be unique. The enablement of the group elements will be all non-cores enabled, all cores disabled.</td>
</tr>
<tr>
<td><code>mc::group rename 0 &quot;My Group Name&quot;</code></td>
<td>Renames the group at index 0 to &quot;My Group Name&quot;.</td>
</tr>
<tr>
<td><code>mc::group enable 0 1.0</code></td>
<td>Enables the group at index 0 and the element at index 1.0 of the <code>mc::group</code> command.</td>
</tr>
<tr>
<td><code>mc::group enable -context swbps hwbps 0</code></td>
<td>Enables the group contexts for software and hardware breakpoints at index 0 of the <code>mc::group</code> command. Note, the index must correspond to a group, not a specific core.</td>
</tr>
<tr>
<td><code>mc::group remove &quot;My Group Name&quot;</code></td>
<td>Removes the group named &quot;My Group Name&quot;.</td>
</tr>
<tr>
<td><code>mc::group removeall</code></td>
<td>Removes all groups.</td>
</tr>
</tbody>
</table>

`mc::kill`

Terminate multiple cores.
Debugger Shell

Debugger Shell Commands

Syntax
mc::kill

Remarks
mc::kill terminates the debug session for the selected cores associated with the current thread context (see "switchtarget").

mc::reset
Reset multiple cores.

Syntax
mc::reset

Remarks
mc::reset resets the debug session associated with the current thread context (see "switchtarget").

mc::restart
Restart multiple cores.

Syntax
mc::restart

Remarks
mc::restart restarts the debug session for the selected cores associated with the current thread context (see "switchtarget").

mc::stop
Suspend multiple cores.

Syntax
mc::stop
Debugger Shell
Debugger Shell Commands

Remarks

\texttt{mc::stop} stops the selected cores associated with the current thread context (see “\texttt{switchtarget}”).

\textbf{mc::type}

Display or edit target types

\textbf{SYNTAX}

type

type import <filename>

type remove <filename>|<type-index> ...

type removeAll

\textbf{Examples}

The following table lists and defines examples of the \texttt{mc::type} command.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Command} & \textbf{Description} \\
\hline
\texttt{mc::type} & Shows the target types available for multicore debugging as well as type indices for use by the \texttt{mc::type remove} and \texttt{mc::group new} commands. \\
\hline
\texttt{mc::type import 8572_jtag.txt} & Creates a new type from the JTAG configuration file. \\
\hline
\texttt{mc::type remove 8572_jtag.txt} & Removes the type imported from the specified file. \\
\hline
\texttt{mc::group removeAll} & Removes all imported types. \\
\hline
\end{tabular}
\end{table}

\textbf{mem}

Read and write memory.
Debugger Shell

Debugger Shell Commands

Syntax

mem <addr-spec> [<range>] [-s|-ns] [%<conv>] [-np]
mem
mem <addr-spec> [<range>] [-s|-ns] [%<conv>] =<value>
mem -ms

Overview

The mem command reads or writes one or more adjacent "cells" of memory, where a cell is defined as a contiguous block of bytes. The cell size is determined by the <cell-size> parameter or by the config command option "MemWidth".

Options

The following table lists and defines options of the mem command.

Table 4.40  mem Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[none]</td>
<td>With no option, next block of memory is read.</td>
</tr>
<tr>
<td>&lt;ms&gt;</td>
<td>On architectures supporting multiple memory spaces, specifies the memory space in which &lt;addr&gt; is to be found. See the help for the option -ms of display or mem for more information on memory spaces. If unspecified, the setting &quot;config MemIdentifier&quot; is used.</td>
</tr>
<tr>
<td>&lt;addr&gt;</td>
<td>Target address in hex.</td>
</tr>
<tr>
<td>&lt;count&gt;</td>
<td>Number of memory cells.</td>
</tr>
<tr>
<td>x&lt;cell-size&gt;</td>
<td>Memory is displayed in units called cells, where each cell consists of &lt;cell-size&gt; bytes. If unspecified, the setting &quot;config MemWidth&quot; is used.</td>
</tr>
<tr>
<td>h&lt;access-size&gt;</td>
<td>Memory is accessed with a hardware access size of &lt;access-size&gt; bytes. If unspecified, the setting &quot;config MemAccess&quot; is used.</td>
</tr>
<tr>
<td>{8,16,32,64}bit</td>
<td>Sets both &lt;cell-size&gt; and &lt;access-size&gt;.</td>
</tr>
<tr>
<td>-np</td>
<td>Don't print anything to the display, only return the data.</td>
</tr>
</tbody>
</table>
Debugger Shell
Debugger Shell Commands

Table 4.40 mem Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ms</td>
<td>On architectures supporting multiple memory spaces, displays the list of available memory spaces including a mnemonic and/or an integer index which may be used when specifying a target address.</td>
</tr>
<tr>
<td>-s</td>
<td>Specifies whether each value is to be swapped. For memory, specifies whether each cell is to be swapped. With a setting of -ns, target memory is written in order from lowest to highest byte address. Otherwise, each cell is endian swapped. If unspecified, the setting &quot;config MemSwap&quot; is used.</td>
</tr>
<tr>
<td>%&lt;conv&gt;</td>
<td>Specifies the type of the data. Possible values for &lt;conv&gt; are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal.</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal.</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal.</td>
</tr>
<tr>
<td>%f</td>
<td>Floating point.</td>
</tr>
<tr>
<td>[%Q&lt;n&gt;]F</td>
<td>Fixed or Fractional. The range of a fixed point value depends on the (fixed) location of the decimal point. The default location is set by the config command option &quot;MemFixedIntBits&quot;.</td>
</tr>
<tr>
<td>%s</td>
<td>ASCII.</td>
</tr>
</tbody>
</table>

Examples
The examples assume the following settings:
• radix x
• config MemIdentifier 0
• config MemWidth 32
• config MemAccess 32
• config MemSwap off

The following table lists and defines examples of the mem command.
### Table 4.41 mem Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>Display the next block of memory.</td>
</tr>
<tr>
<td>mem 10000</td>
<td>Change memory range 0x10000-3 as one cell.</td>
</tr>
<tr>
<td>mem 1:10000</td>
<td>Change memory range 0x10000-3, memory space 1, as one cell.</td>
</tr>
<tr>
<td>mem 10000 16</td>
<td>Display memory range 0x10000-3f as 16 cells.</td>
</tr>
<tr>
<td>mem 10000 16x1h8</td>
<td>Display 16, 1-byte cells, with a hardware access size of 8-bytes per read.</td>
</tr>
<tr>
<td>mem 10000 8bit</td>
<td>Display one byte, with a hardware access size of one byte.</td>
</tr>
<tr>
<td>mem 10000 -np</td>
<td>Return one cell, but don’t print it to the Command Window.</td>
</tr>
<tr>
<td>mem 10000 -s</td>
<td>Display one byte with the data endian-swapped.</td>
</tr>
<tr>
<td>mem 10000 %d</td>
<td>Display one cell in decimal format.</td>
</tr>
<tr>
<td>mem -ms</td>
<td>Display the available memory spaces, if any.</td>
</tr>
<tr>
<td>mem 10000 =10</td>
<td>Change memory range 0x10000-3 to 0x10 (because radix is hex).</td>
</tr>
<tr>
<td>mem 1:10000 =20</td>
<td>Change memory range 0x10000-3, memory space 1, to 0x20.</td>
</tr>
<tr>
<td>mem 10000 16x1h8 =31</td>
<td>Change each of 16, 1-byte cells to 0x31, using a hardware access size of 8-bytes per write.</td>
</tr>
<tr>
<td>mem 10000 -s %d =200</td>
<td>Change memory range 0x10000-3 to c8000000.</td>
</tr>
</tbody>
</table>

### next

Runs to next source line or assembly instruction in current frame.

**Syntax**

```plaintext
next
```
Debugger Shell

Debugger Shell Commands

Remarks
If you execute the next command interactively, the command returns immediately, and target-program execution starts. Then you can wait for execution to stop (for example, due to a breakpoint) or type the stop command.

If you execute the next command in a script, the command-line debugger polls until the debugger stops (for example, due to a breakpoint). Then the command line debugger executes the next command in the script. If this polling continues indefinitely because debugging does not stop, press the ESC key to stop the script.

nexti
Execute over function calls, if any, to the next assembly instruction.

Syntax
nexti

Remarks
If you execute nexti command, it will execute the thread to the next assembly instruction unless current instruction is a function call. In such a case, the thread is executed until the function returns.

oneframe
Query or set the one-frame stack crawl mode for the current thread.

Syntax
oneframe [on | off]

Examples
The following table lists and defines examples of the oneframe command.
**pwd**

Lists current working directory.

**Syntax**

pwd

**quitIDE**

Quits the IDE.

**Syntax**

quitIDE

**radix**

Lists or changes the default input radix (number base) for command entries, registers and memory locations. Entering this command without any parameter values lists the current default radix.

**Syntax**

radix [x|d|u|b|f|h]

**Parameter**

x
Debugger Shell

Debugger Shell Commands

Hexadecimal
d
  Decimal
u
  Unsigned decimal
b
  Binary
f
  Fractional
h
  Hexadecimal

Examples

The following table lists and defines examples of the `radix` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radix</td>
<td>Lists the current setting.</td>
</tr>
<tr>
<td>radix d</td>
<td>Change the setting to decimal.</td>
</tr>
<tr>
<td>radix x</td>
<td>Change the setting to hexadecimal.</td>
</tr>
</tbody>
</table>

redirect

Redirect I/O streams of the current target process.

Syntax

`redirect <stream> <destination>`

Options

The following table lists and defines options of the `redirect` command.
Examples

The following table lists and defines examples of the `red` command.

### Table 4.45  red Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>redirect stdout server 27018</code></td>
<td>Redirects output of stdout for the current process to a server socket on local port 27018.</td>
</tr>
<tr>
<td><code>redirect stderr socket logmachine.com 22018</code></td>
<td>Attempts to connect to the server socket at port 22018 on host &quot;logmachine.com&quot; and redirects output of stderr for the current process to that connection.</td>
</tr>
<tr>
<td><code>redirect both stop</code></td>
<td>Ends redirection (if any) currently in place for both stdout and stderr for the current process.</td>
</tr>
</tbody>
</table>
Debugger Shell

Debugger Shell Commands

refresh

Discard all cached target data and refresh views.

**Syntax**

```
refresh [all | -p <pid> <pid> ...]
```

**Options**

The following table lists and defines options of the `refresh` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[none]</td>
<td>No option will refresh current process only.</td>
</tr>
<tr>
<td>all</td>
<td>Refresh all currently debugged processes.</td>
</tr>
<tr>
<td>-p &lt;+pid&gt;</td>
<td>Specify list of process ID for the processes to be refreshed.</td>
</tr>
</tbody>
</table>

**Examples**

```
refresh -p 0 1
```

Refreshes debugger data for debugged processes with PID ‘0’ and ‘1’.

---

reg

Read and write registers.
Syntax

reg export <reg-spec> [<n>] <file>
reg export <file>
reg import <file>
reg <reg-spec> [<n>] [-{d|nr|nv|np} ...] [-s|-ns] [%<conv>]
reg <reg-spec>{..<reg>|#<n>} [-{d|nr|nv|np} ...] [-s|-ns] [%<conv>]
reg all|r|nr: [-{d|nr|nv|np} ...] [-s|-ns] [%<conv>]
reg <reg-spec> [<n>] [-s|-ns] [%<conv>] =<value>
reg <reg-spec>{..<reg>|#<n>} [-s|-ns] [%<conv>] =<value>
reg -regset
reg

Options

The following table lists and defines options of the reg command.

Table 4.47 reg Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>export</td>
<td>Export all or a set (specified through &lt;reg-spec&gt; [&lt;n&gt;]) of registers into a specified file. The traversal of the register hierarchy is recursive. Existing files are overwitten.</td>
</tr>
<tr>
<td>import</td>
<td>Import registers from a specified file.</td>
</tr>
<tr>
<td>[none]</td>
<td>No option is equivalent to reg -regset.</td>
</tr>
<tr>
<td>&lt;reg-spec&gt; [{r</td>
<td>nr}:]&lt;reg&gt;</td>
</tr>
<tr>
<td>&lt;reg&gt;</td>
<td>A register name or a register group name.</td>
</tr>
<tr>
<td>...&lt;reg&gt;</td>
<td>The end point for a range of registers to access.</td>
</tr>
</tbody>
</table>
The following table lists and defines examples of the `reg` command.

Table 4.47  reg Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;n&gt;</td>
<td>Number of registers.</td>
</tr>
<tr>
<td>all</td>
<td>Specifies all registers.</td>
</tr>
<tr>
<td>-d</td>
<td>Print detailed data book information.</td>
</tr>
<tr>
<td>-nr</td>
<td>Print only register group, i.e. no registers.</td>
</tr>
<tr>
<td>-nv</td>
<td>Print only register groups and register names, i.e. no values.</td>
</tr>
<tr>
<td>-np</td>
<td>Do not print anything to the display, only return the data.</td>
</tr>
<tr>
<td>regset</td>
<td>Display the register group hierarchy.</td>
</tr>
<tr>
<td>-s</td>
<td>-ns</td>
</tr>
<tr>
<td>%&lt;conv&gt;</td>
<td>Specifies the type of the data. Possible values for &lt;conv&gt; are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal.</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal.</td>
</tr>
<tr>
<td>%f</td>
<td>Floating point.</td>
</tr>
<tr>
<td>%[Q&lt;n&gt;]F</td>
<td>Fixed or fractional. The range of a fixed point value depends on the (fixed) location of the decimal point. The default location is set by the config command option MemFixedIntBits.</td>
</tr>
<tr>
<td>%s</td>
<td>ASCII</td>
</tr>
</tbody>
</table>

**Examples**

The following table lists and defines examples of the `reg` command.
### Table 4.48  reg Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg -regset</td>
<td>List all the available register sets on the target chip.</td>
</tr>
<tr>
<td>reg R1</td>
<td>Display the value of register R1.</td>
</tr>
<tr>
<td>reg &quot;General Purpose Registers/R1&quot;</td>
<td>Display value of register R1 in the General Purpose Register group.</td>
</tr>
<tr>
<td>reg R1 -d</td>
<td>Display detailed “data book” contents of R1, including bitfields and definitions.</td>
</tr>
<tr>
<td>reg &quot;nr:General Purpose Registers/R1&quot; 25</td>
<td>Beginning with register R1, display the next 25 registers. Register groups are not recursively searched.</td>
</tr>
<tr>
<td>reg R1 =123</td>
<td>Change register R1 to 0x123.</td>
</tr>
<tr>
<td>reg R1..R5 =5432</td>
<td>Change registers R1 through R5 to 0x5432.</td>
</tr>
<tr>
<td>reg &quot;General Purpose Registers/R1&quot; 25 =100</td>
<td>Change register R1 in the General Purpose Register group to 0x100.</td>
</tr>
<tr>
<td>reg export filename</td>
<td>Export all registers from the target to the specified file.</td>
</tr>
<tr>
<td>reg export R1 filename</td>
<td>Export the value of register R1 to the specified file.</td>
</tr>
<tr>
<td>reg export &quot;General Purpose Registers/R1&quot; 25 filename</td>
<td>Beginning with register R1, export the next 25 registers to the specified file.</td>
</tr>
<tr>
<td>reg import filename</td>
<td>Import registers from the specified file.</td>
</tr>
</tbody>
</table>
Debugger Shell

Debugger Shell Commands

reset
Reset the target hardware.

Syntax
reset [h/ard|s/oft] [run]

Options
h/ard|s/oft
The type of reset, either hard or soft. If unspecified, the default depends on the hardware support. If soft is supported, then that is the default. Otherwise, if hard is supported, then that is the default.

run
Let's the target run after the reset, also called "reset to user". Otherwise, the target is halted at the reset vector.

Examples
The following table lists and defines examples of the reset command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>Issue a soft reset if supported, otherwise a hard reset.</td>
</tr>
<tr>
<td>reset s</td>
<td>Issue a soft reset.</td>
</tr>
<tr>
<td>reset hard</td>
<td>Issue a hard reset.</td>
</tr>
<tr>
<td>reset run</td>
<td>Issue a soft reset if supported, otherwise a hard reset. The target is allowed to run after the reset.</td>
</tr>
</tbody>
</table>

restart
Restarts the current debug session.

Syntax
restart
Examples

```plaintext
restart

This command will restart the current debug session.
```

---

```plaintext
restore

Write file contents to memory.
```

**Syntax**

```plaintext
restore -h <filename> [[<ms>:]<addr>|+<offset>] [8bit|16bit|32bit|64bit]
restore -b <filename> [<ms>:]<addr> [8bit|16bit|32bit|64bit]
```

**Options**

The following table lists and defines options of the `restore` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Specifies the input file format as hex or binary.</td>
</tr>
<tr>
<td>-b</td>
<td>Specifies the input file format as hex or binary.</td>
</tr>
<tr>
<td>[&lt;ms&gt;:]&lt;addr&gt;</td>
<td>Address to load to. For hex format, this selection overrides the address specified in the file. For architectures with multiple memory spaces, a memory space id may be specified. See config MemIdentifier and mem -ms for more details.</td>
</tr>
<tr>
<td>&lt;offset&gt;</td>
<td>Loads the contents of the hex file at an offset of the original location.</td>
</tr>
<tr>
<td>8bit</td>
<td>16bit</td>
</tr>
</tbody>
</table>

---

**Examples**

The following table lists and defines examples of the `restore` command.

---
Debugger Shell

Debugger Shell Commands

Table 4.51  restore Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>restore -h dat.txt</td>
<td>Loads the contents of the hex file dat.txt into memory.</td>
</tr>
<tr>
<td>restore -b dat.bin 0x20</td>
<td>Loads the contents of binary file dat.bin into memory beginning at 0x20.</td>
</tr>
<tr>
<td>restore -h dat.bin +0x20</td>
<td>Load the contents of the binary file dat.lod into memory with an offset of 0x20 relative to the address saved in dat.bin.</td>
</tr>
</tbody>
</table>

run

Launch a process

Syntax

run [[-index] <index> | [-name] <debug-config-name>]

Examples

The following table lists and defines examples of the run command.

Table 4.52  run Command-Line Debugger Command—Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>Start a process using the default launch configuration, which is the last run configuration if one exists and index 0 otherwise.</td>
</tr>
<tr>
<td>run -index 3</td>
<td>Start a process using the launch configuration at index 3. Type 'launch' for the current set of launch configurations.</td>
</tr>
<tr>
<td>run -name 3</td>
<td>Start a process using the launch configuration named '3'. Type 'launch' for the current set of launch configurations.</td>
</tr>
<tr>
<td>run 3</td>
<td>Start a process using the launch configuration named '3'. If '3' does not exist then configuration with index 3 will be started.</td>
</tr>
<tr>
<td>run {My Launch Config}</td>
<td>Start debugging using the launch configuration named 'My Launch Config'. Type 'launch' for the current set of launch configurations.</td>
</tr>
</tbody>
</table>
save

Saves the contents of memory locations to a binary file or a text file containing hexadecimal values.

Syntax
save -h|-b [<ms>:]<addr>... <filename> [-a|-o] [8bit|16bit|32bit|64bit]

Parameter
- -h|-b
Sets the output file format to hex or binary. For hex format, the address is also saved so that the contents can easily be restored with the restore command.

[<ms>:]<addr>
Address to read from. For architectures with multiple memory spaces, a memory space id may be specified.

-a
Append specifier. Instructs the command-line debugger to append the saved memory contents to the current contents of the specified file.

- o
Overwrite specifier. Tells the debugger to overwrite any existing contents of the specified file.

8bit|16bit|32bit|64bit
Controls the memory access size.

Examples
The following table lists and defines examples of the save command.
Debugger Shell
Debugger Shell Commands

Table 4.53 save Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set addressBlock1 &quot;p:10..&quot;31&quot; set addressBlock2 &quot;p:10000#20&quot; save -h $addressBlock1 $addressBlock2 hexfile.txt -a</td>
<td>Dumps contents of two memory blocks to the text file <em>hexfile.txt</em> (in append mode).</td>
</tr>
<tr>
<td>set addressBlock1 &quot;p:10..&quot;31&quot; set addressBlock2 &quot;p:10000#20&quot; save -b $addressBlock1 $addressBlock2 binfile.bin -o</td>
<td>Dumps contents of two memory blocks to the binary file <em>binfile.bin</em> (in overwrite mode).</td>
</tr>
</tbody>
</table>

**setpc**

Set the value of the program counter register.
## Debugger Shell Commands

### Syntax

- `setpc [-va|-ve|-vn] -address <address>`
- `setpc [-va|-ve|-vn] -line <line_number> {source_file} {target}`
- `setpc [-va|-ve|-vn] -line [+|-]n`
- `setpc [-va|-ve|-vn] -gsymbol <symbol>`
- `setpc [-va|-ve|-vn] -lsymbol <symbol> <source_file> {target}`
- `setpc [-va|-ve|-vn] -line [+|-]n <symbol>`
- `setpc [-va|-ve|-vn] -line [+|-] <symbol> <source_file> {target}`

### Examples

The following table lists and defines examples of the `setpc` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setpc -address 0x1000</code></td>
<td>Set the PC to address 0x1000.</td>
</tr>
<tr>
<td><code>setpc -line 10</code></td>
<td>Set the PC to source line 10 in the current source file.</td>
</tr>
<tr>
<td><code>setpc -line 10 myfile.c</code></td>
<td>Set the PC to source line 10 in source file 'myfile.c'.</td>
</tr>
<tr>
<td><code>setpc -line +10</code></td>
<td>Set the PC to an offset of 10 lines from the current source line.</td>
</tr>
<tr>
<td><code>setpc -gsymbol myExternFunction</code></td>
<td>Set the PC to the address of the 'myExternFunction' global symbol.</td>
</tr>
<tr>
<td><code>setpc -lsymbol myStaticFunction myfile.c</code></td>
<td>Set the PC to the address of the 'myStaticFunction' local symbol defined in source file 'myfile.c'.</td>
</tr>
<tr>
<td><code>setpc -line +10 myExternFunction</code></td>
<td>Set the PC to the address corresponding to an offset of 10 source lines from the location where 'myExternFunction' global symbol was defined.</td>
</tr>
</tbody>
</table>
Debugger Shell
Debugger Shell Commands

**setpicloadaddr**
Indicate where a PIC executable is loaded.

**Syntax**

\[ \text{setpicloadaddr [symfile] <PIC load addr (hex) | reset>} \]

**Examples**

The following table lists and defines examples of the `setpicloadaddr` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setpicloadaddr 0x40000</code></td>
<td>Tells the debugger the main executable is loaded at 0x40000.</td>
</tr>
<tr>
<td><code>setpicloadaddr myapp.elf 0x40000</code></td>
<td>Tells the debugger myapp.elf is loaded at 0x40000.</td>
</tr>
<tr>
<td><code>setpicloadaddr myapp.elf reset</code></td>
<td>Tells the debugger myapp.elf is loaded at the address set in the ELF.</td>
</tr>
</tbody>
</table>

**Table 4.55  setpicloadaddr Command-Line Debugger Command — Examples (continued)**

**stack**
Print the call stack.
**Syntax**

```
stack [num_frames] [-default]
```

**Examples**

The following table lists and defines examples of the `stack` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stack</code></td>
<td>Prints the entire call stack unless limited with <code>stack -default</code>.</td>
</tr>
<tr>
<td><code>stack 6</code></td>
<td>Prints the 6 innermost call stack levels.</td>
</tr>
<tr>
<td><code>stack -6</code></td>
<td>Prints the 6 outermost call stack levels.</td>
</tr>
<tr>
<td><code>stack 6 -default</code></td>
<td>Limits the number of stack frames shown to the 6 innermost levels.</td>
</tr>
<tr>
<td><code>stack -default</code></td>
<td>Removes the stack frame limit.</td>
</tr>
</tbody>
</table>

---

**status**

Lists the debug status of all existing active targets.

**Syntax**

```
status
```

---

**step**

Steps through a program, automatically executing the `display` command.

**Syntax**

```
step [asm|src] [into|over|out]
step [nve|nxt|fwd|end|aft]
```

**Parameter**

`asm|src`
Controls whether the step is performed at the assembly instruction level or the source code level.

\texttt{into|over|out}

Controls the type of step operation. If unspecified, \texttt{into} is used.

\texttt{nve}

Step non optimized action.

\texttt{nxt}

Step next action.

\texttt{fwd}

Step forward action.

\texttt{end}

Step end of statement action.

\texttt{aft}

Step end all previous action.

\textbf{Examples}

The following table lists and defines examples of the \texttt{step} command.

\begin{table}[h]
\centering
\begin{tabular}{|c|p{10cm}|}
\hline
\textbf{Command} & \textbf{Description} \\
\hline
\texttt{step} & Step into the current source or assembly line. \\
\texttt{step over} & Step over the current source or assembly line. \\
\texttt{step out} & Step out of a function. \\
\texttt{step asm} & Step over a single assembly instruction. \\
\hline
\end{tabular}
\end{table}

\textbf{stepl}

Execute to the next assembly instruction.

\textbf{Syntax}

\texttt{stepl}
stop

Stops a running program (started by a go, step, or next command).

Syntax
stop

Examples
The following table lists and defines examples of the stop command.

Table 4.58 stop Command-Line Debugger Command — Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>Using it after command go/step out/next, this will stop the target program.</td>
</tr>
</tbody>
</table>

switchtarget

Displays information about debugged threads, processes and connections or changes the debug context for subsequent commands.

Syntax
switchtarget [<index> | -cur | -ResetIndex -pid | -tid | -conn | -arch]
switchtarget -tid [-pid=<procID>] [[-arch=<name>] | [-conn=<name>]]
switchtarget -pid [[-arch=<name>] | [-conn=<name>]]
switchtarget -arch [-conn=<name>]
switchtarget -conn [-arch=<name>]
switchtarget [-pid=<procID>] [-tid=<threadID>] [[-arch=<name>] | [-conn=<name>]]

Parameter
index

    Session Index number.
Debugger Shell


debugger shell commands

Examples

The following table lists and defines examples of the `switchtarget` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>switchtarget</code></td>
<td>list currently available debug sessions.</td>
</tr>
<tr>
<td><code>switchtarget 0</code></td>
<td>select the thread with index 0</td>
</tr>
<tr>
<td><code>switchtarget - cur</code></td>
<td>list the index of the current thread.</td>
</tr>
<tr>
<td><code>switchtarget - ResetIndex</code></td>
<td>reset the index counter to 0, not valid while debugging.</td>
</tr>
<tr>
<td><code>switchtarget - tid</code></td>
<td>list the thread IDs of the current process of the current connection.</td>
</tr>
<tr>
<td><code>switchtarget - pid</code></td>
<td>list the process IDs of the debugged processes of the current connection.</td>
</tr>
<tr>
<td><code>switchtarget - pid - arch=EPPC</code></td>
<td>list the process IDs of the debugged processes of EPPC architecture on the current debug system.</td>
</tr>
<tr>
<td><code>switchtarget - pid - conn=Launch-1</code></td>
<td>list the process IDs of the debugged processes of the Launch-1 connection.</td>
</tr>
<tr>
<td><code>switchtarget - arch - conn=Launch-1</code></td>
<td>list the architectures debugged on Launch-1 connection.</td>
</tr>
<tr>
<td><code>switchtarget - conn - arch=EPPC</code></td>
<td>list the name of the connection of EPPC architecture on the current debug system.</td>
</tr>
<tr>
<td><code>switchtarget - pid=0 -tid=0 - arch=EPPC</code></td>
<td>switch current context to thread 0 of process 0 of EPPC architecture on the current debug system.</td>
</tr>
<tr>
<td><code>switchtarget - pid=0 -tid=0 - conn=Launch-1</code></td>
<td>switch current context to thread 0 of process 0 on Launch-1 connection.</td>
</tr>
</tbody>
</table>
**system**

Execute system command.

**Syntax**

```
system [command]
```

**Parameter**

`command`

Any system command that does not use a full screen display.

**Examples**

The following table lists and defines examples of the `system` command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>system del *.tmp</code></td>
<td>Delete from the current directory all files that have the <code>.tmp</code> filename extension.</td>
</tr>
</tbody>
</table>

**var**

Read and write variables or C-expressions.

**Syntax**

```
var
var <var-spec> [-np] [-s|-ns] [%<conv>]
var v: [np] [-s|-ns] [%<conv>]
var <var-spec> [-s|-ns] [%<conv>] =<value>
```

**Options**

The following table lists and defines options of the `var` command.
Debugger Shell
Debugger Shell Commands

Table 4.61 var Command-Line Debugger Command — Options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>No option is equivalent to “var v:”.</td>
</tr>
<tr>
<td>v:</td>
<td>If this option appears with no &lt;var&gt; following it, then all variables pertinent to the current scope are printed.</td>
</tr>
<tr>
<td>&lt;var&gt;</td>
<td>Symbolic name of the variable to print. Can be a C expression as well.</td>
</tr>
<tr>
<td>-np</td>
<td>Don’t print anything to the display, only return the data.</td>
</tr>
<tr>
<td>-s</td>
<td>-ns</td>
</tr>
<tr>
<td>%&lt;conv&gt;</td>
<td>Specifies the type of the data. Possible values for &lt;conv&gt; are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal.</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal.</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal.</td>
</tr>
<tr>
<td>%f</td>
<td>Floating point.</td>
</tr>
<tr>
<td>%[Q&lt;n&gt;]F</td>
<td>Fixed or Fractional. The range of a fixed point value depends on the (fixed) location of the decimal point. The default location is set by the config command option &quot;MemFixedIntBits&quot;.</td>
</tr>
<tr>
<td>%s</td>
<td>ASCII.</td>
</tr>
</tbody>
</table>

Examples
The following table lists and defines examples of the var command.
wait

Tells the debugger to wait for a specified amount of time, or until you press the space bar.

Syntax

wait <time-ms>

Parameter

time-ms

Number of milliseconds to wait.

Examples

The following table lists and defines examples of the wait command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait</td>
<td>Debugger waits until you press the space bar.</td>
</tr>
<tr>
<td>wait 2000</td>
<td>Wait for 2 seconds.</td>
</tr>
</tbody>
</table>

watchpoint

Sets, removes, disables, enables or list watchpoints. You can also set condition on watchpoint.
Debugger Shell
Debugger Shell Commands

Syntax

watchpoint [-{r|w|rw}] {<var>|[<ms>:]<addr> <size>}
watchpoint all|#<id>|<var>|[<ms>:]<addr> off|enable|disable
watchpoint #<id> ignore <count>
watchpoint #<id> cond <c-expr>
watchpoint #<id> type -{r|w|rw}
watchpoint #<id> size <units>

Examples

The following table lists and defines examples of the watchpoint command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>watchpoint</td>
<td>Display all watchpoints.</td>
</tr>
<tr>
<td>watchpoint gData</td>
<td>Set read-write (the default) watchpoint on variable gData.</td>
</tr>
<tr>
<td>watchpoint -r gData</td>
<td>Set read-only watchpoint on variable gData.</td>
</tr>
<tr>
<td>watchpoint all off</td>
<td>Remove all watchpoints.</td>
</tr>
<tr>
<td>watchpoint #4 disable</td>
<td>Disable watchpoint number 4.</td>
</tr>
<tr>
<td>watchpoint 10343 4</td>
<td>Set a watchpoint at memory address 10343 of length 4.</td>
</tr>
<tr>
<td>watchpoint #4 ignore 3</td>
<td>Set ignore count to 3 for watchpoint number 4.</td>
</tr>
<tr>
<td>watchpoint #4 cond x == 3</td>
<td>Set the condition for watchpoint number 4 to fire only if x == 3.</td>
</tr>
<tr>
<td>watchpoint #4 type -rw</td>
<td>Set the access type read/write for watchpoint number 4.</td>
</tr>
<tr>
<td>watchpoint #4 size 8</td>
<td>Set the size to 8 units for watchpoint number 4.</td>
</tr>
</tbody>
</table>
Debugger Script Migration

This chapter describes the migration from the Command window of the CodeWarrior Classic IDE to the debugger shell of the CodeWarrior Eclipse IDE. The Debugger Shell of the CodeWarrior Eclipse IDE uses the same TclScript scripting engine as the Command Window with some notable exceptions and changes, as follows:

- new command line syntax for launching the CodeWarrior Eclipse IDE
- removal of the build commands
- removal of the display commands, which are replaced by GUI preferences
- improved step command syntax
- new commands for starting a debug session

This chapter explains:

- 5.1 Command-Line Syntax
- 5.2 Launching a Debug Session
- 5.3 Stepping
- 5.4 Config Settings

5.1 Command-Line Syntax

Start the CodeWarrior Eclipse IDE and execute a Debugger Shell script with a TclScript script as input from the command-line, as shown in the example below:

D:\SC\eclipse>cwide.exe -vmargsplus -Dcw.script=D:\my_script.tcl

**NOTE** Users familiar with the `-vmargs` option in the CodeWarrior Eclipse IDE should note that CodeWarrior will not work properly if `-vmargs` is used. Please use the custom `-vmargsplus` option in place of the `-vmargs` option.
5.2 Launching a Debug Session

In the CodeWarrior Classic IDE, you use the `project -list` command to browse the list of projects to debug and the `debug` command to start a debug session. However, in the CodeWarrior Eclipse IDE, you use the following commands in the Debugger Shell:

- `launch`: to list the launch configurations
- `debug`: to start a debug session
- `run`: to start a process

Figure 5.1 The launch Command

Figure 5.2 The debug and run Commands
NOTE The `debug` command in the CodeWarrior Eclipse IDE also replaces the `attach` and `connect` commands, which have been removed in the CodeWarrior Eclipse IDE.

The `help launch`, `help debug`, and `help run` commands display the help details of the respective commands, as shown in Figure 5.3:

**Figure 5.3 The help Command**

![Help Command](image)

### 5.3 Stepping

In the CodeWarrior Classic IDE, the `cmdwin::step` command uses the Thread window source view mode to determine if the step is performed at the assembly instruction level or at the source instruction level. The CodeWarrior Eclipse IDE does not support the view mode concept. Use the new commands, `stepi` and `nexti`, at the assembly instructional level. The `stepi` command executes to the next assembly instruction, and the `nexti` command executes to the next assembly instruction unless the current instruction is a function call.

In addition, the syntax of the step commands has been redesigned to match the expected behavior. The `step` command in the CodeWarrior Classic IDE is used to step over a source line. However, in the CodeWarrior Eclipse IDE, the `step` or `step in` command means ‘step into’, which is used to step into a source line and the `next` command means ‘step over’. The `step li` command has been removed. A new command, `finish`, has been added for stepping out of a function.
NOTE For backwards compatibility, you can enable the original CodeWarrior Classic IDE syntax by setting the environment variable, FREESCALE_CMDWIN_CLASSIC_STEP.

5.4 Config Settings

Table 5.1 shows the config command settings that have been removed in the CodeWarrior Eclipse IDE.

Table 5.1 The config Command Settings

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config c</td>
<td>Sets the syntax coloring</td>
</tr>
<tr>
<td>config o</td>
<td>Aborts a script</td>
</tr>
<tr>
<td>config page</td>
<td>Controls the paging behavior</td>
</tr>
<tr>
<td>config s</td>
<td>Sets the page size</td>
</tr>
<tr>
<td>config project</td>
<td>Accesses the build projects</td>
</tr>
<tr>
<td>config target</td>
<td>Accesses the build targets</td>
</tr>
</tbody>
</table>

The remaining config command settings work the same as in the CodeWarrior Classic IDE.
Index

A
access breakpoint - refer to watchpoints 238
actions for debugging 113
Active Configuration 8
active configuration 8
Active debug context 110
Address checkbox 143
Address Line fault 143
attach command 323

B
Breakpoint Actions 96
add a action 98
attach breakpoint actions 99
preference page 97
Breakpoints 84
actions 96
disable 95
enable 95
group breakpoints 94
limit breakpoints to active debug context 93
line breakpoint 87
method breakpoint 88
modify properties 90
regular breakpoints 86
remove 95
remove all 96
restrict to selected targets and threads 92
setting special breakpoints 89
skip all 96
special breakpoints 88
hardware 88
software 88
states 85
view 85
breakpoints
conditional 85
disabled 85, 89
enabled 85, 89
regular 85
temporary 85

C
Cache View 102
opening 103
preserve sorting 104
toolbar menu 105
Change
debugger settings 107
Change Program Counter Value 117
changing
register data views 217
register values 214
checkboxes
Address 143
Bus Noise 143
Use Target CPU 143
Walking 1’s 143
code
disabling breakpoints 95
disabling watchpoints 243
enabling watchpoints 243
modifying breakpoint properties 90
setting watchpoints in 239
viewing watchpoint properties 241
CodeWarrior
overview 3
Codewarrior Drag & Drop support 34
CodeWarrior Projects view 7
Column Headers 8
Command-Line syntax to launch CW Eclipse
IDE 321
commands
   Debug 114
   Restart 116
   Resume 115
   Run 115
   Step Into 114
   Step Out 114
   Step Over 115
common debugging actions 113
conditional breakpoints 85
Config settings in CW Eclipse IDE 324
connect command 323
contents
   of register 217
context label 110
context menus
   using 25
controlling program execution 83
conventions
   figures 2
   for manual 2
   keyboard shortcuts 2
Copy Cache 106
core 109
core index 109
cores, debugging multiple 189
Creating
   MMU configuration 172
Customize Column Headers 10, 11

D
Data Line fault 144
debug command 114
debug command 322
Debug perspective 111
Debug view 109
download view 112
debugger
   restarting 116
   starting 114
debugger data
   symbolics 226
Debugger Settings 107
   changing 107
   revert 108
Debugger settings
   change 107
debugger shell 248
code hints 247
debugger, defined 84
debugging
   common actions 113
   multiple cores 189
   program execution 83
   restarting a session 116
   starting a session 114
definition
   of bus noise 145
   of debugger 84
   of memory aliasing 144
   of watchpoints 238
Deselect AI 10
details
   viewing for registers 217
development-process cycle for software 3
Diagnostic Information Export 26
Disable LRU 106
disabled breakpoint 85, 89
Disassembly View 117
display register view 213
documentation
   formats 1
   structure 1
types 2
Duplicate Action 133

E
e500 core, 110
ecd.exe 13
   build 14
   generateMakefiles 15
   getOptions 14
   references 16
   setOptions 17
   updateWorkspace 18
Enable 106
enabled breakpoint 85, 89
Erase Text 12
execution of program, controlling 83
execution, resuming 115
execution, stopping 115
Export Cache 106

F
figure conventions 2
filter 11
Find and Open File 33
finish command 323
Flash Programmer
run target task 134
Flash programmer 119
add flash device 123
configure target task 122
create a target task 120
flash programmer actions 125
target ram settings 124
Flash Programmer Actions 125
add checksum actions 128
add erase/blank check actions 126
add program/verify actions 127
remove an action 133
flat list viewing 9
Flush 106
Flush Line 106
formats
for documentation 1
Fractional Variable Formats 235
Full Range Converging subtest 145

G
Global Variables window
236

H
hardware diagnostic panels
Memory Read / Write 141
Memory Tests 142
Address 144
Bus Noise 145
Bus Noise in address lines 145

CodeWarrior Common Features Guide
resume program execution 115
run a program 115
set a watchpoint 239
start the debugger 114
step into a routine 114
step out of a routine 114
step over a routine 115
stop program execution 115
use context menus 25
view registers 214
view watchpoint properties 241

I
Import Existing Project 35, 38
Import/Export/Fill Memory 147
create target task 147
exporting memory to a file 151
fill memory with data pattern 153
Invalidate 106
Invalidate Line 106
Inverse LRU 106

K
kernel awareness 110
Kernel Awareness 109, 110
kernel threads 110
Key Mappings 40
keyboard conventions 2

L
launch command 322
launch configuration 107
Launch Group 155
create 155
debug view 159
launching 159
Launching a debug session in CW Eclipse IDE 322
least significant bit 144
Linux 110
Load Multiple Binaries 159
Lock 106
Lock Line 106
Lock Way 107
Lock Ways 107
Long Word option button 141
LSB 144

M
manual conventions 2
Manual Path Mapping
  Adding Path Mapping to Workspace 208
Maximum Invert Convergence subtest 145
memory
clearing watchpoint 170
setting watchpoint 169
memory aliasing, defined 144
Memory Read / Write panel 141
memory tests
  Address 144
  Bus Noise 145
    address lines 145
data lines 146
  Bus Noise test
    Full Range Converging subtest 145
    Maximum Invert Convergence subtest 145
    Sequential subtest 145
    Walking Ones 143
    Walking Ones test
      Address Line fault 143
      Data Line fault 144
      Ones Retention subtest 144
      Retention fault 144
      Walking Ones subtest 144
      Walking Zeros subtest 144
      Zeros Retention subtest 144
Memory Tests panel 142
Address test 144
Bus Noise test 145
  address lines 145
  data lines 146
Walking Ones test 143
Memory View 162
  adding memory monitor 163
clearing watchpoint 170
setting watchpoint 169
MMU configuration
creating 172
MMU Configurator
pages 176
MMU configurator
create MMU configuration 172
open view 188
pages 176
saving MMU configurator settings 175
toolbar 175
MMU configurator pages 176
data MATT page 182
general page 176
program MATT page 178
MMU Configurator Settings
save 175
MMU Configurator View
open 188
MMU configurator view 188
most significant bit 144
MSB 144
multi-core debugging 189
Multicore Group
creating 192
Multicore Groups 191
Multicore OS 110
Multiple homogeneous cores 110

N
New External File 45
next command 323
nexti command 323

O
Ones Retention subtest 144
Opening the Cache Viewer 103
option buttons
  Byte 141
  Long Word 141
  Read 141
  Word 141
  Write 141
options
  Memory Read / Write panel 141
  Memory Tests panel 142
  Scope Loop panel 142
overview
  of CodeWarrior 3

P
P4080 110
Pages
  MMU configurator 176
  Passes text box 143
  Path Mapping 202
    Automatic Path Mapping 203
    Manual Path Mapping 205
perspective
deploy 111
Pop up 11
preserve sorting 104
Preserves sorting 106
process cycle
do deployment 111
program
  resume execution 115
  running 115
  stopping execution 115
Project View 7
  active configuration 8
  column headers 10
  list view 9
  quick search 11
  tree view 9

Q
Quick Search 8, 11

R
Read option button 141
Referenced projects 47
Refresh 105
Register Details view 217
register view
  change register data format 217
  registers
  changing data views of 217
changing values of 214
Register Details view 217
viewing 214
viewing details of 217
registers view
change register values 214
displaying 213
viewing register 214
Registers window
opening 213, 222
regular breakpoints 85
Restart command 116
restarting
debugger 116
Resume command 115
resuming program execution 115
Retention fault 144
Reverting Debugger Settings 108
routine
stepping into 114
stepping out of 114
stepping over 115
Run command 115
run command 322
running
a program 115
S
Scope Loop panel 141
Search 106
Search Again 106
Select All 10
Sequential subtest 145
setting access breakpoint 239
shortcut conventions 2
Show files in a flat view 9
Show files in a hierarchal view 9
software
development process cycle 3
source code
disabling breakpoints 95
disabling watchpoints 243
enabling watchpoints 243
modifying breakpoint properties 90
setting watchpoints in 239
viewing watchpoint properties 241
Speed slider 142
starting
debugger 114
state
disabled, for breakpoints 85, 89
enabled, for breakpoints 85, 89
Step commands in CW Eclipse IDE 323
step in command 323
Step Into command 114
step li command 323
Step Out command 114
Step Over command 115
stepi command 323
stepping into a routine 114
stepping out of a routine 114
stepping over a routine 115
stopping program execution 115
structure
of documentation 1
Symbolics 226
Synchronize 106
Synchronize Line 106
System Browser 109, 110
console view 111
kernal awareness 110
OS application 110
system browser view 227
T
Target Address text box 141
target process selection 79
Target Task View 232
tasks
adding global variables for different processes 236
changing register data views 217
changing register values 214
disabling a breakpoint 95
disabling a watchpoint 243
enabling a watchpoint 243
manipulating variable formats 235
modifying breakpoint properties 90
opening the Registers window 213, 222
restarting the debugger 116
resuming program execution 115
running a program 115
setting a watchpoint 239
starting the debugger 114
stepping into a routine 114
stepping out of a routine 114
stepping over a routine 115
stopping program execution 115
using context menus 25
viewing registers 214
viewing watchpoint properties 241
temporary breakpoints 85
text boxes
  Passes 143
  Target Address 141
Tree and List View 8
Type Ahead 11
types
  of documentation 2

U
Unlock Way 107
Use Target CPU checkbox 143

V
variable formatting 234
variables
  manipulating formats 235
Variables View 233
view
  breakpoints 85
  cache 102
disassembly 117
memory 162
MMU configurator 188
project 7
System Browser 109
system browser 227
target task 232
View Memory 106
viewing
  register details 217
  registers 214
  viewing access breakpoint 241
  Viewing Binaries 161
  viewing registers 214
Views
  Breakpoints 85
  Cache 102
  Disassembly 117
  Memory 162
  MMU configurator 188
  project 7
  system browser 109
  Target Task 232
  Variables 233

W
Walking 1's checkbox 143
Walking Ones subtest 144
Walking Ones test
  Address Line fault 143
  Data Line fault 144
  Retention fault 144
  subtests
    Ones Retention 144
    Walking Ones 144
    Walking Zeros 144
  Zeros Retention 144
Walking Zeros subtest 144
watchpoint
  memory 169, 170
watchpoints
  access breakpoint 238
defined 238
  watchpoints, disabling 243
  watchpoints, enabling 243
  watchpoints, setting 239
  watchpoints, viewing properties for 241
what is
  a debugger 84
Wildcard character support 11
Word option button 141
working with breakpoints 90
Write 105
Write option button 141

Z
Zeros Retention subtest 144