



Freescale Eclipse Extensions Guide





Freescale, the Freescale logo, CodeWarrior, ColdFire, PowerQUICC, and StarCore are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Processor Expert, QorIQ, QUICC Engine, and VortiQa are trademarks of Freescale Semiconductor, Inc. Java and all other Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. 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. These products incorporate SuperFlash® technology licensed from SST. All other product or service names are the property of their respective owners.

© 2008-2011 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

Corporate Headquarters	Freescale Semiconductor, Inc. 6501 William Cannon Drive West Austin, Texas 78735 U.S.A.
World Wide Web	http://www.freescale.com/codewarrior
Technical Support	http://www.freescale.com/support



Table of Contents

1	Introduction	13
	Release Notes	13
	Documentation Structure	13
	Documentation Formats	14
	Documentation Types	14
	Manual Conventions	14
	Figure Conventions	14
	Keyboard Conventions	14
	CodeWarrior IDE Overview	15
	Development Cycle	15
	CodeWarrior IDE Advantages	17
2	IDE Extensions	19
	Cache View	20
	Opening Cache View	20
	Preserving Sorting	21
	Cache View Pull-down Menu	22
	CodeWarrior Drag and Drop Support	24
	CodeWarrior Classic Project Importer	24
	Launching CodeWarrior Classic Project Importer Wizard	25
	Select Project File	27
	Configure Options	27
	Edit Global Settings	30
	Edit Access Paths	31
	Locate Missing Files	34
	Specify Project Name	34
	Concurrent Compilation	36
	Console View	37
	CodeWarrior Projects View	38
	Active Configuration	39
	Tree and List View	40
	Column Headers	40

Table of Contents

Quick Search	42
Core Index Indicators in Homogeneous Multicore Environment	43
System Browser View	43
Console View	45
Extracting CodeWarrior Configuration Details	45
Find and Open File	46
Flash Programmer	47
Create a Flash Programmer Target Task	48
Configure the Flash Programmer Target Task	50
Run Flash Programmer Target Task	64
Hardware Diagnostics	65
Creating Hardware Diagnostics Task	65
Working with Hardware Diagnostic Action Editor	67
Memory Test Use Cases	74
Import/Export/Fill Memory	75
Creating a Task for Import/Export/Fill Memory	75
Importing Data from a File into Memory	77
Exporting Memory Contents to a File	79
Fill Memory with a Data Pattern	81
Key Mappings	83
Linker Command File Navigation	85
Memory Management Unit Configurator	86
Creating an MMU Configuration	87
Saving MMU Configurator Settings	90
MMU Configurator Toolbar	90
MMU Configurator Pages	91
Opening the MMU Configurator View	102
Multiple Compiler Support	104
New External File	106
Problems View	107
Target Management via Remote System Explorer	108
Creating Remote System	108
Creating Hardware or Simulator System Configuration	111
Creating Hardware or Simulator Connection Configuration	114
Creating TRK System Configuration	116

Remote Systems View	117
Automatic Project Remote System Setting Cache	122
Compatibility with Older Products	125
Target Processor Selection	135
Target Tasks View	136
Exporting Target Tasks	136
Importing Target Tasks	137
Flash File to Target	137
Erasing Flash Device.	139
Programming a File.	139
Viewing CodeWarrior Plug-ins.	139
3 Debugger	143
About Debugger	144
Automated Builds	144
ecd.exe Tool Commands	145
build.	145
getOptions	145
generateMakefiles	146
setOptions	147
updateWorkspace.	147
Breakpoints	148
Breakpoints View	148
Breakpoint Annotations.	149
Regular Breakpoints	150
Special Breakpoints.	151
Working with Breakpoints.	152
Breakpoint Actions	158
Selecting Breakpoint Template	162
Build While Debugging	163
CodeWarrior Debugger Settings.	164
Modifying Debugger Settings.	165
Reverting Debugger Settings.	167
Stopping Debugger at Program Entry Point	168
Command-Line Debugger Shell	169

Table of Contents

Setting Hardware Breakpoints	170
Context Menus	170
Debug Perspective	171
Debug View	172
Common Debugging Actions	173
Disassembly View	177
Environment Variables in Launch Configuration	178
Launch Group	179
Creating a Launch Group	179
Launching the Launch Group	184
Load Multiple Binaries	185
Viewing Binaries	187
Memory View	188
Opening Memory View	188
Adding Memory Monitor	189
Adding Memory Renderings	190
Mixed Source Rendering	191
Setting Memory Access Size	192
Exporting Memory	193
Importing Memory	194
Setting Watchpoint in Memory View	195
Clearing Watchpoints from the Memory View	196
Memory Browser View	196
Multicore Debugging	197
Multicore Suspend	197
Multicore Resume	198
Multicore Terminate	198
Multicore Restart	199
Multicore Groups	199
Creating a Multicore Group	200
Modifying a Multicore Group	202
Editing a System Type	203
Using Multicore Group Debugging Commands	206
Multicore Breakpoint Halt Groups	207
Multicore Reset	208

On Demand Reset	210
Path Mapping	210
Automatic Path Mapping	211
Manual Path Mapping	213
Redirecting Standard Output Streams to Socket.	217
Refreshing Data During Run Time	219
Registers View	220
Displaying the Registers View	221
Viewing Registers	222
Changing Register Values	222
Exporting Registers	223
Importing Registers	224
Changing Register Data Display Format.	225
Register Details Pane	225
Customizing Register Details Pane	227
Remote Launch	228
Remote Launch View	229
Stack Crawls	230
One Frame Mode.	230
Global Preference	231
Symbolics	233
System Browser View.	234
Opening the System Browser View.	234
Target Connection Lost.	236
Target Initialization Files	237
Selecting Target Initialization File	238
TAP Remote Connections.	240
Creating an Ethernet TAP Remote Connection.	240
Variables	241
Opening the Variables View	241
Adding Variable Location to the View	241
Manipulating Variable Values	243
Fractional Variable Formats	243
Adding Global Variables	244
Cast to Type.	245

Table of Contents

Watchpoints	246
Setting a Watchpoint	247
Creating Watchpoint	248
Viewing Watchpoint Properties	249
Modifying Watchpoint Properties	250
Disabling a Watchpoint	251
Enabling a Watchpoint.	251
Remove a Watchpoint	252
Remove All Watchpoints	252
4 Debugger Shell	253
Executing Previously Issued Commands	254
Using Code Hints	255
Using Auto-Completion	255
Debugger Shell Commands.	256
about	256
alias	256
bp	257
cd	258
change	259
cls.	263
cmdwin::ca.	263
cmdwin::caln	264
config.	265
copy	271
debug	271
dir.	272
disassemble	273
display	274
evaluate	278
finish	280
fl::blankcheck	280
fl::checksum.	281
fl::device	282
fl::diagnose	283

fl::disconnect	284
fl::dump	284
fl::erase	285
fl::image	286
fl::protect	287
fl::secure	287
fl::target	288
fl::verify	289
fl::write	289
funcs	290
getpid	290
go	290
help	291
history	292
jtagclock	292
kill	293
launch	293
linux::displaylinuxlist	293
linux::loadsymbolics	295
linux::refreshmodules	295
linux::selectmodule	295
linux::unloadsymbolics	296
loadsym	296
log	296
mc::go	297
mc::kill	298
mc::reset	298
mc::restart	298
mc::stop	299
mem	299
next	302
nexti	302
oneframe	303
pwd	303
quitIDE	303

Table of Contents

radix	304
redirect	305
refresh	306
reg	307
reset	310
restart	311
restore	311
run	312
save	313
setpc	315
setpicloadaddr	316
stack	317
status	317
step	318
stepi	319
stop	319
switchtarget	320
system	321
var	322
wait	323
watchpoint	324
5 Debugger Script Migration	327
Command-Line Syntax	327
Launching a Debug Session	328
Stepping	329
Config Settings	330
Index	331

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.

In this chapter:

- Release Notes
- Documentation Structure
- Manual Conventions
- About this Manual
- CodeWarrior IDE Overview

NOTE The Freescale Eclipse Extensions 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. See your product's Targeting Manual for documentation of its product-specific features.

Release Notes

Before using the CodeWarrior IDE, read the developer notes. These notes contain important information about 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.

Documentation Structure

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

Documentation Formats

CodeWarrior documentation presents information in various formats:

- **PDF** — Electronic versions of the CodeWarrior manuals, such as the Freescale Eclipse Extensions Guide and the product-specific Targeting manuals.
- **HTML** (Hypertext Markup Language) — HTML versions of the CodeWarrior manuals. To access the HTML version of CodeWarrior manuals, select **Help > Help Contents** from the CodeWarrior IDE menu bar.

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 *Freescale Eclipse Extensions Guide*.

Manual Conventions

This topic explains conventions in the Freescale Eclipse Extensions Guide.

Figure Conventions

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

Keyboard Conventions

The CodeWarrior IDE accepts keyboard shortcuts, or *key bindings*, for frequently used operations. For each operation, this manual lists corresponding key bindings by platform. At any time, you can obtain a list of available key bindings using Key Assist (**Help > Key Assist** or **Ctrl+Shift+L**).

CodeWarrior IDE Overview

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

This topic explains following:

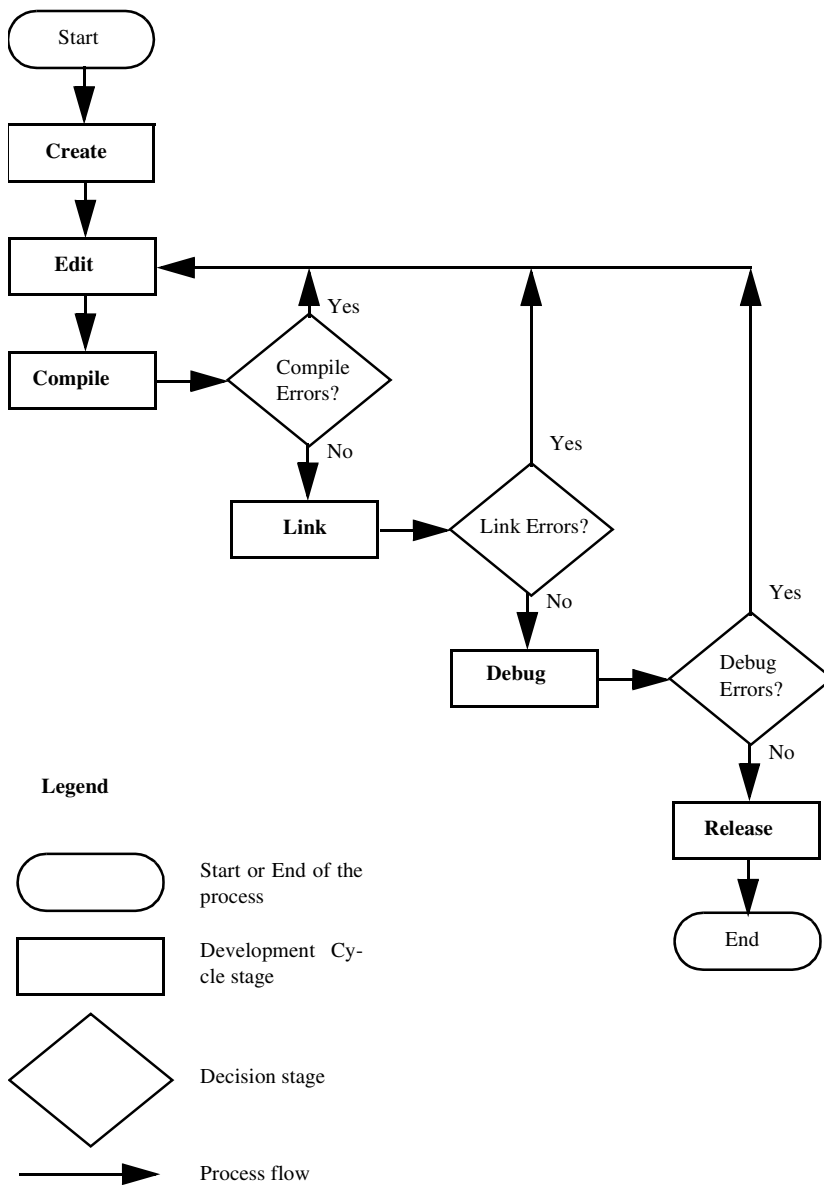
- [Development Cycle](#)
- [CodeWarrior IDE Advantages](#)

Development Cycle

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

1. Begin with an idea for new software.
2. Implement new idea in source code.
3. Compile source code into machine code.
4. Link machine code and create an executable file.
5. Correct errors (debug).
6. Compile, link, and release a final executable file. [Figure 1.1](#) shows the development cycle as a flowchart.

Figure 1.1 Development Cycle Diagram



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. The CodeWarrior IDE 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.



Introduction

CodeWarrior IDE Overview

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.

In this chapter:

- [Cache View](#)
- [CodeWarrior Drag and Drop Support](#)
- [CodeWarrior Classic Project Importer](#)
- [CodeWarrior Projects View](#)
- [Concurrent Compilation](#)
- [Console View](#)
- [Core Index Indicators in Homogeneous Multicore Environment](#)
- [Extracting CodeWarrior Configuration Details](#)
- [Find and Open File](#)
- [Flash Programmer](#)
- [Hardware Diagnostics](#)
- [Import/Export/Fill Memory](#)
- [Key Mappings](#)
- [Linker Command File Navigation](#)
- [Memory Management Unit Configurator](#)
- [Multiple Compiler Support](#)
- [New External File](#)
- [Problems View](#)
- [Target Management via Remote System Explorer](#)
- [Target Processor Selection](#)
- [Target Tasks View](#)
- [Flash File to Target](#)
- [Viewing CodeWarrior Plug-ins](#)

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.

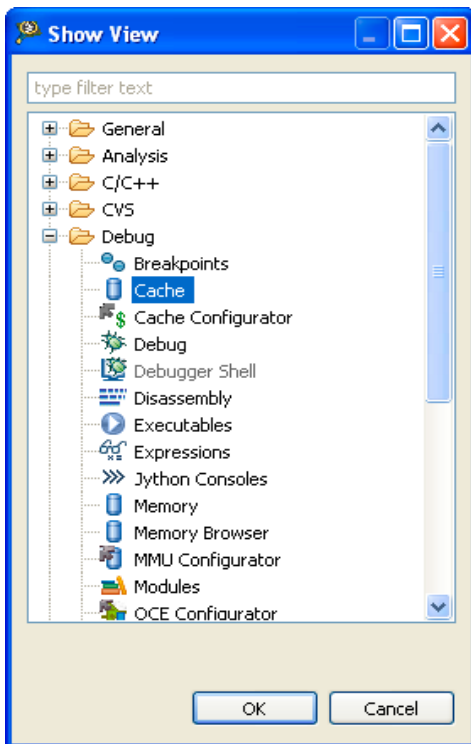
Opening Cache View

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 ([Figure 2.1](#)) appears.

Figure 2.1 Show View Dialog Box

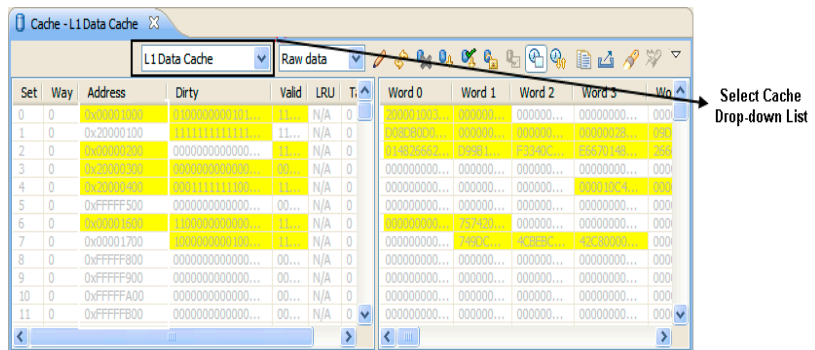


3. Expand the **Debug** group and Select **Cache**.

- Click **OK**.
- The **Cache** view ([Figure 2.2](#)) 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**.

Figure 2.2 Cache View



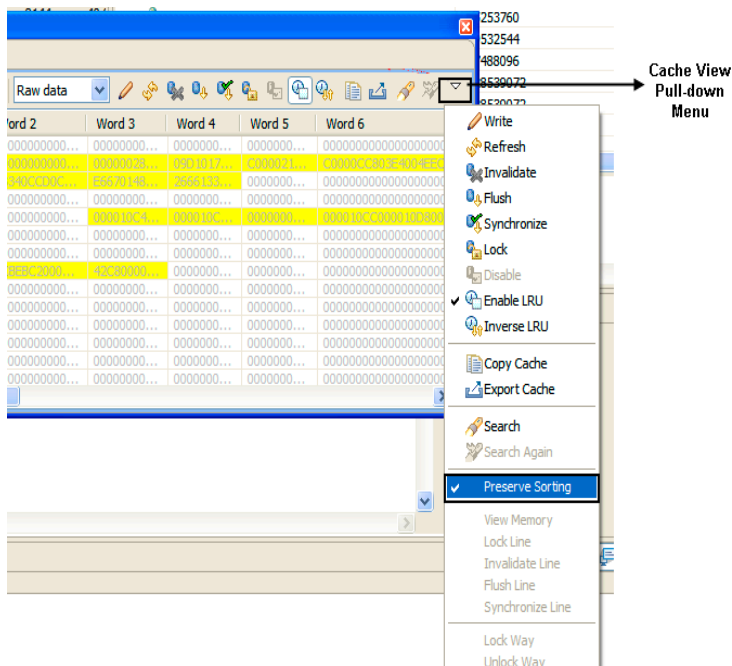
NOTE If the **Select Cache** drop-down list is grayed out in the **Cache** view, then the current target does not support viewing cache.

Preserving Sorting

To preserve sorting of the cache when you update and refresh the cache:

- Start a debugging session.
- Open the **Cache** view. For more information about how to open the **Cache** view, refer [Opening Cache View](#).
- Select the Preserve Sorting command from the **Cache** view pull-down menu ([Figure 2.3](#)).

Figure 2.3 Cache View — Preserve Sorting



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.

Cache View Pull-down Menu

You can perform various actions on the cache using the **Cache** view pull-down menu. Alternatively, you can use the **Cache** view toolbar that includes the same commands as the Cache view pull-down menu. [Table 2.1](#) lists the **Cache** view pull-down menu commands and their description.

Table 2.1 Cache View Pull-down Menu Commands

Command	Description
Write	Commits changes in the Cache view to the cache register of the target hardware, if supported by the target hardware.
Refresh	Reads data from the target hardware and updates the cache display.

Table 2.1 Cache View Pull-down Menu Commands

Command	Description
Invalidate	Invalidates the entire content of the cache.
Flush	Flushes the entire content of the cache. Flushing the cache involves committing uncommitted data to the next level of the memory hierarchy, and then invalidating the data within the cache.
Lock	Locks the cache. Locking cache prevents the cache from fetching the new lines or discarding the current valid lines.
Enable	Turns on the cache.
Disable LRU	Removes the Least Recently Used attribute from the existing display for each cache line.
Inverse LRU	Displays the inverse of the Least Recently Used attribute for each cache line.
Copy Cache	Copies the cache contents to the system clipboard.
Export Cache	Exports the cache contents to a file.
Search	Finds an occurrence of a string in the cache lines.
Search Again	Finds the next occurrence of a string in the cache lines.
View Memory	Views the corresponding memory for the selected cache lines.
Lock Line	Locks the selected cache lines.
Invalidate Line	Invalidates the selected cache lines.
Flush Line	Flushes the entire contents of the selected cache lines.
Lock Way	Locks the cache ways specified with the Lock Ways menu command. 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.
Unlock Way	Unlocks the cache ways specified with the Lock Ways command.
Lock Ways	Specifies the cache ways on which the Lock Way and Unlock Way commands operate.

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.
- Allows user to drop multiple files and folders at once and have them handled 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.

CodeWarrior Classic Project Importer

If you have used a classic version of a CodeWarrior product, you may have some projects that you want to use an Eclipse-hosted CodeWarrior product. Recreating these projects in the Eclipse IDE "by-hand" is a difficult and error-prone task. The CodeWarrior Classic Project Importer wizard automates this migration chore. It translates the classic project's build target settings to Eclipse build configurations and launch configurations.

To import a classic CodeWarrior project:

1. [Launching CodeWarrior Classic Project Importer Wizard](#)
2. [Select Project File](#)
3. [Configure Options](#)
4. [Edit Global Settings](#)
5. [Edit Access Paths](#)
6. [Locate Missing Files](#)

7. [Specify Project Name](#)

Although the CodeWarrior Project Importer wizard successfully imports many classic CodeWarrior project, the CodeWarrior Project Importer wizard is not foolproof. It has these known limitations:

- In some classic CodeWarrior projects, certain runtime files and tool settings might have changed or are not available in the CodeWarrior IDE. Once the CodeWarrior Project Importer wizard finishes importing, you might have to complete the settings in the build and launch configurations manually.
- A classic CodeWarrior project that contains build targets that use different linkers must be imported once per linker used. This is because, in a given invocation, the CodeWarrior Project Importer wizard can import just those build targets that use the same linker.
- The CodeWarrior Project Importer wizard discards a classic CodeWarrior project's link order information. The CodeWarrior IDE does not support the link order feature.

Launching CodeWarrior Classic Project Importer Wizard

The CodeWarrior Classic Project Import wizard lets you import a classic CodeWarrior project to a new Eclipse C/C++ Development Tools (CDT) project.

To import a classic CodeWarrior project:

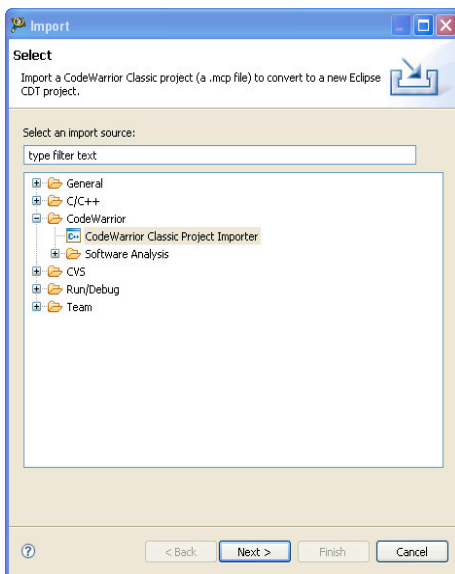
NOTE Before you start importing a classic CodeWarrior project, verify that the project includes all of its files, such as the source code files, the linker command file, and the settings files.

1. Select **File > Import** from the IDE menu bar.
The **Import** dialog box ([Figure 2.4](#)) appears.

IDE Extensions

CodeWarrior Classic Project Importer

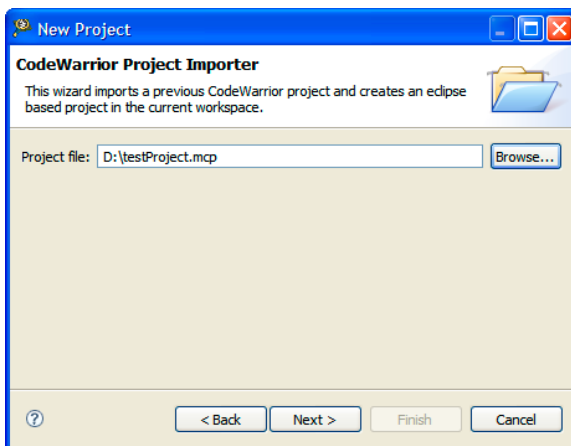
Figure 2.4 Import Dialog Box



2. Expand the **CodeWarrior** folder.
3. Select **CodeWarrior Classic Project Importer**.
4. Click **Next**.

The **CodeWarrior Project Importer** wizard ([Figure 2.5](#)) appears.

Figure 2.5 CodeWarrior Project Importer Wizard



TIP Another way to launch the **CodeWarrior Project Importer** wizard is to drag and drop a classic project (.mcp) file onto the Eclipse Workbench. If you do this, the **CodeWarrior Project Importer** wizard automatically opens with its **Project file** text box set to the path and filename of the selected classic project.

Select Project File

The first page of the CodeWarrior Project Importer wizard lets you specify the classic CodeWarrior project file to import. To specify a classic CodeWarrior project (a .mcp file) to import:

1. Click the **Browse** button on the **CodeWarrior Project Importer** page ([Figure 2.5](#)).
The **Select The CodeWarrior Project File to Import** dialog box appears.
2. Navigate to the directory that contains the classic CodeWarrior project file to import.
3. Select the classic CodeWarrior project file.
4. Click **Open**.
The name and the path of the project file appears in the **Project file** text box.
5. Click **Next**.

The **CodeWarrior Project Importer — Options** page ([Figure 2.6](#)) appears. Use this page to configure the advanced project importer options.

Configure Options

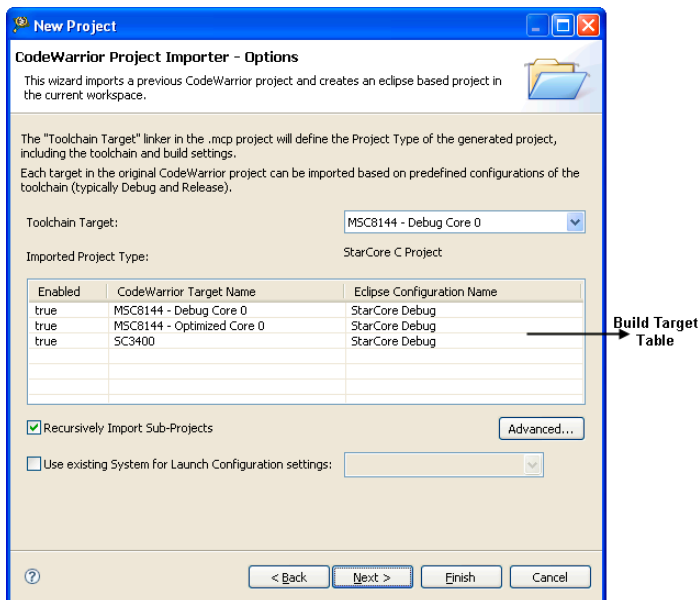
The **CodeWarrior Project Importer — Options** page ([Figure 2.6](#)) summarizes the build target information the project importer retrieves from the selected project file. You can import each target in the classic CodeWarrior project based on the predefined configurations of the toolchain.

In the **Options** page, you can copy all project files in the new Eclipse project directory and enable/disable the generation of the target configuration.

IDE Extensions

CodeWarrior Classic Project Importer

Figure 2.6 CodeWarrior Project Importer — Options Page



[Table 2.2](#) lists and defines each control on the **Options** page:

Table 2.2 Options Page Controls

Control	Description
Toolchain Target drop-down list	Lets you select the build target whose settings you want to modify in the Build Target table.
Build Target table	Displays the build targets retrieved by the project importer from the classic project file. This is used to generate equivalent Eclipse build configurations.
Recursively Import Sub-Projects check box	If checked, imports all the sub-projects as projects along with the main project.
Advanced button	Lets you to specify settings to copy all project files in the new Eclipse project directory.
Use existing System for Launch Configuration settings	If checked, enables you select an existing system to import launch configuration settings.

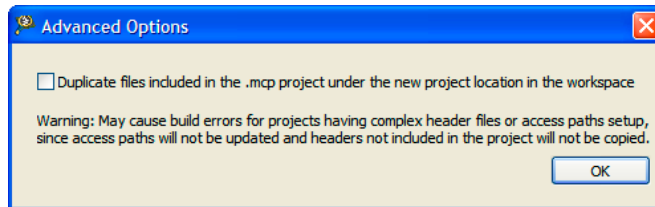
NOTE The classic project file does not contain any information about the files that are not displayed in the project window in the classic IDE. Therefore, the project importer copies only those files to the new Eclipse project that are displayed in the project window.

To copy all project files in the new Eclipse project directory:

1. Click the **Advanced** button.

The **Advanced Options** dialog box ([Figure 2.7](#)) appears.

Figure 2.7 Advanced Options Dialog Box



2. Check the **Duplicate files included in the .mcp project under the new project location in the workspace** check box to copy all project files in the new eclipse project directory.
3. Click **OK**.

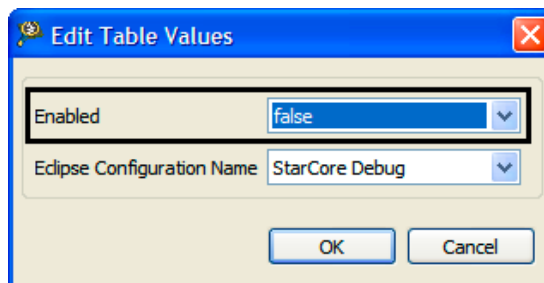
The **Advanced Options** dialog box closes.

To enable or disable the generation of a target configuration.

1. Click a row in the **Build Target** table ([Figure 2.6](#)).

The **Edit Table Values** dialog box ([Figure 2.8](#)) appears.

Figure 2.8 Edit Table Values Dialog Box



IDE Extensions

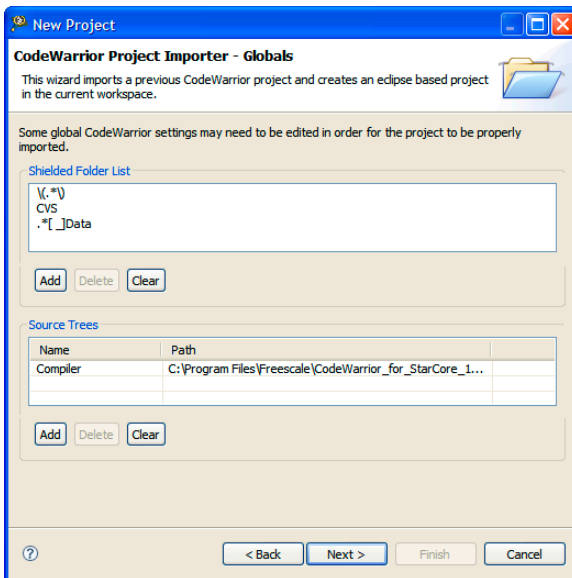
CodeWarrior Classic Project Importer

2. Select `true` from the **Enabled** drop-down list to enable the generation of the target configuration. Alternatively, you can select `false` to disable the generation of the target configuration.
3. Click **OK**.
The **Edit Table Values** dialog box closes.
4. Click **Next**.
The **CodeWarrior Project Importer — Globals** page ([Figure 2.9](#)) appears.

Edit Global Settings

The **CodeWarrior Project Importer — Globals** page ([Figure 2.9](#)) allows you to edit the global settings affecting the project's build options are imported.

Figure 2.9 CodeWarrior Project Importer — Globals Page



[Table 2.3](#) lists and defines each control on the **Globals** page:

Table 2.3 Globals Page Controls

Control	Description
Shielded Folder List	Specifies the folders whose contents are concealed from the IDE's search operations. This is done by placing special characters in the directory name. For example, sample code is concealed in the CodeWarrior Examples folder. You use the Add , Delete , and Clear buttons to modify the list.
Source Tree	Specifies the location of the source trees. If an access path is defined relative to a source tree, the source tree should be listed in this table. The {Project} source tree is defined automatically.
Add button	Adds a new entry to the list.
Delete button	Deletes the selected item.
Clear button	Clears the entire list.

Edit Access Paths

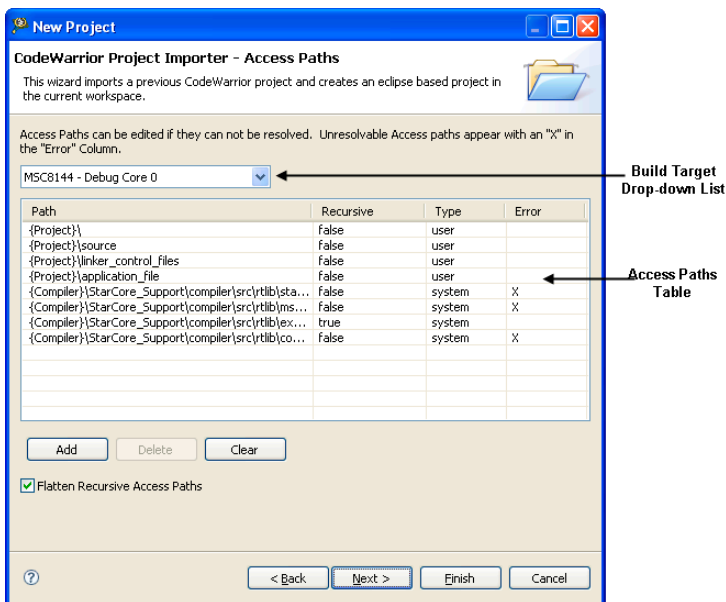
The **CodeWarrior Project Importer** — **Access Paths** page ([Figure 2.10](#)) lets you add or delete access paths.

Access paths are directory paths used by the CodeWarrior tools to search for libraries, runtime support files, and other object files.

IDE Extensions

CodeWarrior Classic Project Importer

Figure 2.10 CodeWarrior Project Importer — Access Paths Page



[Table 2.4](#) lists and defines each control on the **Access Paths** page:

Table 2.4 Access Paths Page Controls

Control	Description
Build Target drop-down list	Lets you choose the build target whose access paths you want to modify.
Access Paths table	Displays the access paths used by the build target selected in the Build Target table. Each row displays the directory path (Path column), specifies whether this path is to be searched recursively (Recursive column), and specifies the type of the path (Type column). You use the Add , Delete , and Clear buttons to modify the information in this table.

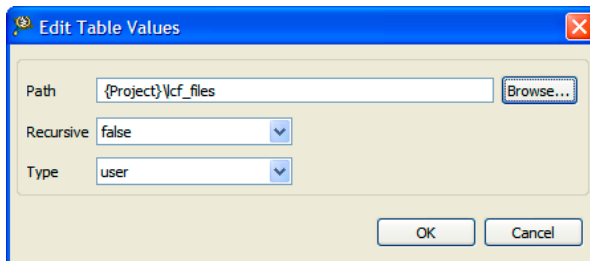
Table 2.4 Access Paths Page Controls

Control	Description
Flatten Recursive Access Paths check box	If checked, the CodeWarrior Project Importer wizard automatically generates separate include paths for each subdirectory that is a part of the recursive path. This check box is checked by default, because most compilers do not support recursive include paths passed from the command line.
Add button	Adds a directory path to the list.
Delete button	Deletes the selected directory path.
Clear button	Clears the entire list.

To edit a value in the Access Paths table:

1. Select a row from the Access Paths table and double-click to edit the access path. The **Edit Table Values** dialog box (Figure 2.11) appears.
 - a. Click the **Browse** button and navigate to the required directory.
 - b. Select `true` from the **Recursive** drop-down list to enable the path to be recursively searched, else select `false`.
 - c. Click **OK**.

Figure 2.11 Edit Table Values Dialog Box



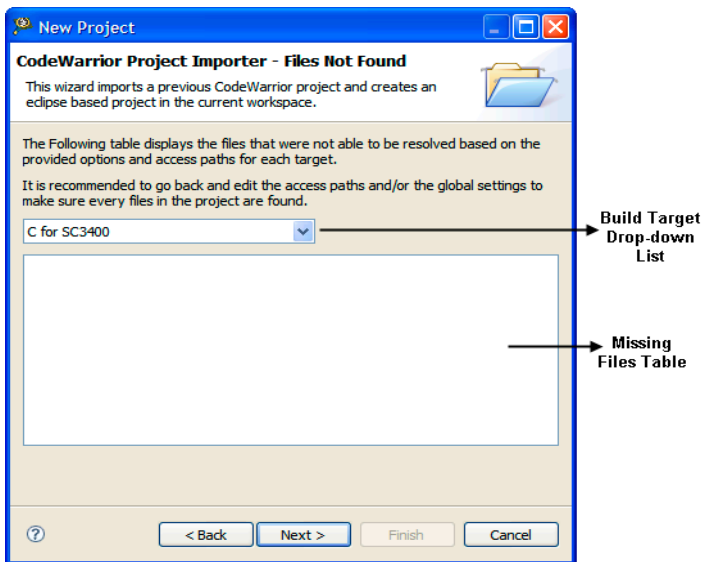
NOTE If project importer fails to find the specified path, it displays an **X** in the **Error** column.

2. Click **Next**.
The **CodeWarrior Project Importer — Files Not Found** page (Figure 2.12) appears.

Locate Missing Files

The **CodeWarrior Project Importer — Files Not Found** page ([Figure 2.12](#)) displays the project files that the project importer could not locate. You can use the Build Target drop-down list to select another build target and view the missing files.

Figure 2.12 CodeWarrior Project Importer — Files Not Found Page



To locate the missing files:

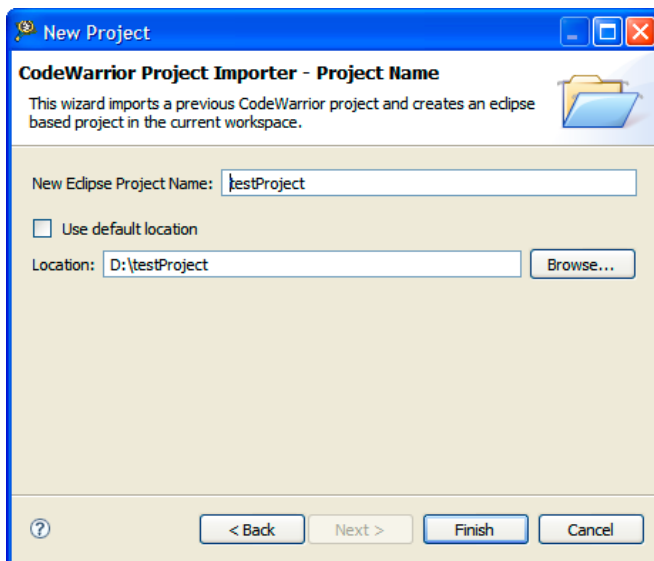
1. Click the **Back** button to adjust the settings in the **Globals** and **Access Paths** pages so that the project importer can locate the missing files. Keep doing this until you have narrowed down the number of missing files.
2. Click **Next**.

The **CodeWarrior Project Importer — Project Name** page ([Figure 2.13](#)) appears.

Specify Project Name

The **CodeWarrior Project Importer — Project Name** page ([Figure 2.13](#)) lets you specify the name of the imported Eclipse project.

Figure 2.13 CodeWarrior Project Importer — Project Name Page



To specify the name of the imported Eclipse project:

1. Enter a name for the imported Eclipse project in the **New Eclipse Project Name** text box.
By default, the project importer inserts the old project name in the text box.
2. Specify the location of the newly imported Eclipse project by performing either of these:
 - Check the **Use default location** check box to save the project to the default Eclipse workspace.
 - Type the location of the project in the **Location** text box. Alternatively, click the **Browse** button to navigate to the desired location.

NOTE By default, the project importer sets the location of the Eclipse project to the location of the Classic CodeWarrior project. You must specify a folder that does not already exist.

3. Click **Finish**.

The project importer imports the selected classic CodeWarrior project to the Eclipse project. The new Eclipse project appears in the **CodeWarrior Project** view of the Workbench window.

NOTE Before debugging the new Eclipse project, you might need to edit the build and launch configuration settings of the project.

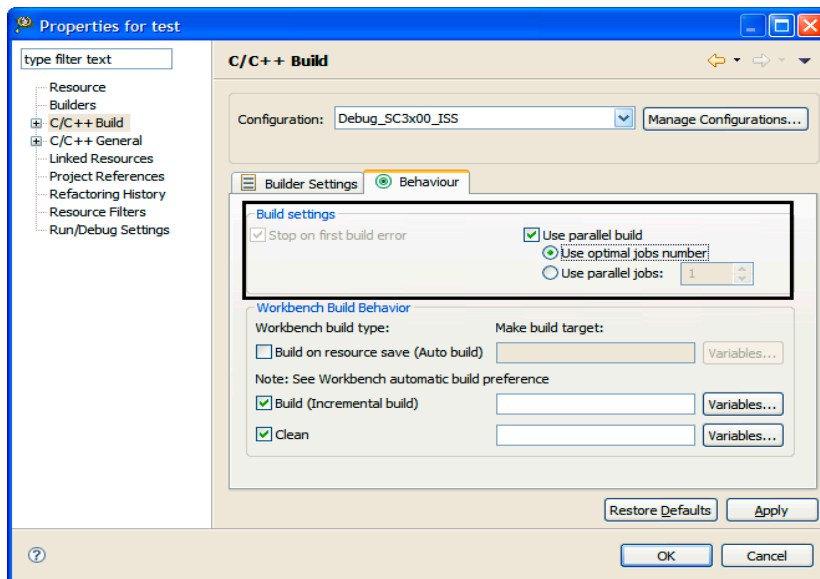
Concurrent Compilation

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 2.14](#)).

Figure 2.14 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**.

Console View

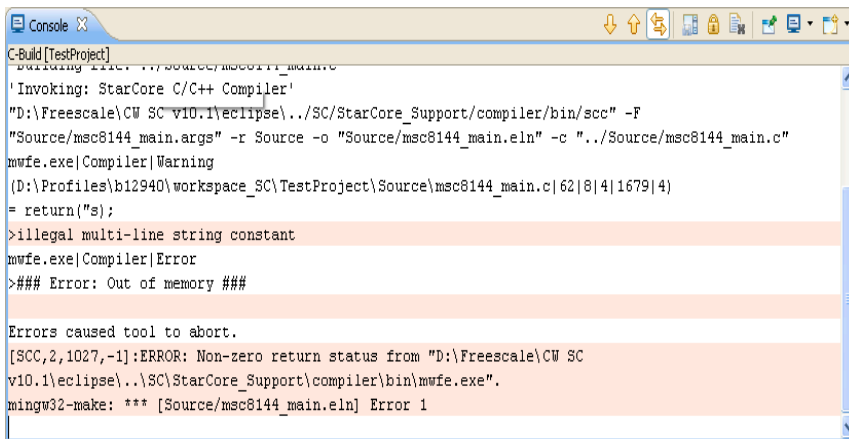
The CodeWarrior **Console** view ([Figure 2.15](#)) displays the the output from the build (standard out and standard error) as it is generated by the build process. Double-clicking

IDE Extensions

CodeWarrior Projects View

the error or warning message in the **Console** view moves the cursor to the error-source in the Editor window.

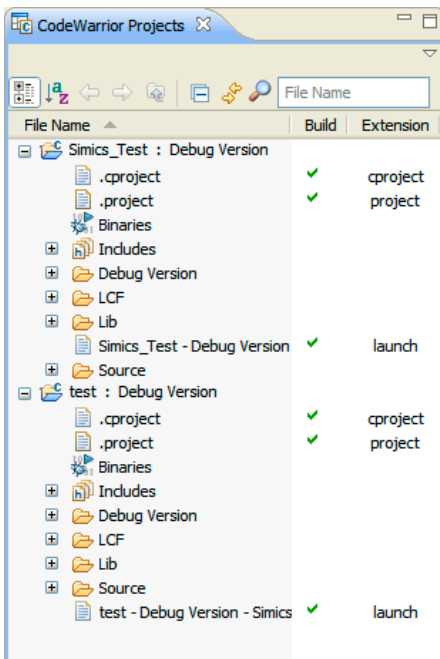
Figure 2.15 Console View



CodeWarrior Projects View

The **CodeWarrior Projects** view ([Figure 2.16](#)) displays all the resources in a workspace.

Figure 2.16 CodeWarrior Projects View



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

- [Active Configuration](#)
- [Tree and List View](#)
- [Column Headers](#)
- [Quick Search](#)

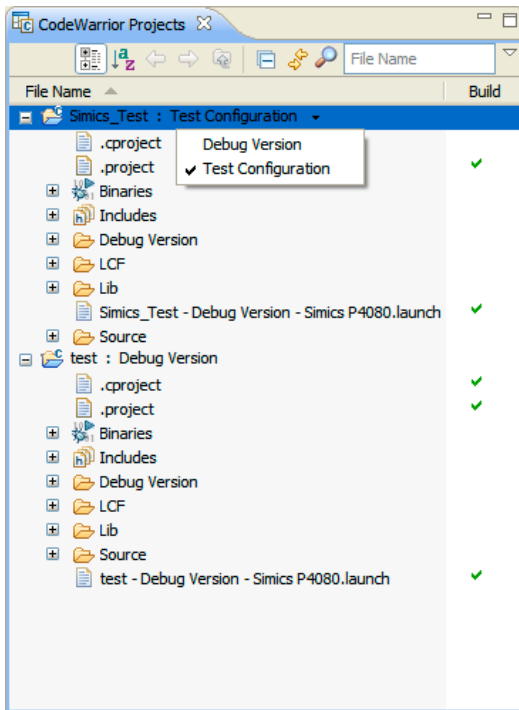
Active Configuration

The **CodeWarrior Projects** view ([Figure 2.17](#)) 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.

IDE Extensions

CodeWarrior Projects View

Figure 2.17 CodeWarrior Projects View — Active Configuration



Tree and List View

CodeWarrior Project view supports both hierarchal tree and flat list viewing of the resources in a workspace.



Click the **Show files in a hierarchal view** button in the **CodeWarrior Project** view toolbar to display the resources in hierarchal tree view.



Click the **Show files in a flat view** button in the **CodeWarrior Project** view toolbar to display the resources in flat list view.

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.

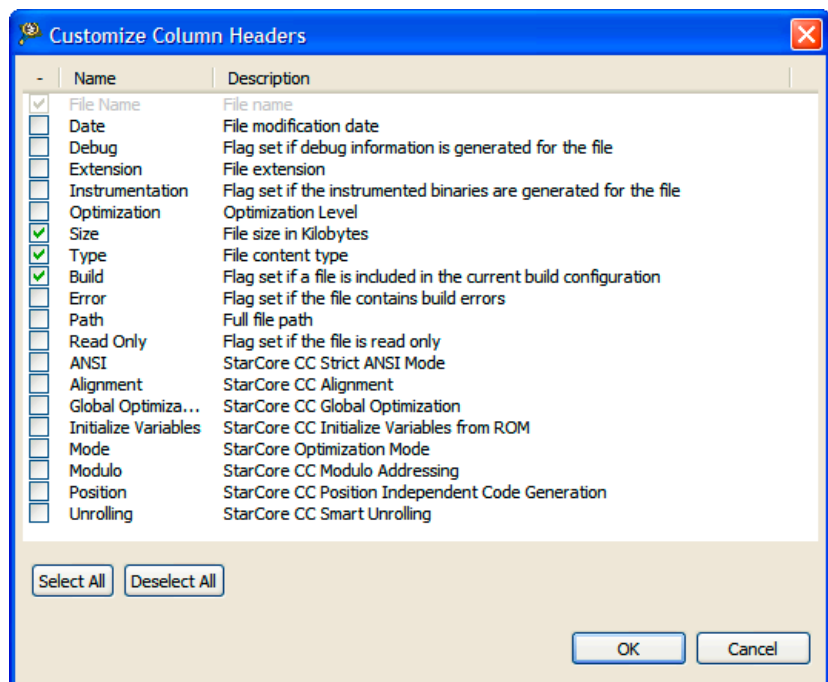
NOTE The files can be sorted in both hierarchal and flat list views. Sorting is not case-sensitive for strings.

To add a column header in the **CodeWarrior Projects** view:

1. Select **Customize Column Headers** from the **CodeWarrior Projects** view pull-down menu.

The **Customize Column Headers** dialog box ([Figure 2.18](#)) appears.

Figure 2.18 Customize Column Headers



2. Check a check box to enable or disable the corresponding column in the **CodeWarrior Projects** view. Alternatively, you can click the **Select All** or **Deselect All** buttons 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.

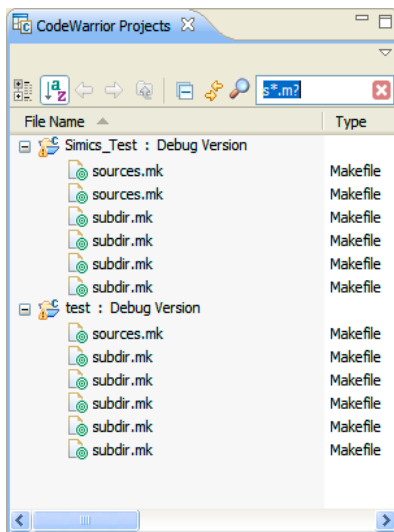
Quick Search


CodeWarrior Projects view provides Quick Search ([Figure 2.19](#)) 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 2.19 CodeWarrior Projects View — Quick Search



 Click the **Popup** button 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.

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

- ✘ Click the **Erase Text** button 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.

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*.

- [System Browser View](#)
- [Console View](#)

System Browser View

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

- [Kernel Awareness](#)
- [OS Application](#)

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*.

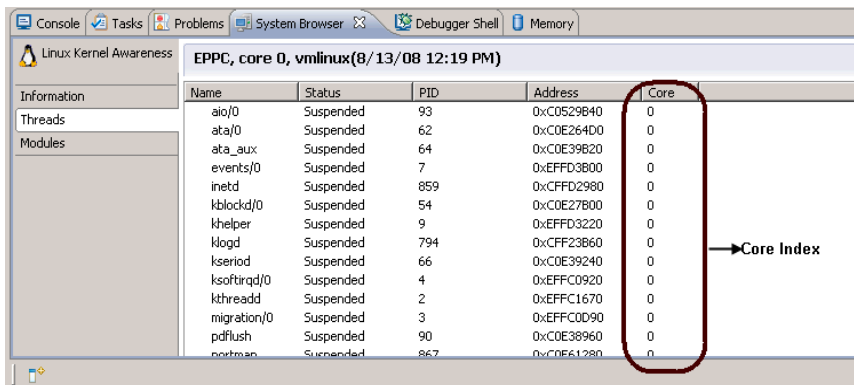
In a multicore OS scenario, the system browser shows kernel threads for all cores being managed by the OS. The **System Browser** view that displays kernel threads indicates the core index for each thread.

IDE Extensions

Core Index Indicators in Homogeneous Multicore Environment

[Figure 2.20](#) shows the core index information for kernel threads in a multicore environment.

Figure 2.20 System Browser View — Kernel Awareness

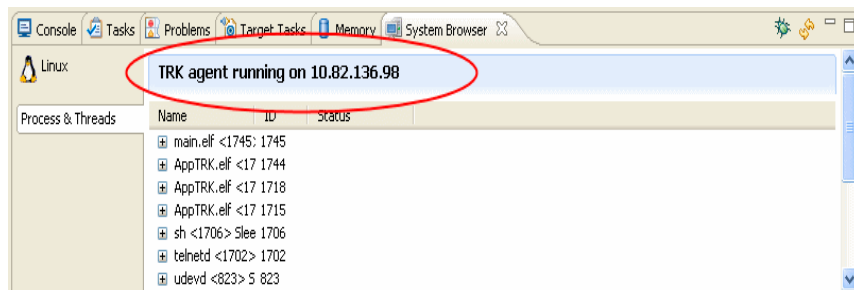


OS Application

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 ([Figure 2.21](#)) shows the IP address or COM port in the context label.

Figure 2.21 System Browser View — OS Application

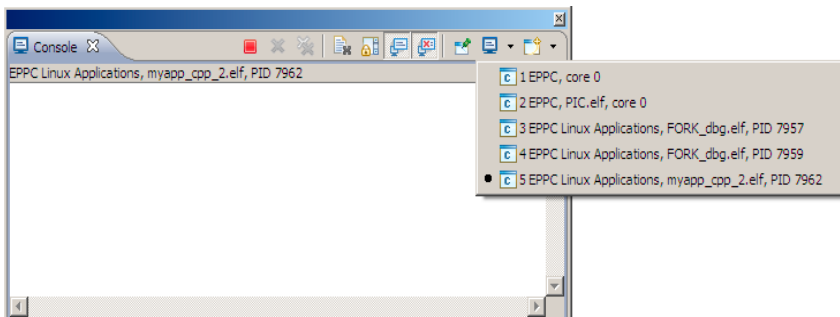


Console View

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.

[Figure 2.22](#) shows the core index in the **Console** view.

Figure 2.22 Console View

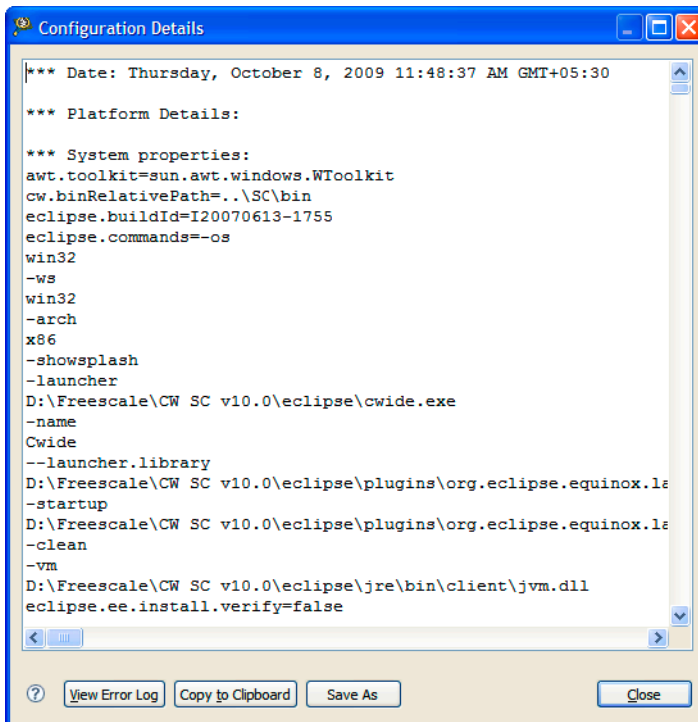


Extracting CodeWarrior Configuration Details

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

1. Select **Help > About CodeWarrior Development Studio** from the IDE menu bar.
The **About Freescale CodeWarrior** dialog box appears.
2. Click the **Configuration Details** button.
The **Configuration Details** dialog box ([Figure 2.23](#)) appears.

Figure 2.23 Configuration Details Dialog Box



3. Click the **Save As** button.

The **Save As** dialog box appears.

4. Specify the text file name in the **File name** text box.
5. Click **Save**.

The configuration information displayed in the text area of the **Configuration Details** dialog box is saved as text file.

Find and Open File

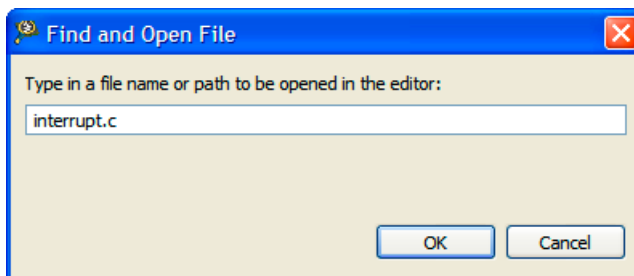
The **Find and Open File** dialog box ([Figure 2.24](#)) enables you to open a selected path or file in the **Editor** area.

To open particular path or file in the Editor area:

1. Select **File > Open Path** from the IDE menu bar.

The **Find and Open File** dialog box ([Figure 2.24](#)) appears.

Figure 2.24 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.

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:

- Program/Verify, see [Program/Verify Actions](#).
- Erase/BlankCheck, see [Erase/Blank Check Actions](#).
- Checksum, see [Checksum Actions](#).
- Diagnostics, see [Diagnostics Actions](#).
- Dump Flash, see [Dump Flash Actions](#).

IDE Extensions

Flash Programmer

- Protect/Unprotect, see [Protect/Unprotect Actions](#).
- Secure/Unsecure, see [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:

1. [Create a Flash Programmer Target Task](#)
2. [Configure the Flash Programmer Target Task](#)
3. [Run Flash Programmer Target Task](#)

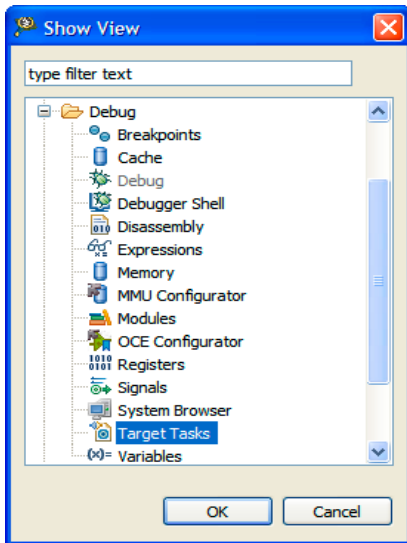
NOTE Click the **Save** button or press **CTRL + s** to save task settings.

Create a Flash Programmer Target Task

1. Select **Windows > Show Views > Others** from the IDE menu bar.

The **Show View** dialog box ([Figure 2.25](#)) appears.

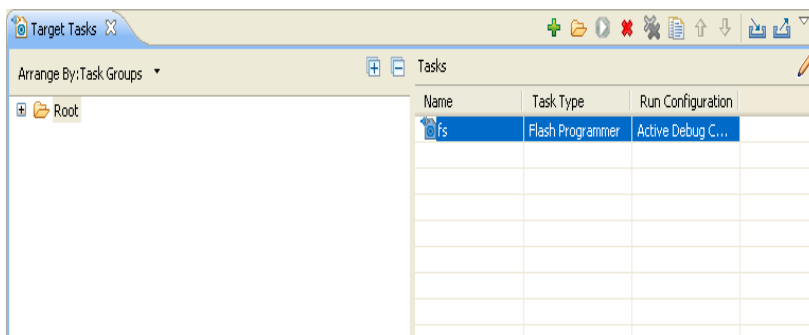
Figure 2.25 Show View Dialog Box



2. Expand the **Debug** group and select **Target Tasks**.
3. Click **OK**.

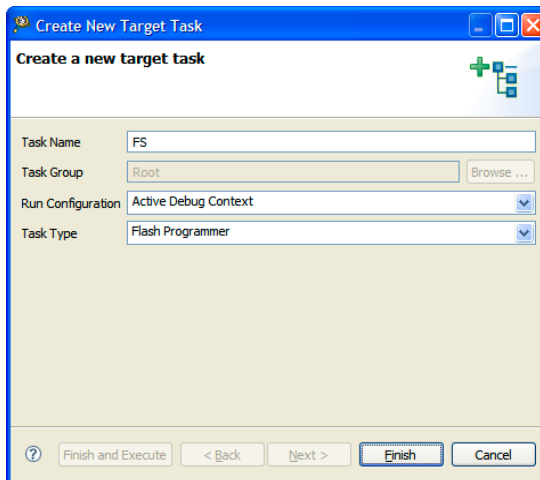
The **Target Tasks** view ([Figure 2.26](#)) appears.

Figure 2.26 Target Tasks View



- Click the **Create a new Target Task** button in the **Target Tasks** view toolbar. The **Create New Target Task** wizard (Figure 2.27) appears.

Figure 2.27 Create New Target Task Window



- In the **Task Name** text box, type name for the new flash programming target task.
- 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.
- 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 ([Figure 2.28](#)) appears. You use this window to configure the flash programmer target task.

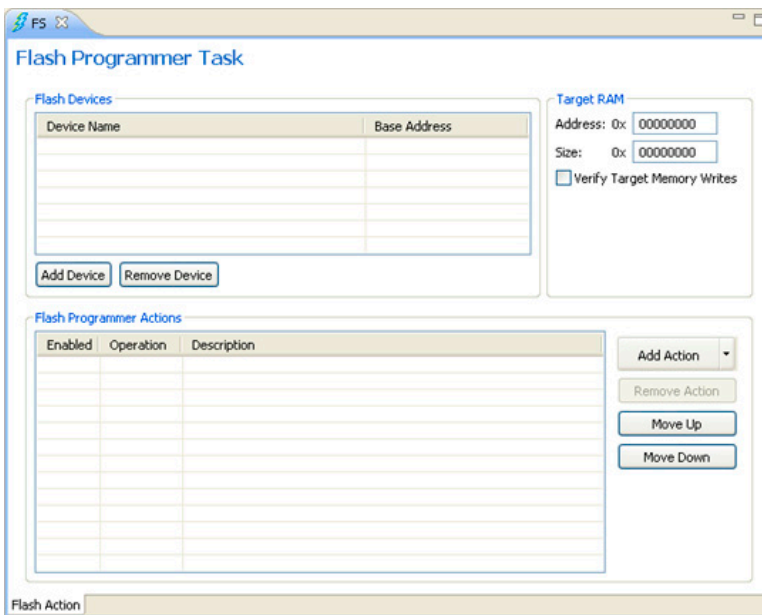
Configure the Flash Programmer Target Task

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

- [Add a Flash Device](#)
- [Specify Target RAM Settings](#)
- [Add Flash Programmer Actions](#)

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

Figure 2.28 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.

Add a Flash Device

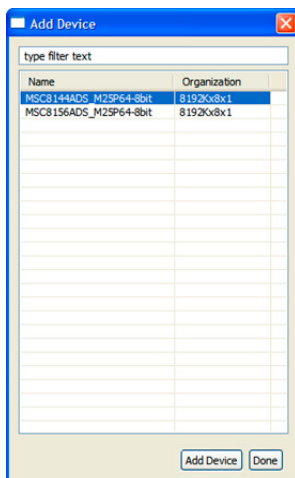
You can add flash devices in the **Flash Devices** table ([Figure 2.28](#)).

To add a flash device to the **Flash Devices** table:

1. Click the **Add Device** button.

The **Add Device** dialog box ([Figure 2.29](#)) appears.

Figure 2.29 Add Device Dialog Box



2. Select a flash device from the device list.
3. Click the **Add Device** button.

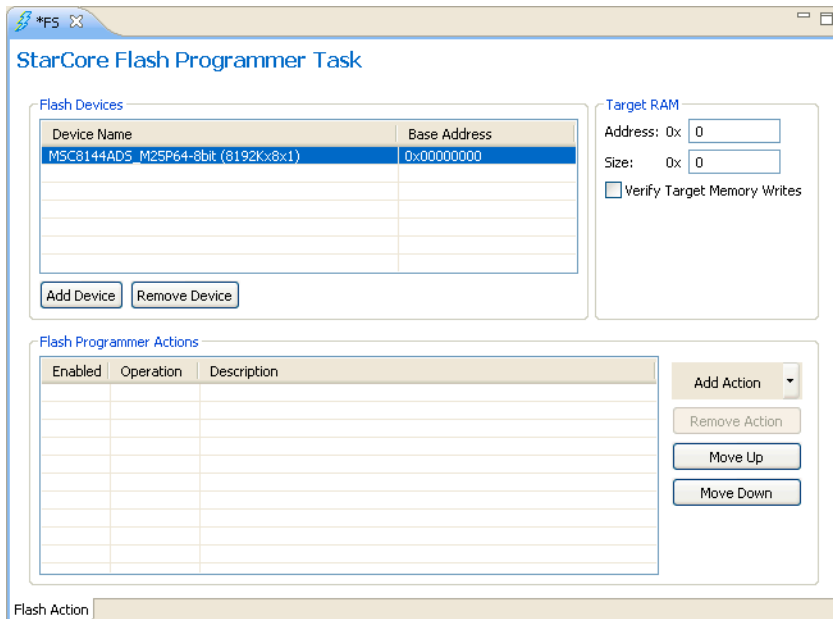
The flash device is added to the **Flash Devices** table in the **Flash Programmer Task** editor window.

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 ([Figure 2.30](#)).

Figure 2.30 Added Device



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**.

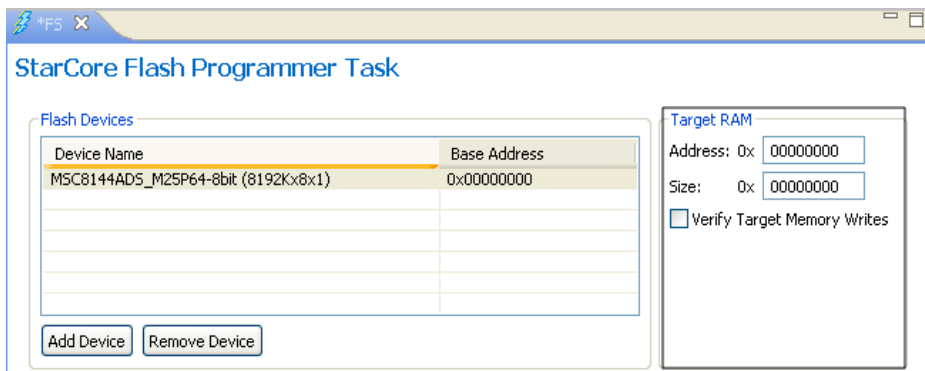
Specify Target RAM Settings

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** group ([Figure 2.31](#)) contains fields to specify settings for the Target RAM.

Figure 2.31 Target RAM Group



- **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.

Add Flash Programmer Actions

In the **Flash Programmer Actions** group in the Flash Programmer Task editor window (Figure 2.28), you can add following actions on the flash device.

- [Erase/Blank Check Actions](#)
- [Program/Verify Actions](#)
- [Checksum Actions](#)
- [Diagnostics Actions](#)
- [Dump Flash Actions](#)
- [Protect/Unprotect Actions](#)
- [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.

IDE Extensions

Flash Programmer

- **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. **Remove Action** button — Enables you to remove a flash program action from the **Flash Programmer Actions** table.
 3. **Move Up** button — Enables you to move up the selected flash action in the **Flash Programmer Actions** table.
 4. **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.

Erase/Blank Check Actions

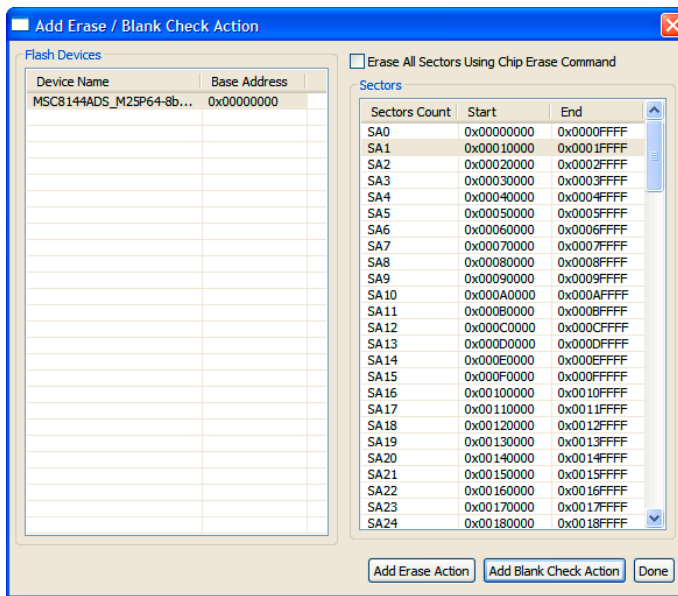
The erase action enables you erase sectors from the flash device. You can also use the erase action to erase the entire flash memory without selecting sectors. The blank check action verifies if the specified areas have been erased from the flash device.

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 ([Figure 2.32](#)) appears.

Figure 2.32 Add Erase/Blank Check Action Dialog Box



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.

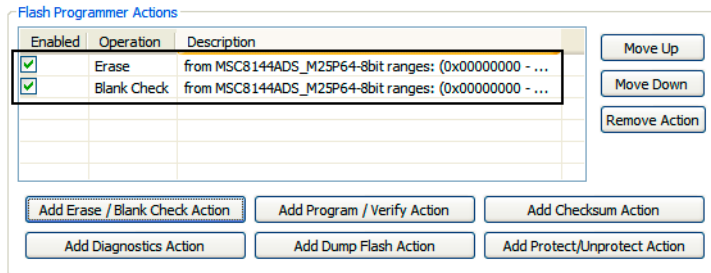
3. Click the **Add Blank Check** button to add a blank check operation on the selected sector.
4. Check the **Erase All Sectors Using Chip Erase Command** check box to erase the entire flash memory.

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 ([Figure 2.33](#)).

Figure 2.33 Added Erase / Blank Check Actions



Program/Verify Actions

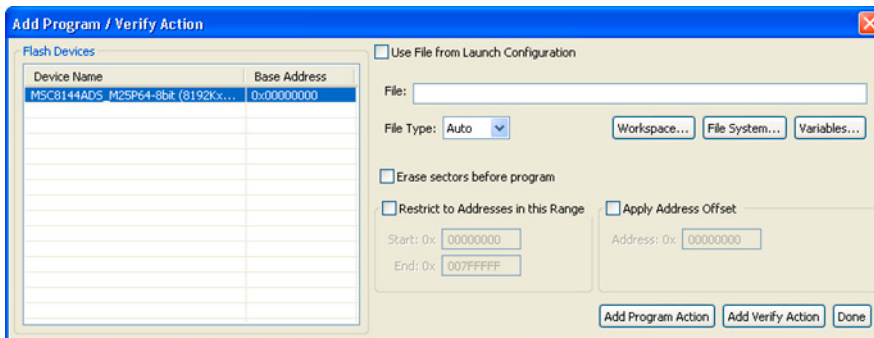
The program action enables you to program the flash device and the verify action verifies the programmed flash device.

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 (Figure 2.34) appears.

Figure 2.34 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.

- 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 **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.
 5. 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.
 6. Click the **Add Program Action** button to add a program action on the flash device.
 7. Click the **Add Verify Action** button to add a verify action on the flash device.
 8. 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 ([Figure 2.35](#)).

Figure 2.35 Added Program/Verify Actions

Enabled	Operation	Description
<input checked="" type="checkbox"/>	Erase	from MSC8144ADS_M25P64-8bit ranges: (0x00050000 - 0x00...
<input checked="" type="checkbox"/>	Blank Check	from MSC8144ADS_M25P64-8bit ranges: (0x00040000 - 0x00...
<input checked="" type="checkbox"/>	Program	file from launch configuration in MSC8144ADS_M25P64-8bit
<input checked="" type="checkbox"/>	Verify	file from launch configuration in MSC8144ADS_M25P64-8bit
<input type="checkbox"/>		
<input type="checkbox"/>		

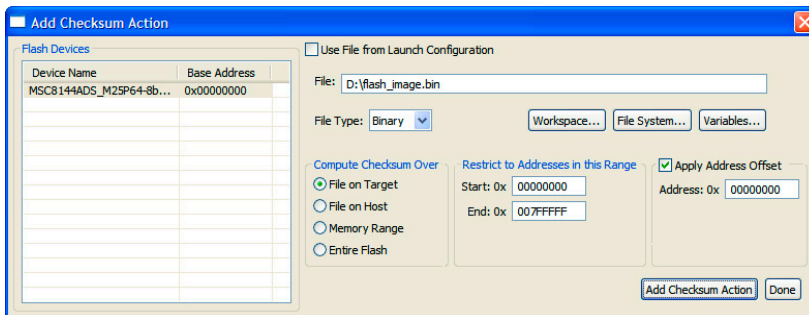
Checksum Actions

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 ([Figure 2.36](#)) appears.

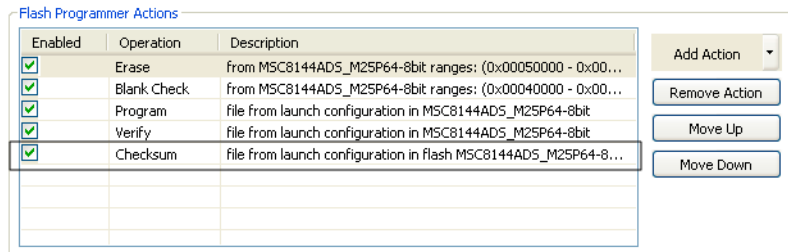
Figure 2.36 Add CheckSum Action Dialog Box



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**.

The **Add Checksum Action** dialog box closes and the added checksum actions appear in the **Flash Programmer Actions** table in the **Flash Programmer Task** editor window ([Figure 2.37](#)).

Figure 2.37 Added Checksum Actions



Diagnostics Actions

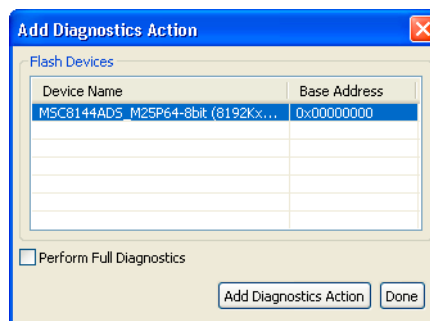
The diagnostics action generates the diagnostic information for a selected flash device.

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 ([Figure 2.34](#)) appears.

Figure 2.38 Add Diagnostics Dialog Box



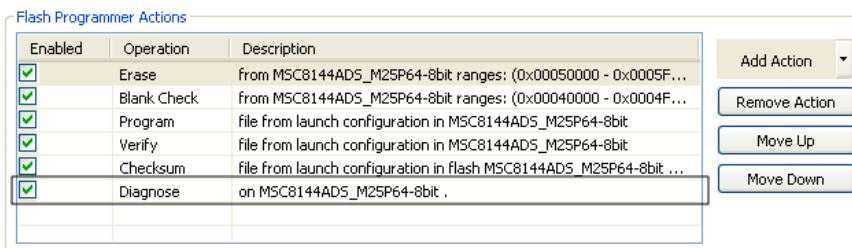
2. Select a device to perform the diagnostics action.
3. Click the **Add Diagnostics Action** button to add diagnostic action on the selected flash device.

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 ([Figure 2.35](#)).

Figure 2.39 Added Diagnostics Actions



Dump Flash Actions

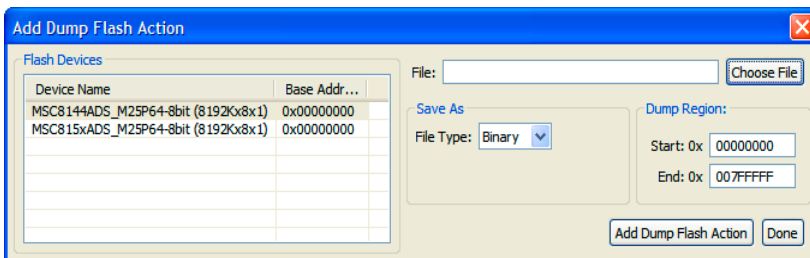
The dump flash action enables you to dump selected sectors of a flash device or the entire flash device.

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 ([Figure 2.34](#)) appears.

Figure 2.40 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 ([Figure 2.35](#)).

Figure 2.41 Added Dump Flash Actions

Enabled	Operation	Description
<input checked="" type="checkbox"/>	Erase	from MSC8144ADS_M25P64-8bit ranges: (0x00050000 - 0x0005FFFF).
<input checked="" type="checkbox"/>	Blank Check	from MSC8144ADS_M25P64-8bit ranges: (0x00040000 - 0x0004FFFF).
<input checked="" type="checkbox"/>	Program	file from launch configuration in MSC8144ADS_M25P64-8bit
<input checked="" type="checkbox"/>	Verify	file from launch configuration in MSC8144ADS_M25P64-8bit
<input checked="" type="checkbox"/>	Checksum	file from launch configuration in flash MSC8144ADS_M25P64-8bit usin...
<input checked="" type="checkbox"/>	Diagnose	on MSC8144ADS_M25P64-8bit .
<input checked="" type="checkbox"/>	Dump Flash	MSC8144ADS_M25P64-8bit from 0x00000000 to 0x007FFFFF and sa...

Buttons: Add Action (dropdown), Remove Action, Move Up, Move Down

Protect/Unprotect Actions

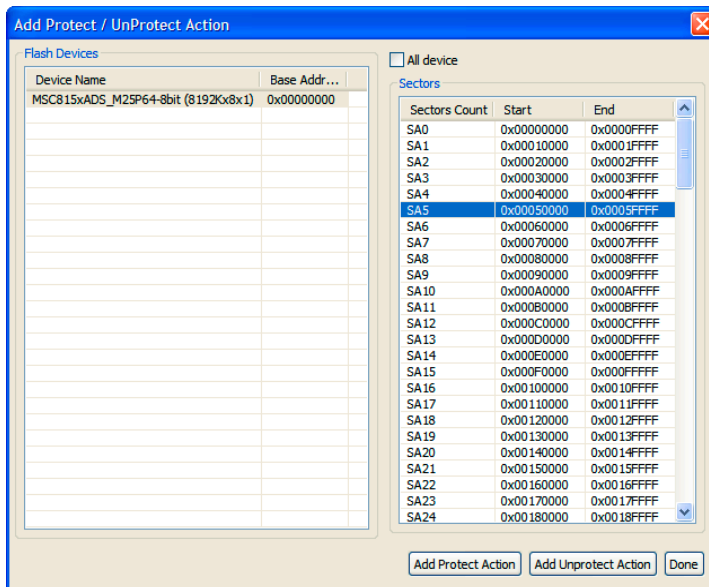
The protect/unprotect actions enable you to change the protection of a sector in the flash device.

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 ([Figure 2.32](#)) appears.

Figure 2.42 Add Protect / Unprotect Action Dialog Box



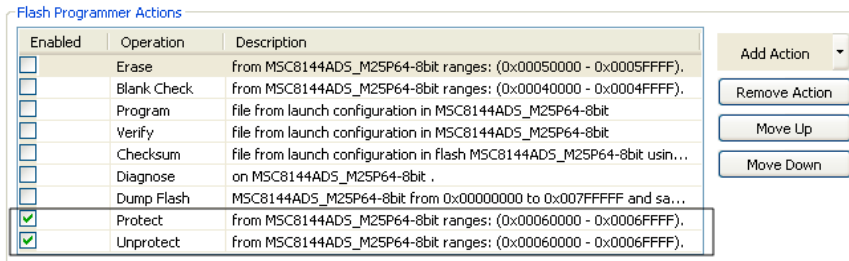
2. Select a sector from the **Sectors** table and click the **Add Protect Action** button to add a protect operation on the selected sector.

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 ([Figure 2.33](#)).

Figure 2.43 Added Protect / Unprotect Actions



Secure/Unsecure Actions

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

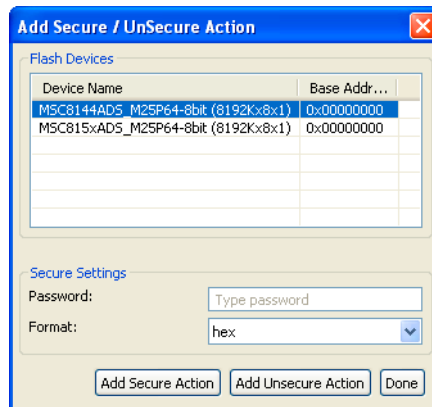
NOTE The Secure/Unsecure flash actions are not supported for StarCore devices.

To add a secure/unsecure action:

1. Select the **Secure/Unsecure Action** from the **Add Action** drop-down list.

The **Add Secure/UnSecure Action** dialog box ([Figure 2.32](#)) appears.

Figure 2.44 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.

Remove an Action

To remove a flash programmer action from the **Flash Programmer Actions** table:

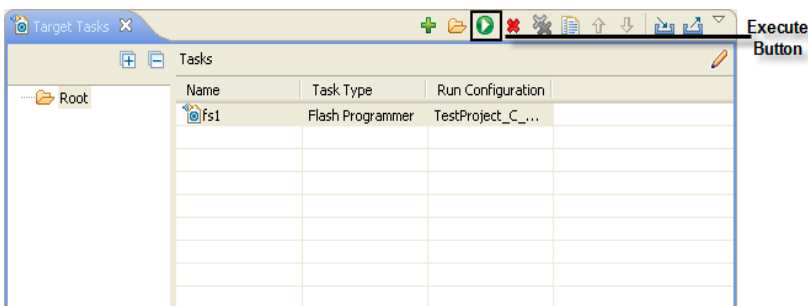
1. Select the action in the **Flash Programmer Actions** table.
2. Click the **Remove Action** button.

The selected action is removed from the **Flash Programmer Action** table.

Run Flash Programmer Target Task

To execute the configured flash programmer target task, select a target task and click the **Execute** button (Figure 2.45) in the **Target Tasks** view toolbar. Alternatively, right-click on a target task and select **Execute** from the context menu.

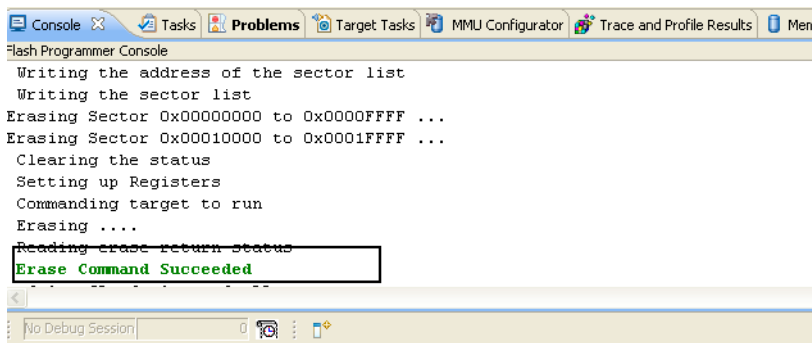
Figure 2.45 Run Target Task



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 (Figure 2.46). The green color indicates the success and the red color indicates the failure of the task.

Figure 2.46 Console View



```
Flash Programmer Console
Writing the address of the sector list
Writing the sector list
Erasing Sector 0x00000000 to 0x0000FFFF ...
Erasing Sector 0x00010000 to 0x0001FFFF ...
Clearing the status
Setting up Registers
Commanding target to run
Erasing ...
Reading erase return status
Erase Command Succeeded
```

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.

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**.

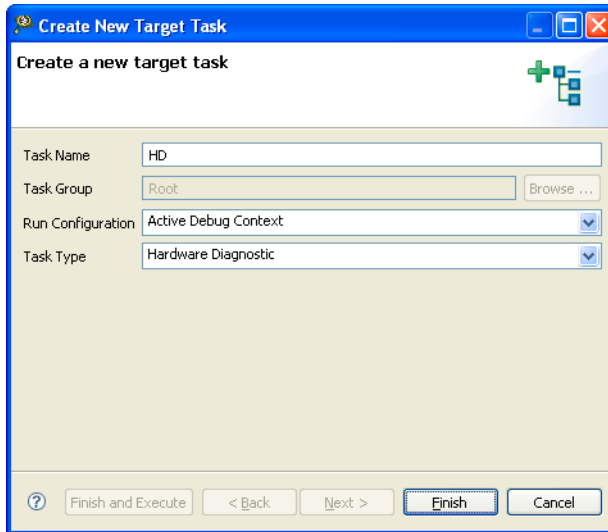
IDE Extensions

Hardware Diagnostics

- 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 ([Figure 2.47](#)) appears.

Figure 2.47 Create New target Task Wizard



- Type name for the new task in the **Task Name** text box.
- 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.

- Select **Hardware Diagnostic** from the **Task Type** drop-down list.
- 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.

Working with Hardware Diagnostic Action Editor

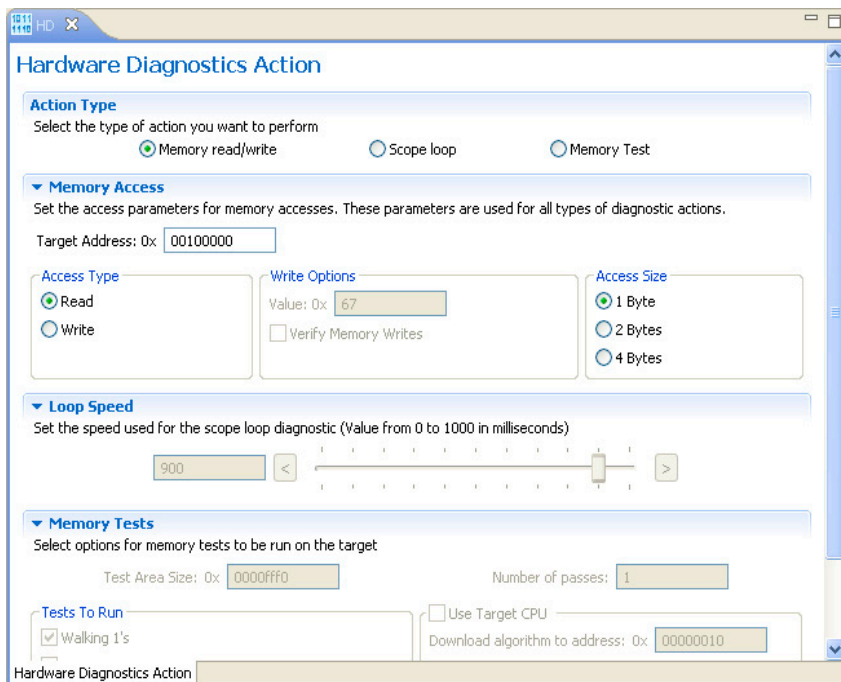
The **Hardware Diagnostic Action** editor is used to configure a hardware diagnostic task. To open the **Hardware Diagnostic Action** editor for a particular task, double-click the task in the **Target Tasks** view.

The **Hardware Diagnostics Action** editor window includes the following groups:

- [Action Type](#)
- [Memory Access](#)
- [Loop Speed](#)
- [Memory Tests](#)

[Figure 2.48](#) shows the **Hardware Diagnostics Action** editor window.

Figure 2.48 Hardware Diagnostics Action Editor Window



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.

Memory Access

The **Memory Access** pane ([Figure 2.49](#)) configures diagnostic tests for performing memory reads and writes over the remote connection interface. [Table 2.5](#) lists and describes the items in the pane.

Figure 2.49 Hardware Diagnostics Action Editor - Memory Access Pane

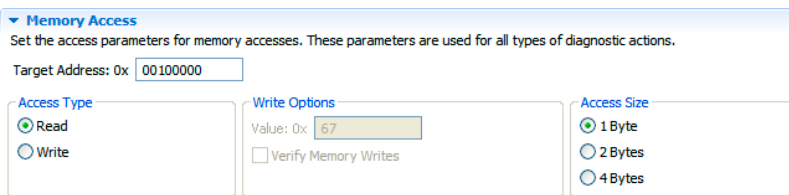


Table 2.5 Memory Access Pane Items

Item	Description
Read	Select to have the hardware diagnostic tools perform read tests.
Write	Select to have the hardware diagnostic tools perform write tests.
1 Byte	Select to have the hardware diagnostic tools perform byte-size operations.
2 Bytes	Select to have the hardware diagnostic tools perform word-size operations.
4 Bytes	Select to have the hardware diagnostic tools perform long-word-size operations.

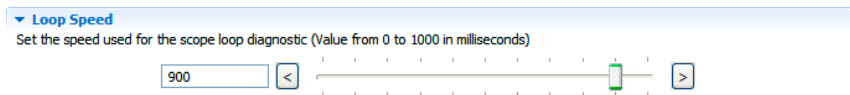
Table 2.5 Memory Access Pane Items (continued)

Item	Description
Target Address	Specify the address of an area in RAM that the hardware diagnostic tools should analyze. The tools must be able to access this starting address through the remote connection (after the hardware initializes).
Value	Specify the value that the hardware diagnostic tools write during testing. Select the Write option to enable this text box.

Loop Speed

The **Loop Speed** pane ([Figure 2.50](#)) 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. [Table 2.6](#) lists and describes the items in the section.

Figure 2.50 Hardware Diagnostics Action Editor - Loop Speed Pane



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.

Table 2.6 Loop Speed Pane Items

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

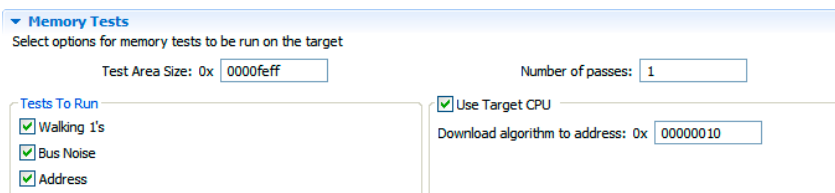
Memory Tests

The **Memory Tests** pane ([Figure 2.51](#)) lets you perform three hardware tests:

- [Walking Ones](#)
- [Address](#)
- [Bus Noise](#)

[Table 2.7](#) explains the items in the **Memory Tests** pane.

Figure 2.51 Hardware Diagnostics Action Editor - Memory Tests Pane



▼ **Memory Tests**
Select options for memory tests to be run on the target

Test Area Size: 0x

Number of passes:

Tests To Run

- Walking 1's
- Bus Noise
- Address

Use Target CPU

Download algorithm to address: 0x

You can specify any combination of tests and number of passes to perform. For each pass, the hardware diagnostic tools performs 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.

Table 2.7 Memory Tests Pane Items

Item	Explanation
Walking 1's	<p>Check to have the hardware diagnostic tools perform the Walking Ones test.</p> <p>Clear to have the diagnostic tools skip the Walking Ones test.</p>
Address	<p>Check to have the hardware diagnostic tools perform the Address test.</p> <p>Clear to have the diagnostic tools skip the Address test.</p>
Bus Noise	<p>Check to have the hardware diagnostic tools perform the Bus Noise test.</p> <p>Clear to have the diagnostic tools skip the Bus Noise test.</p>
Test Area Size	Specify the size of memory to be tested. This setting along with Target Address defines the memory range being tested.
Number of Passes	Enter the number of times that you want to repeat the specified tests.
Use Target CPU	<p>Check to have the hardware diagnostic tools download the test code to the hardware device.</p> <p>Clear to have the hardware diagnostic tools execute the test code through the remote connection interface.</p> <p>Execution performance improves greatly if you execute the test code on the hardware CPU, but requires that the hardware has enough stability and robustness to execute the test code.</p>
Download Algorithm to Address	Specify the address where the test driver is downloaded in case the Use target CPU is activated

Walking Ones

This test detects these memory faults:

- **Address Line**—The board or chip address lines are shorting or stuck at 0 or 1. Either condition could result in errors when the hardware reads and writes to the memory location. Because this error occurs on an address line, the data may end up in the wrong location on a write operation, or the hardware may access the wrong data on a read operation.
- **Data Line**—The board or chip data lines are shorting or stuck at 0 or 1. Either condition could result in corrupted values as the hardware transfers data to or from memory.

- Retention—The contents of a memory location change over time. The effect is that the memory fails to retain its contents over time.

The Walking Ones test includes four 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.

Address

This test detects memory aliasing. *Memory aliasing* exists when a physical memory block repeats one or more times in a logical memory space. Without knowing about this condition, you might conclude that there is much more physical memory than what actually exists.

The address test uses a simplistic technique to detect memory aliasing. The test writes sequentially increasing data values (starting at one and increasing by one) to each successive memory location. The maximum data value is a prime number and its specific value depends on the addressing mode so as to not overflow the memory location.

The test uses a prime number of elements to avoid coinciding with binary math boundaries:

- For byte mode, the maximum prime number is 2^8-5 or 251.
- For word mode, the maximum prime number is $2^{16}-15$ or 65521.
- For long word mode, the maximum prime number is $2^{32}-5$ or 4294967291.

If the test reaches the maximum value, the value rolls over to 1 and starts incrementing again. This sequential pattern repeats throughout the memory under test. Then the test reads back the resulting memory and verifies it against the written patterns. Any deviation from the written order could indicate a memory aliasing condition.

Bus Noise

This test stresses the memory system by causing many bits to flip from one memory access to the next (both addresses and data values). *Bus noise* occurs when many bits change consecutively from one memory access to another. This condition can occur on both address and data lines.

Address lines

To force bit flips in address lines, the test uses three approaches:

- Sequential—This approach works sequentially through all of the memory under test, from lowest address to highest address. This sequential approach results in an average number of bit flips from one access to the next.
- Full Range Converging—This approach works from the fringes of the memory range toward the middle of the memory range. Memory access proceeds in this pattern, where *+ number* and *- number* refer to the next item location (the specific increment or decrement depends on byte, word, or long word address mode):
 - the lowest address
 - the highest address
 - (the lowest address) + 1
 - (the highest address) - 1
 - (the lowest address) + 2
 - (the highest address) - 2
- Maximum Invert Convergence—This approach uses calculated end point addresses to maximize the number of bits flipping from one access to the next. This approach involves identifying address end points such that the values have the maximum inverted bits relative to one another. Specifically, the test identifies the lowest address with all 0x5 values in the least significant nibbles and the highest address with all 0xA values in the least significant nibbles. After the test identifies these end points, memory access alternates between low address and high address, working towards the center of the memory under test. Accessing memory in this manner, the test achieves the maximum number of bits flips from one access to the next.

Data lines

To force bit flips in data lines, the test uses two sets of static data, a pseudo-random set and a fixed-pattern set. Each set contains 31 elements—a prime number. The test uses a prime

number of elements to avoid coinciding with binary math boundaries. The sets are unique to each addressing mode so as to occupy the full range of bits.

- The test uses the pseudo-random data set to stress the data lines in a repeatable but pattern-less fashion.
- The test uses the fixed-pattern set to force significant numbers of data bits to flip from one access to the next.

The 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

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.

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.

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.

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.

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.

IDE Extensions

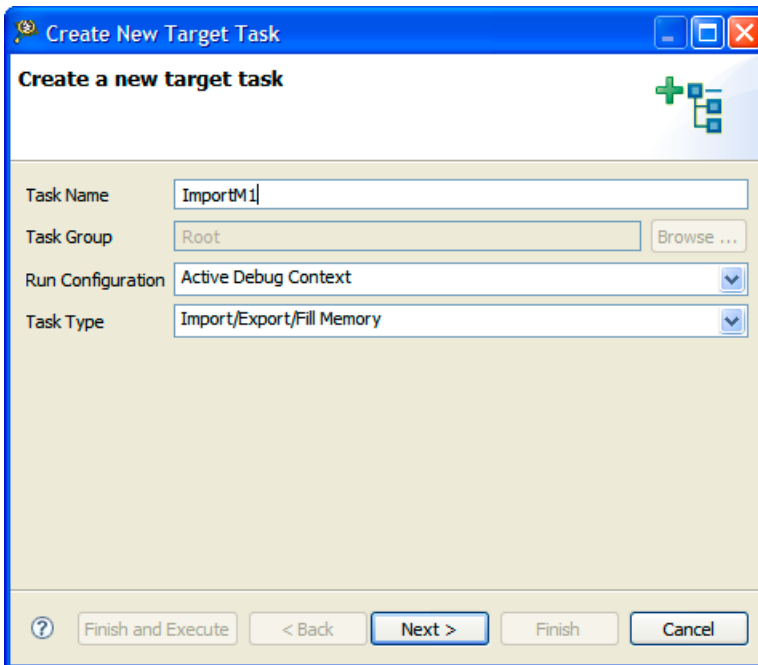
Import/Export/Fill Memory

To create a task:

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 ([Figure 2.52](#)) appears.

Figure 2.52 Create New target Task Window



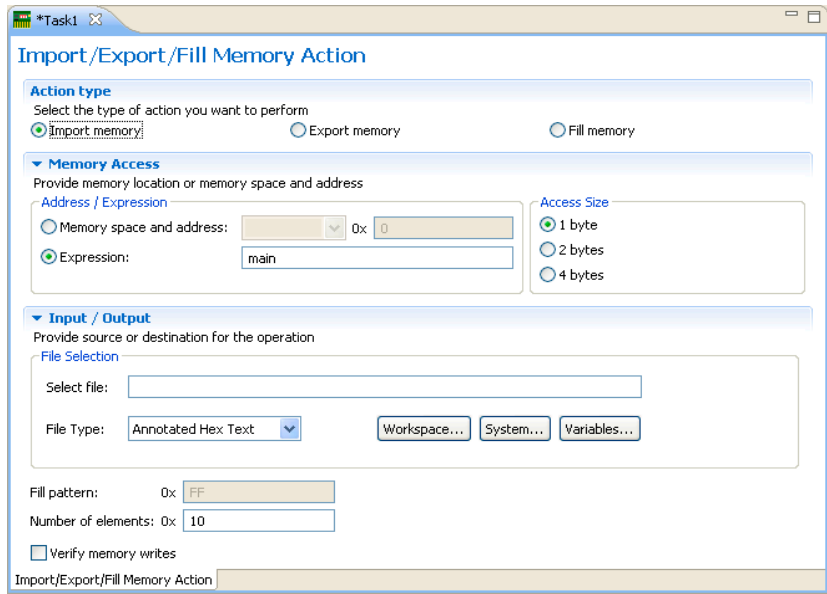
2. In the **Task Name** text box, enter a name for the new task.
3. Use the **Run Configuration** drop-down list to specify the configuration that the task launches and uses to connect to the target.

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 ([Figure 2.53](#)) appears.

Figure 2.53 Configure Import/Export Memory Action Editor



Importing Data from a File into Memory

Select the **Import memory** option from the **Import/Export/Fill Memory Action** editor window (Figure 2.54) to import the encoded data from a user specified file, decode it, and copy it into a user specified memory range. Table 2.8 explains the import memory options.

IDE Extensions

Import/Export/Fill Memory

Figure 2.54 Import Memory

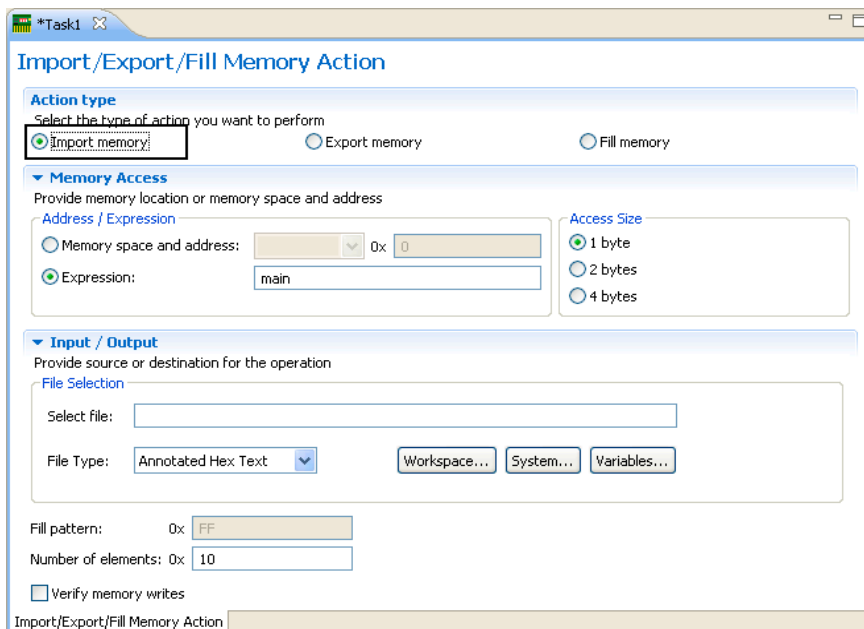


Table 2.8 Import Data from a File into Memory Window items

Item	Explanation
Memory space and address	Enter the literal address and memory space on which the data transfer is performed. The Literal address field allows only decimal and hexadecimal values.
Expression	Enter the memory address or expression at which the data transfer starts.
Access Size	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.

Table 2.8 Import Data from a File into Memory Window items

Item	Explanation
File Type	Defines the format in which the imported data is encoded. By default, the following file types are supported: <ul style="list-style-type: none"> • Signed decimal Text • Unsigned decimal Text • Motorola S-Record format • Hex Text • Annotated Hex Text • Raw Binary
Select file	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. 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.
Number of Elements	Enter the total number of elements to be transferred.
Verify Memory Writes	Check the option to verify success of each data write to the memory.

Exporting Memory Contents to a File

Select the **Export memory** option from the **Import/Export/Fill Memory Action** editor window ([Figure 2.55](#)) 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 2.9](#) explains the export memory options.

IDE Extensions

Import/Export/Fill Memory

Figure 2.55 Export Memory

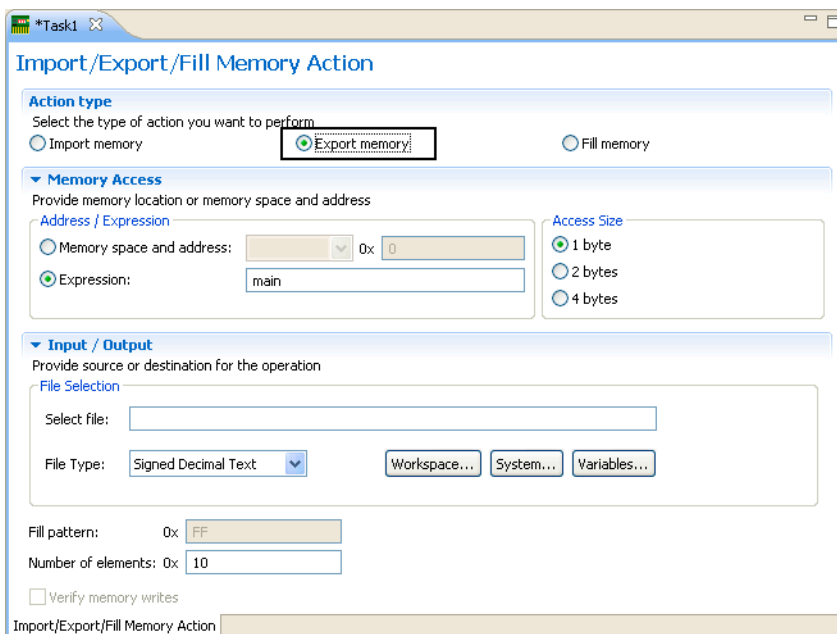


Table 2.9 Export Data from Memory into a File Window Items

Item	Explanation
Memory space and address	Enter the literal address and memory space on which the data transfer is performed. The Literal address field allows only decimal and hexadecimal values.
Expression	Enter the memory address or expression at which the data transfer starts.
Access Size	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.

Table 2.9 Export Data from Memory into a File Window Items

Item	Explanation
File Type	Defines the format in which encoded data is exported. By default, the following file types are supported: <ul style="list-style-type: none"> • Signed decimal Text • Unsigned decimal Text • Motorola S-Record format • Hex Text • Annotated Hex Text • Raw Binary
Select file	Enter the path of the file to write data. Click the Workspace button to select a file from the current project workspace. 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..
Number of Elements	Enter the total number of elements to be transferred.

Fill Memory with a Data Pattern

Select the **Fill memory** option from the **Import/Export/Fill Memory Action** editor window ([Figure 2.56](#)) to fill a user specified memory range with a user specified data pattern. [Table 2.10](#) explains the fill memory options.

IDE Extensions

Import/Export/Fill Memory

Figure 2.56 Fill Memory

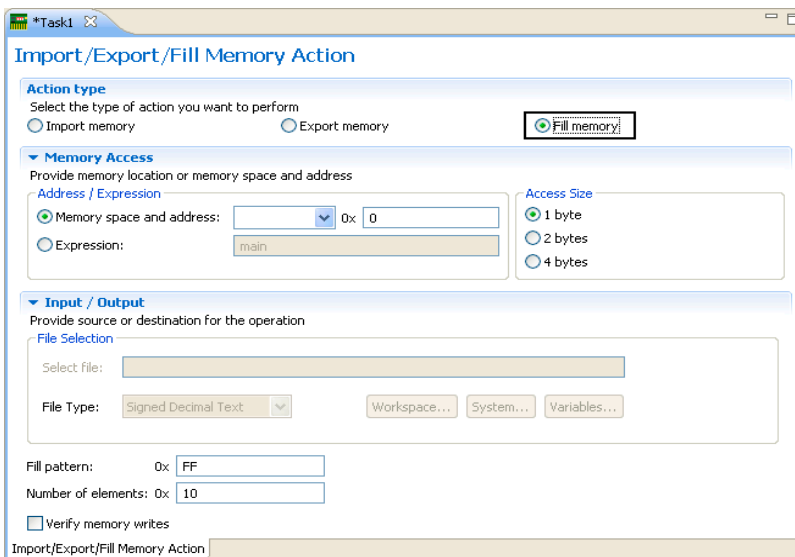


Table 2.10 Fill Memory With a Data Pattern Window items

Item	Explanation
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.

Table 2.10 Fill Memory With a Data Pattern Window items

Item	Explanation
Number of Elements	Enter the total number of elements to be modified.
Verify Memory Writes	Check the option to verify success of each data write to the memory.

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.

[Table 2.11](#) lists and defines the key mappings for Classic IDE and Eclipse IDE.

Table 2.11 Key Mappings - Classic IDE and Eclipse IDE

behaviour	Classic IDE	Eclipse IDE
New	Ctrl + Shift + N	Ctrl + N
Open Open Path (Eclipse IDE)	Ctrl + O	Ctrl + Shift + A
Close	Ctrl + W	Ctrl + F4
Close All	Ctrl + Shift + W	Ctrl + Shift + W Ctrl + Shift + F4
Save	Ctrl + S	Ctrl + S
Save All	Ctrl + Shift + S	Ctrl + Shift + S
Print	Ctrl + P	Ctrl + P

IDE Extensions

Key Mappings

Table 2.11 Key Mappings - Classic IDE and Eclipse IDE

behaviour	Classic IDE	Eclipse IDE
Undo	Ctrl + Z Alt + Backspace	Ctrl + Z
Redo	Ctrl + Shift + Z	Ctrl + Y
Cut	Ctrl + X Shift + Delete	Ctrl + X Shift + Delete
Copy	Ctrl + C Ctrl + Insert	Ctrl + C Ctrl + Insert
Paste	Ctrl + V Shift + Insert	Ctrl + V Shift + Insert
Delete	Del	Del
Select All	Ctrl + A	Ctrl + A
Find Next	F3	Ctrl + K
Find Previous	Shift + F3	Ctrl + Shift + K
Go to Line	Ctrl + G	Ctrl + L
Debug	F5	F11
Run	Ctrl + F5	Ctrl + F11
Step Over	F10	F6
Step Into	F11	F5
Step Out	Shift + F11	F7
Step Return (Eclipse IDE)		
Enable/Disable Breakpoint	Ctrl + F9	Ctrl + Shift + B
Toggle Breakpoint (Eclipse IDE)		
Make	F7	Ctrl + B
Build All (Eclipse IDE)		

Table 2.11 Key Mappings - Classic IDE and Eclipse IDE

behaviour	Classic IDE	Eclipse IDE
Move Line Up	Up	Alt + Up
Move Line Down	Down	Alt + Down

Linker Command File Navigation

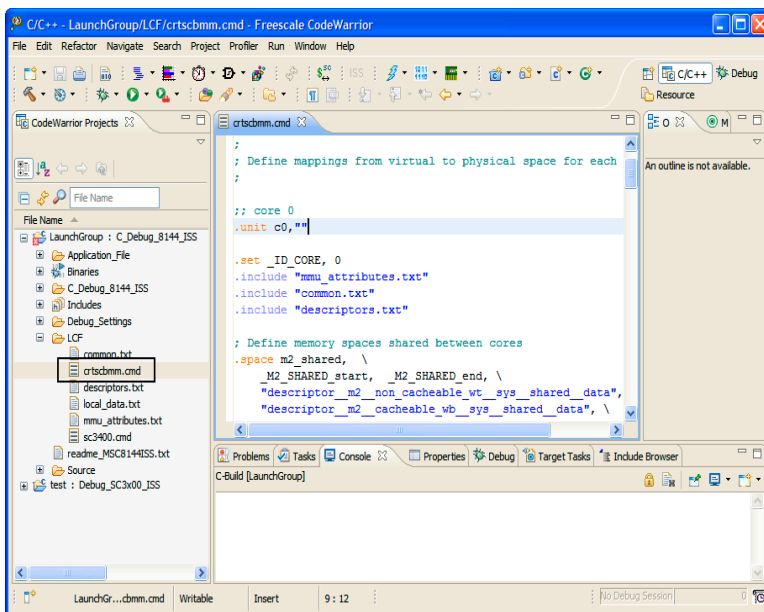
The linker command file (LCF) navigation feature enables you to click on strings containing path names in the CodeWarrior Editor and navigate to the specified file.

NOTE The CodeWarrior Editor recognizes the files with `.lcf`, `.cmd`, and `.l3k` 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 ([Figure 2.57](#)).

Figure 2.57 LCF File

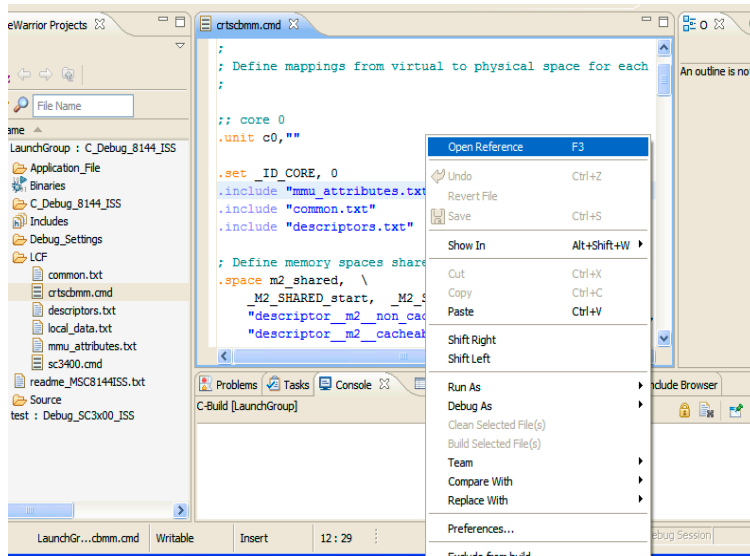


IDE Extensions

Memory Management Unit Configurator

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 ([Figure 2.58](#)).

Figure 2.58 Open Reference Option



The referenced text file opens in the Editor window.

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
- configure MMU program memory-address-translation properties

- configure MMU data memory-address-translation properties
- display the current contents of each register
- write the displayed contents from the MMU configurator to the MMU registers
- save to a file (in a format that you specify) the displayed contents of the MMU configurator

This chapter has these sections:

- [Creating an MMU Configuration](#)
- [Saving MMU Configurator Settings](#)
- [MMU Configurator Toolbar](#)
- [MMU Configurator Pages](#)
- [Opening the MMU Configurator View](#)

Creating an MMU Configuration

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

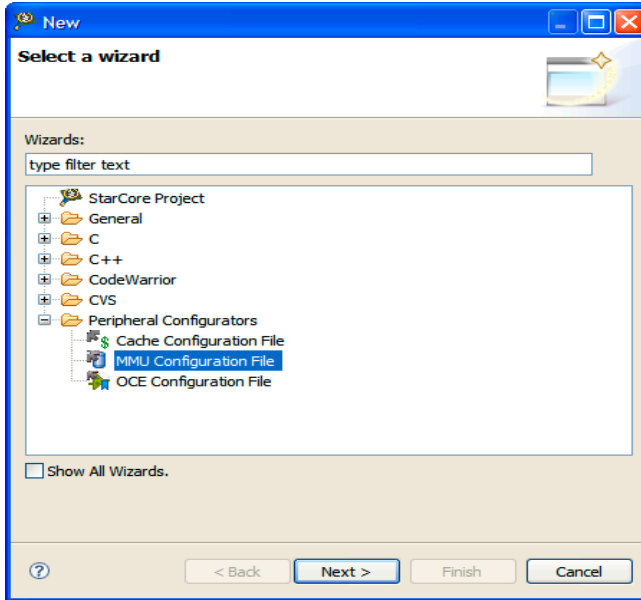
1. From the main menu bar, select **File > New > Other**.

The New window ([Figure 2.59](#)) appears.

IDE Extensions

Memory Management Unit Configurator

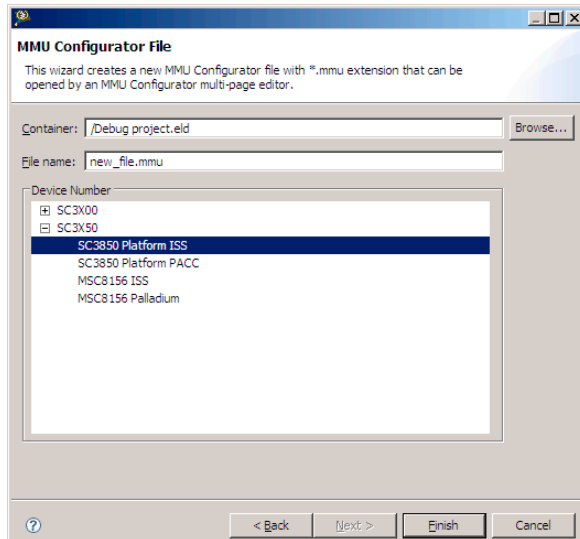
Figure 2.59 New Window—MMU Configuration



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

The **MMU Configurator File** page ([Figure 2.60](#)) appears.

Figure 2.60 MMU Configurator File Page



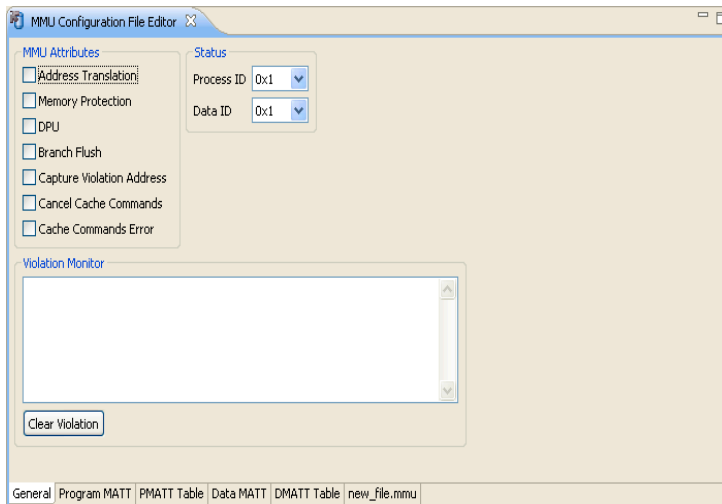
5. Enter in the **Container** text box the path to the directory in which you want to store the MMU configuration. Alternatively, click **Browse**, then use the resulting dialog box to specify the directory.
6. Enter in the **File name** text box a name for the MMU configuration. Alternatively, leave the default name intact.

NOTE If you enter a new name, make sure to preserve the `.oce` 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 ([Figure 2.61](#)).

Figure 2.61 MMU Configuration File Editor View



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.

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. [Table 2.12](#) explains each toolbar button.

Table 2.12 MMU Configurator Toolbar Buttons

Name	Icon	Explanation
Save C Source		Saves to a file the generated code shown in the C page .
Save ASM Source		Saves to a file the generated code shown in the ASM page
Save TCL Source		Saves to a file the generated code shown in the TCL page
Read Target Registers		Updates the content of each MMU configurator page to reflect the current values of the target hardware's registers.
Write Target Registers		Writes to the target hardware's registers the modified contents of all configuration pages of the MMU configurator. You must click this button, or use the corresponding toolbar menu command, in order to write the MMU configurator modifications to the target hardware's registers.

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.

General Page

Use the **General** page ([Figure 2.62](#)) to configure the overall MMU properties. [Table 2.13](#) explains options on the **General** page.

IDE Extensions

Memory Management Unit Configurator

Figure 2.62 MMU Configuration File Editor—General Page

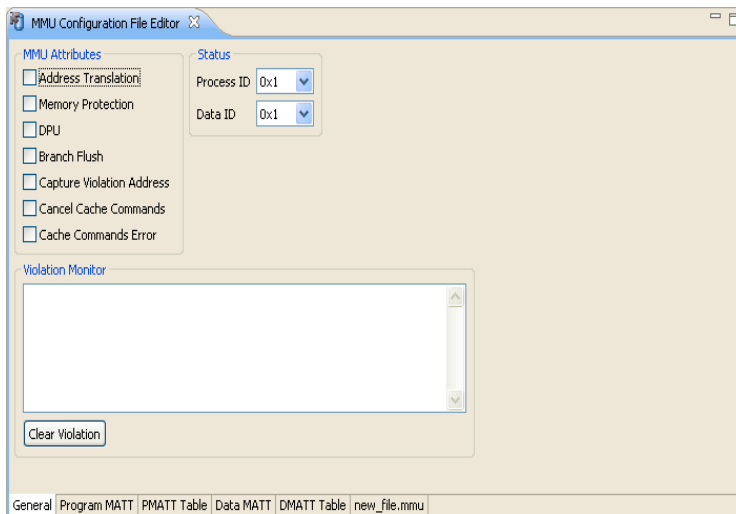


Table 2.13 General Page Settings

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

Table 2.13 General Page Settings

Page Item	Explanation
DPU	<p>Checked—Enables the Debug and Profiling Unit (DPU).</p> <p>Cleared—Disables the DPU. With this option cleared, DPU registers are disabled for read and write accesses.</p> <p>This option corresponds to the Debug and Profiling Unit Enable (DPUE) bit of the MMU Control Register (M_CR).</p>
Branch Flush	<p>Checked—Enables automatic branch-target buffer flush.</p> <p>Cleared—Disables automatic branch-target buffer flush.</p>
Capture Violation Address	<p>Checked—Include the address at which the violation occurred.</p> <p>Cleared—Do not include the address at which the violation occurred.</p>

Program MATT Page

Use the **Program MATT** page ([Figure 2.63](#)) 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.

[Table 2.14](#) explains each option on the **Program MATT** page.

Figure 2.63 MMU Configuration File Editor—Program MATT Page

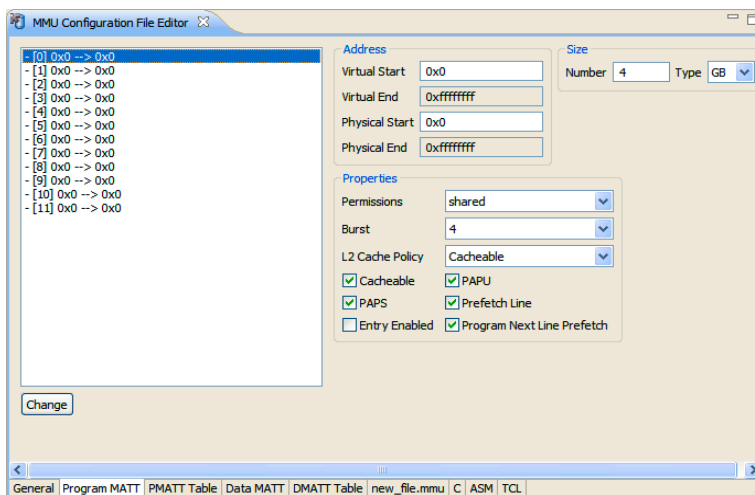


Table 2.14 Program MATT Page Settings

Option	Explanation
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.
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).
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).

Table 2.14 Program MATT Page Settings

Option	Explanation
L2 Cache Policy	<p>Determines the cache policy for the L2 cache for accesses from the core through L1 Instruction cache: Cacheable, NonCacheable, and Reserved.</p> <p>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.</p>
Cacheable	<p>Checked—Enables caching of the segment in instruction cache. Cleared—Disables caching of the segment in instruction cache.</p> <p>This option corresponds to the Instruction Cacheability (IC) bit of the Program Segment Descriptor Registers A (M_PSDAx).</p>
PAPS	<p>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.</p> <p>This option corresponds to the Program Access Permission in Supervisor Level (PAPS) bit of the Program Segment Descriptor Registers A (M_PSDAx).</p>
Entry Enabled	<p>Checked—The MMU enables this mapping entry. Cleared—The MMU disables this mapping entry.</p>
PAPU	<p>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.</p> <p>This option corresponds to the Program Access Permission in User Level (PAPU) bit of the Program Segment Descriptor Registers A (M_PSDAx).</p>

IDE Extensions

Memory Management Unit Configurator

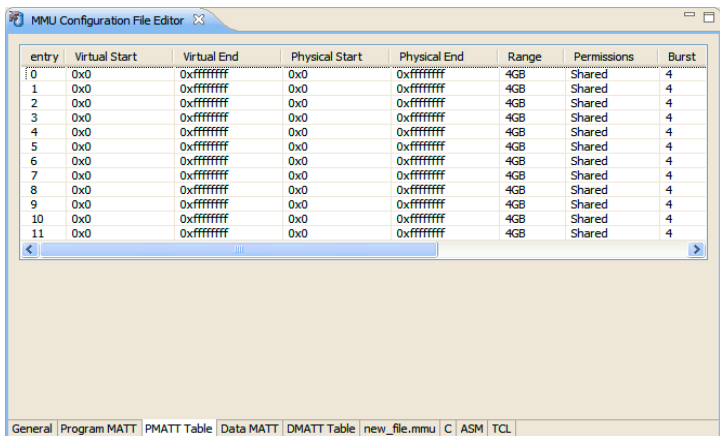
Table 2.14 Program MATT Page Settings

Option	Explanation
Prefetch Line	<p>Checked—Enables the fetch unit’s program-line pre- fetch to a segment cacheable in instruction cache.</p> <p>Cleared—Disables the fetch unit’s program-line prefetch to a segment cacheable in instruction cache.</p> <p>This option corresponds to the Program Pre-fetch Line Enable (PPFE) bit of the Program Segment Descriptor Registers B (M_PSDBx).</p>
Program Next Line Prefetch	<p>Checked—Enables the fetch unit’s program next line pre-fetch mechanism to an ICache cacheable segment.</p> <p>Cleared—Enables the fetch unit’s program next line pre-fetch mechanism to an ICache cacheable segment.</p>

The **PMATT Table** page ([Figure 2.64](#)) 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.

Figure 2.64 MMU Configuration File Editor—PMATT Table Page



Data MATT Page

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.

[Figure 2.65](#) shows the **Data MATT** page. [Table 2.15](#) explains each option on the **Data MATT** page.

Figure 2.65 MMU Configuration File Editor—Data MATT Page

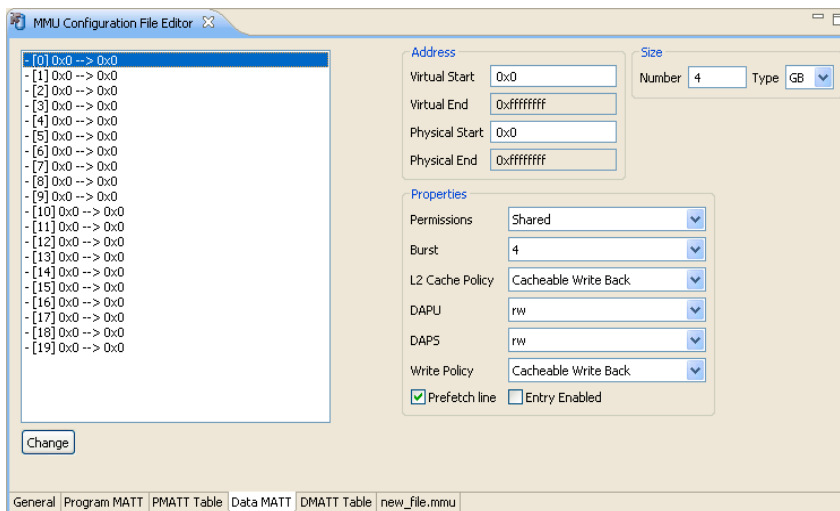


Table 2.15 Data MATT Page Settings

Option	Explanation
Virtual	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.
Physical	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).
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 data segment: shared and non-shared This option corresponds to the Supervisor/Shared Virtual Data Memory (SSVDM) bit of the Data Segment Descriptor Registers A (M_DSDAx).

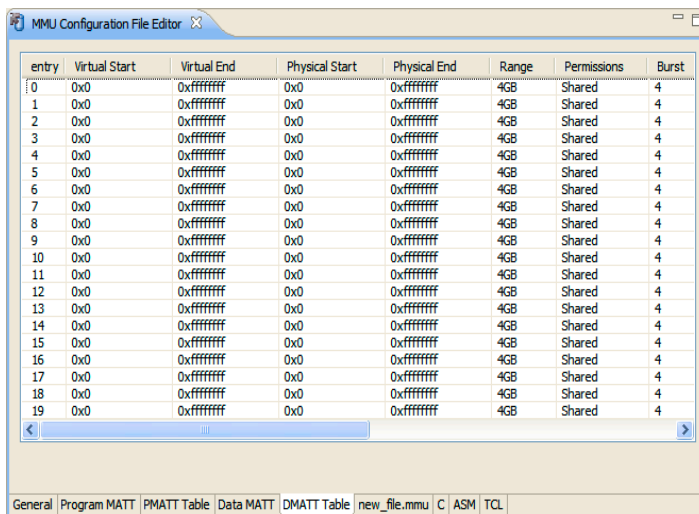
Table 2.15 Data MATT Page Settings

Option	Explanation
Burst	<p>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.</p> <p>This option corresponds to the Data Burst Size (DBS) bits of the Data Segment Descriptor Registers B (M_DSDBx).</p>
L2 Cache Policy	<p>Determines the cache policy for the L2 cache for accesses from the core through L1 Data Cache: Cacheable write through, Cacheable write-back, Non-cacheable, and Adaptive write.</p>
DAPU	<p>Specify whether to allow user-level read (r-), write (-w), both (rw), or neither(--) types of data access.</p> <p>This option corresponds to the Data Access Permission in User Level (DAPU) bits of the Data Segment Descriptor Registers A (M_DSDAx).</p>
DAPS	<p>Specify whether to allow supervisor-level read (r-), write (-w), both (rw), or neither (--) types of data access.</p> <p>This option corresponds to the Data Access Permission in Supervisor Level (DAPS) bits of the Data Segment Descriptor Registers A (M_DSDAx).</p>
Write Policy	<p>Specify the policy to use for data writes and cache:</p> <ul style="list-style-type: none"> • Cacheable write through—Writes are buffered in the writequeue (WRQ) and goes both to the cache and to 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 goes 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 goes through the write through buffer (WTB) to the higher-level memory • Non-cacheable write-through destructive area—writes are buffered in the WRQ and goes through the write through buffer (WTB) to the higher-level memory. Speculative read accesses are blocked in the platform level and does not goes to a higher level memory.

Table 2.15 Data MATT Page Settings

Option	Explanation
Prefetch Line	<p>Checked—Enables the fetch unit’s data-line prefetch to a segment cacheable in data cache.</p> <p>Cleared—Disables the fetch unit’s data-line prefetch to a segment cacheable in data cache.</p> <p>This option corresponds to Data Pre-fetch Line Enable (DPFE) bit of the Data Segment Descriptor Registers B (M_DSDBx).</p>
Entry Enabled	<p>Checked—The MMU enables this mapping entry.</p> <p>Cleared—The MMU disables this mapping entry.</p>

Figure 2.66 MMU Configuration File Editor—Data MATT Table Page



The **DMATT Table** page ([Figure 2.66](#)) 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 changes these settings, use the **Data MATT** page.

Saving MMU Configurator Generated Code

The last four pages of the MMU configurator (C, ASM, TCL, and the page with the name of the saved MMU configuration file) are text-editor pages. These pages contain the generated MMU configuration state, assembly- or C-language source file, or TCL script file. The MMU configurator regenerates these pages when you change settings in the configuration pages, or when you click the **Change** button on the **Program MATT** or **Data MATT** pages.

You can edit the text that appears on a generated-code page. Also, you can copy the text from a generated-code page, then paste that text into a source file. Alternatively, you can save the page's text to a separate file.

[Table 2.16](#) explains the different types of generated code.

Table 2.16 Types of Generated Code

Option	Explanation
MMU configuration file <i>(filename.mmu)</i>	<p>Contains the generated MMU state file. The MMU configurator generates the state file each time you change the MMU configuration. The state file contains target- specific register-state information, as well as Family and Target Device Number state data that you specified in the wizard you used to create the configuration.</p> <p>The MMU configurator uses the state file to re initialize the settings on each page. You can maintain a collection of state files and load the file that initializes settings for a particular set of mappings.</p>
C	<p>Contains the generated C-language code. This generated code is unique for different targets.</p>

IDE Extensions

Memory Management Unit Configurator

Table 2.16 Types of Generated Code

Option	Explanation
ASM	Contains the generated assembly-language code. This generated code is unique for different targets.
TCL	<p>A generated TCL script. You can execute this script within the Debugger Shell view, or the Debugger Shell can execute the generated TCL script as an initialization script for the target hardware. This generated script is unique for different targets.</p> <p>For example, if the name of the generated TCL script is <code>new_file.tcl</code>, you can execute that script in the Debugger Shell view by following this process:</p> <ul style="list-style-type: none"> • Enter this command at a Debugger Shell prompt: <code>source new_file.tcl</code> • Execute the TCL <code>proc</code> by name, <code>init_mmu</code>, by entering the <code>proc</code> name <code>init_mmu</code> on the Debugger Shell command line.

To save to a file the generated code from one of the generated-code pages of the MMU configurator:

1. From the main menu bar, select the menu command to save the corresponding generated code:
 - Select **MMU Editor > Save C** to save the generated C code.
 - Select **MMU Editor > Save ASM** to save the generated assembly language code.
 - Select **MMU Editor > Save TCL** to save the generated TCL script code.

Alternatively, click the corresponding toolbar buttons in the MMU configurator toolbar.

A standard **Save** dialog box appears.

2. Use the dialog box to save the generated code to a file.

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.

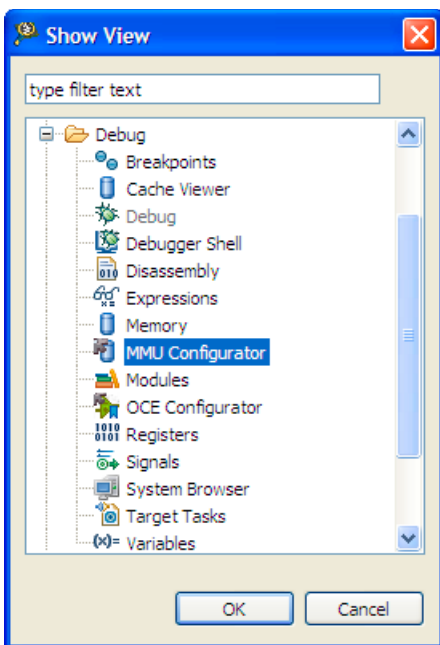
NOTE Because the MMU configurator view does not have an associated configuration file initially, the MMU tab appears in place of the tab that shows the name of the configuration file. Saving the MMU configurator view settings to a file (by selecting **File > Save**) replaces the MMU tab with the name 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 ([Figure 2.67](#)) appears.

Figure 2.67 Show View Dialog Box — MMU Configurator



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.

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 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([Figure 2.68](#)).

Figure 2.68 Properties for <Project> Dialog Box

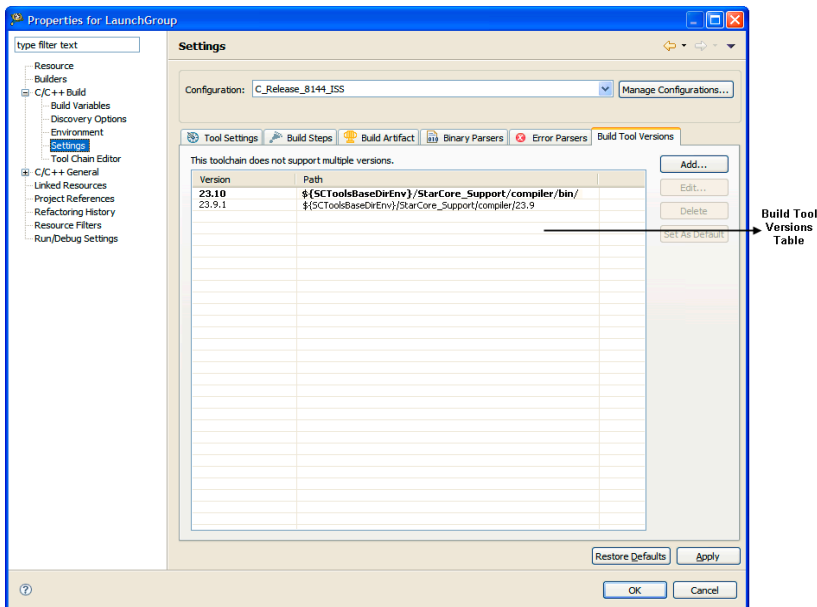


Table 2.2 lists and defines the **Build Tool Versions** panel controls.

Table 2.17 Build Tool Versions Panel Controls

Control	Description
Build Tool Versions table	Lists multiple toolchain versions.
Add button	Enables you to add a new toolchain version.
Edit button	Enables you to edit the currently selected toolchain version.
Delete button	Enables you delete a toolchain version.
Set as Default button	Enable you to set the currently selected toolchain versions as default toolchain version for building projects.
Restore Defaults button	Enables you to restore default toolchain version.

NOTE The default toolchain version is highlighted in **bold** letters in the Build Tool Versions table.

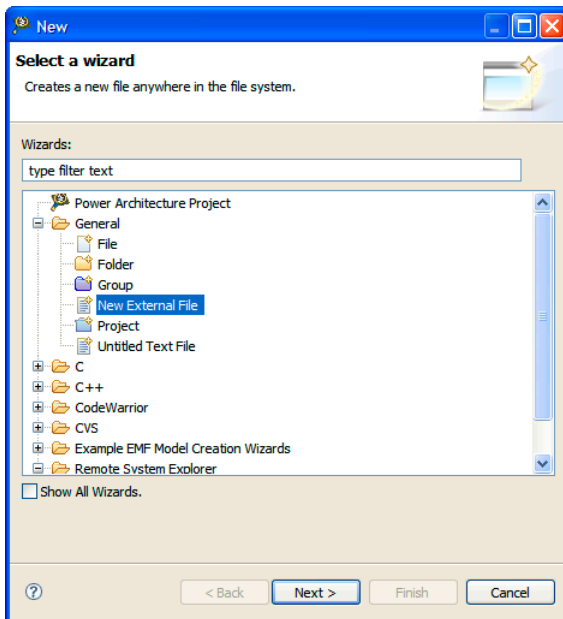
5. Click **Apply**.
6. Click **OK**.

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 ([Figure 2.69](#)) appears.

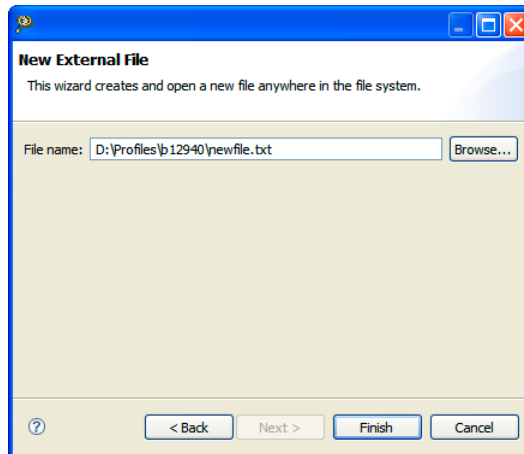
Figure 2.69 New — Select a Wizard Page



2. Select **New External File** under the **General** category.
3. Click **Next**.

The **New External File** page ([Figure 2.70](#)) appears.

Figure 2.70 New External File Page

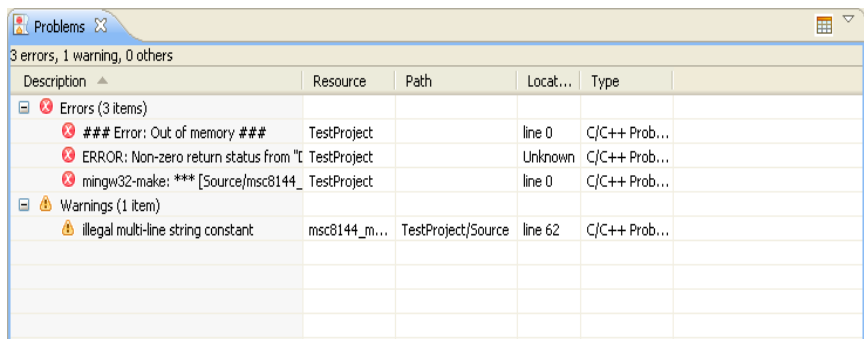


4. Specify the path and filename.
5. Click **Finish**.
IDE opens the file in a new editor window in the **Editor** view.

Problems View

The **Problems** view ([Figure 2.71](#)) 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 a error/warning to go to the location in the source where the error/warning was generated from.

Figure 2.71 Problems View

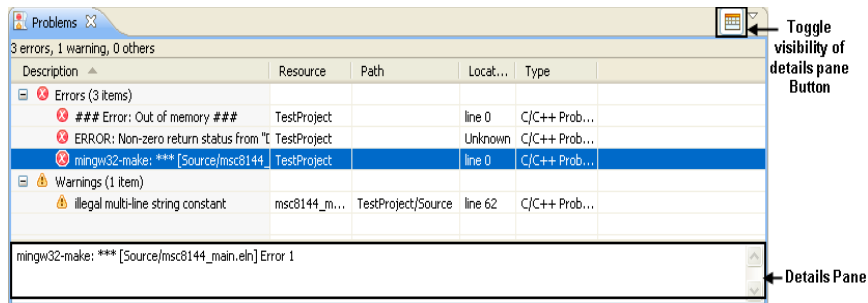


IDE Extensions

Target Management via Remote System Explorer

Clicking the Toggle visibility of the details pane button in the **Problems** view displays the Details pane. The Details pane displays the full description for a selected error/warning.

Figure 2.72 Problems View — Details Pane



Target Management via Remote System Explorer

A remote system is a system configuration that defines connection, initialization, and target parameters. The remote system explorer provides data models and frameworks to configure and manage remote systems, their connections, and their services.

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

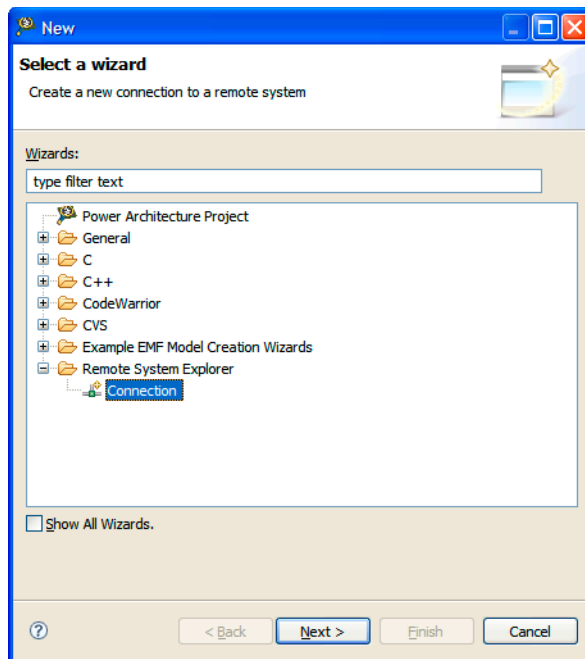
Creating Remote System

To create a remote system:

1. Click **File > New > Other**.

The **New** wizard appears.

Figure 2.73 New — Select a Wizard Page



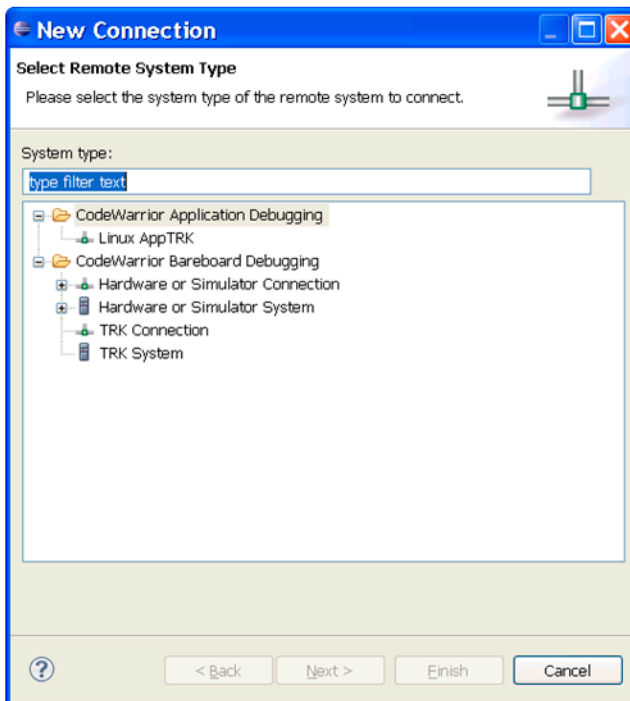
2. Select **Connection** under the **Remote System Explorer** category.
3. Click **Next**.

The **New Connection** page ([Figure 2.74](#)) appears.

IDE Extensions

Target Management via Remote System Explorer

Figure 2.74 New Connection Page



4. Expand **CodeWarrior Bareboard Debugging** and select a remote system type.

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

- **Hardware or Simulator Connection** — Connection configuration for a hardware-based or simulated system. For more information, see [Creating Hardware or Simulator Connection Configuration](#).
- **Hardware or Simulator System** — System configuration for a hardware-based or simulated system. For more information, see [Creating Hardware or Simulator System Configuration](#).
- **TRK System** — System configuration for a system running the TRK debug agent. For more information, see [Creating TRK System Configuration](#).

5. Click **Next**.

The new configuration settings appear. You need to specify configuration settings depending upon the remote system type selected in the **New Connection** page ([Figure 2.74](#)).

6. Click **Finish**.

Creating Hardware or Simulator System Configuration

A hardware or simulator remote system 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.

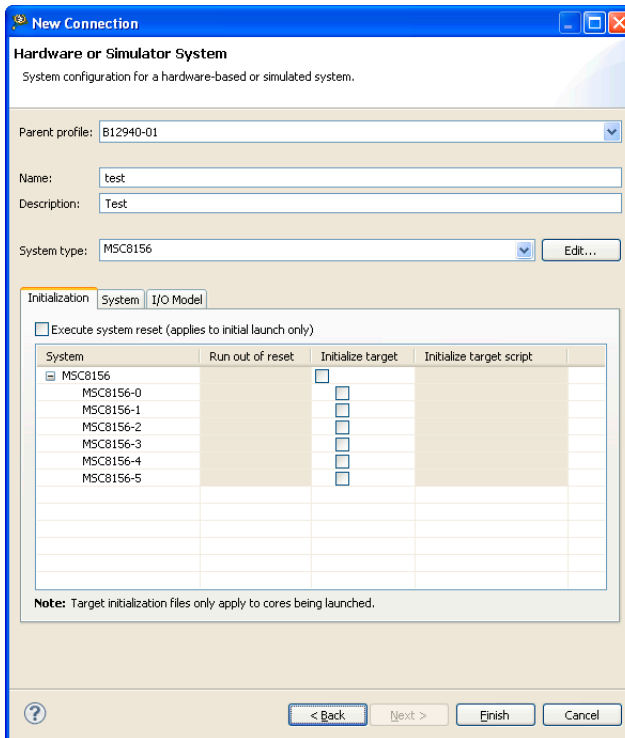
2. Select **Connection** under the **Remote System Explorer** category.
3. Click **Next**.

The **New Connection** page appears.

4. Select **Hardware or Simulator System** from the **CodeWarrior Bareboard Debugging** category.

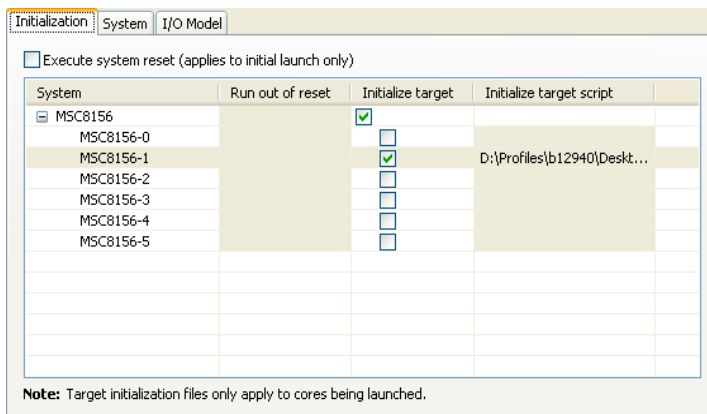
The **New Connection — Hardware or Simulator System** page ([Figure 2.75](#)) appears.

Figure 2.75 New Connection — Hardware or Simulator System Configuration



5. Select a parent profile from the **Parent Profile** drop-down list.
6. Type a configuration name in the **Name** text box.
7. Type system description in the **Description** text box.
8. Select a system configuration from the **System** drop-down list.
A system 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.
9. Click **Edit** to edit the current system configuration. Click **New** to create a new system configuration. For more information, see [Editing a System Type](#).
10. Click the **Initialization** tab.
The initialization settings page ([Figure 2.76](#)) appears.

Figure 2.76 Hardware or Simulator System Configuration — Initialization Settings Page

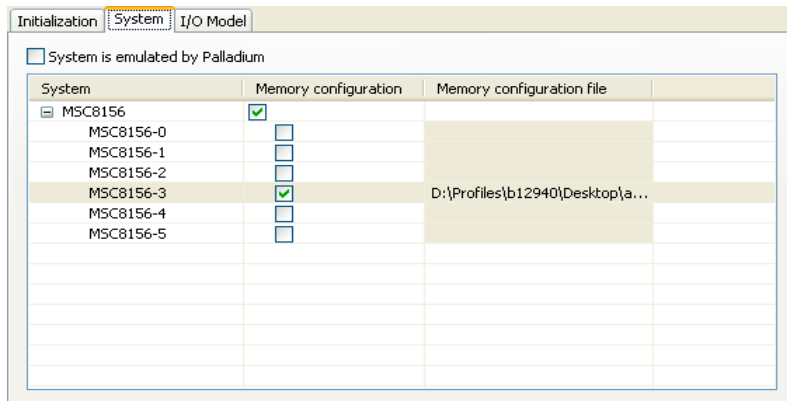


11. Specify the initialization settings to suit your needs.

12. Click the **System** tab.

The system settings page ([Figure 2.77](#)) appears.

Figure 2.77 Hardware or Simulator System Configuration — System Settings Page



13. Specify the system settings to suit your needs.

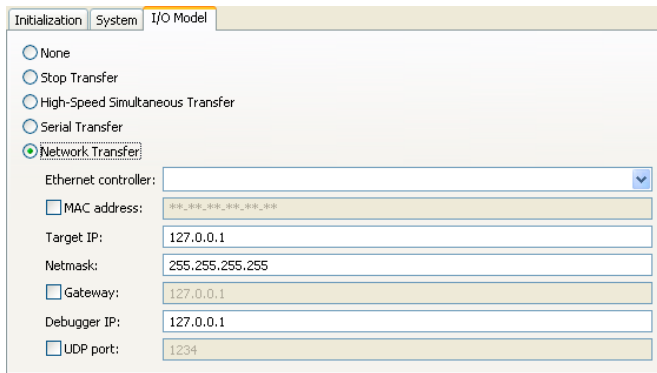
14. Click the **I/O Model** tab.

The **I/O Model** page ([Figure 2.78](#)) appears.

IDE Extensions

Target Management via Remote System Explorer

Figure 2.78 Hardware or Simulator System Configuration — I/O Model Settings Page



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

16. Click **Finish**.

The hardware or simulator system configuration appears in the [Remote Systems View](#).

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 system. 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.

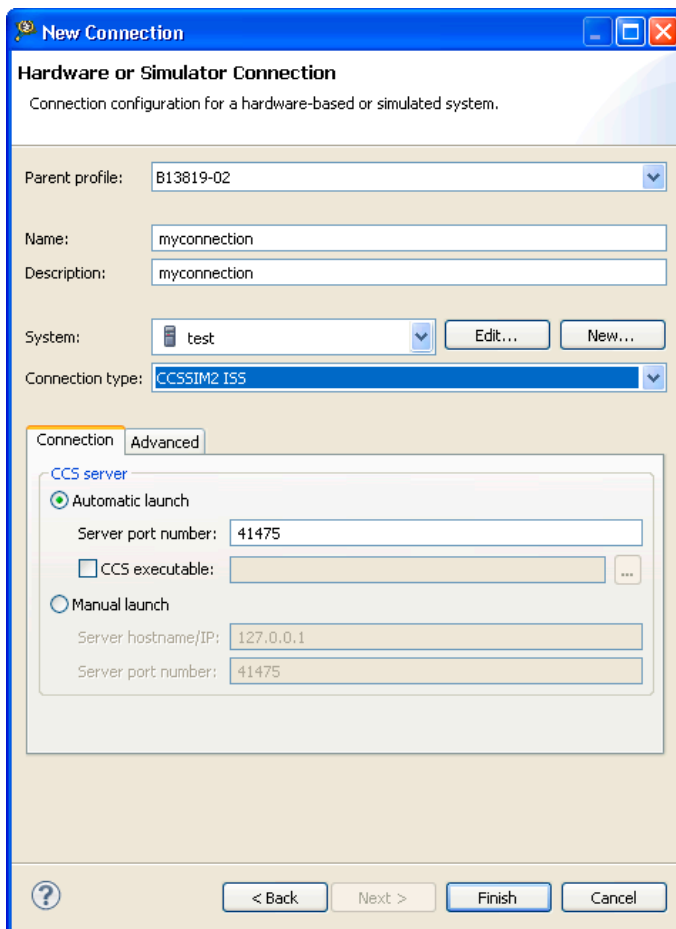
3. Click **Next**.

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 ([Figure 2.75](#)) appears.

Figure 2.79 New Connection — Hardware or Simulator Connection Configuration



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 a remote system type from the **System type** drop-down list.

A system 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.

IDE Extensions

Target Management via Remote System Explorer

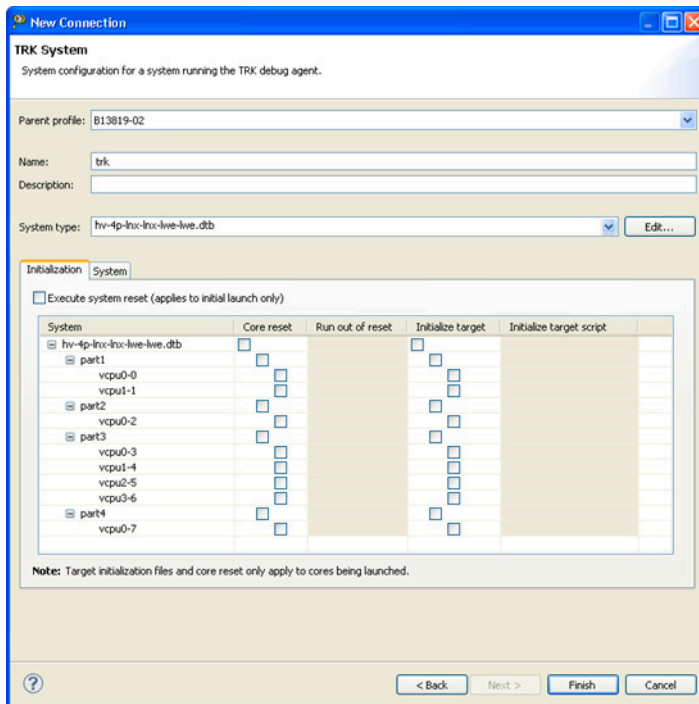
9. Click **Edit** to add or remove system types. For more information, see [Editing a System Type](#).
10. Select a connection type from the **Connection** type drop-down list.
 - Select **CCSSIM2 ISS** to specify setting for CodeWarrior Connection Server (CCS).
 - Select **Ethernet TAP** to specify settings for ethernet TAP connection.
 - Select **USB TAP** to specify settings for USB TAP connection.
11. Click **Finish**.
The hardware or simulator connection configuration appears in the [Remote Systems View](#).

Creating TRK System Configuration

A TRK system enables you to create a system configuration for a system running the TRK debug agent. To create a TRK system 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 System** from the **CodeWarrior Application Debugging** category.
The **New Connection — TRK System** 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 a system type from the **System type** drop-down list or click **Edit** to import system type.
9. Click the **Initialization** tab.
The initialization settings page ([Figure 2.80](#)) appears. Specify the initialization settings to suit your needs.

Figure 2.80 TRK System - New Connection



10. Click the **System** tab.

Specify the system settings to suit your needs.

11. Click **Finish**.

The TRK system configuration appears in the [Remote Systems View](#).

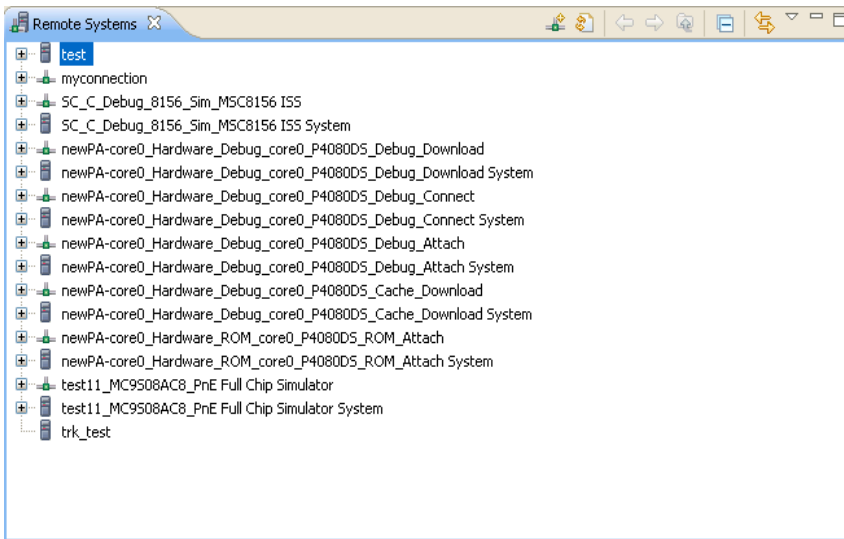
Remote Systems View

Remote Systems view ([Figure 2.81](#)) helps you view and modify remote system settings. It displays various remote system and remote connection configurations. To open the **Remote Systems** view, select **Window > Show View > Remote Systems** from the IDE menu bar.

IDE Extensions

Target Management via Remote System Explorer

Figure 2.81 Remote Systems View



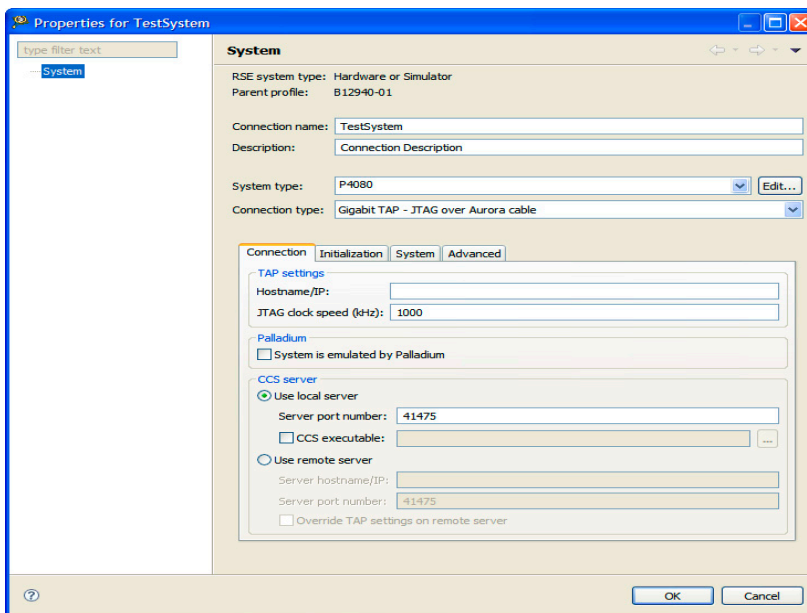
Modifying Remote System

The **Remote System** view enables you to modify settings for a remote system. To change remote system settings:

1. Switch to the **Remote Systems** view.
2. Right-click a remote system name and select **Properties** from the context menu.

The **Properties for <Remote System>** dialog box ([Figure 2.82](#)) appears.

Figure 2.82 Properties for <Remote System> Dialog Box



3. Change the settings in this page to suit your needs.
4. Click **OK**.

The changes are applied to the remote system.

Exporting Remote System

You can export a remote system to an external file. To export a remote system:

1. Switch to the **Remote Systems** view.
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**.

Importing Remote System

You can import a remote system from an external file. To import a remote system:

IDE Extensions

Target Management via Remote System Explorer

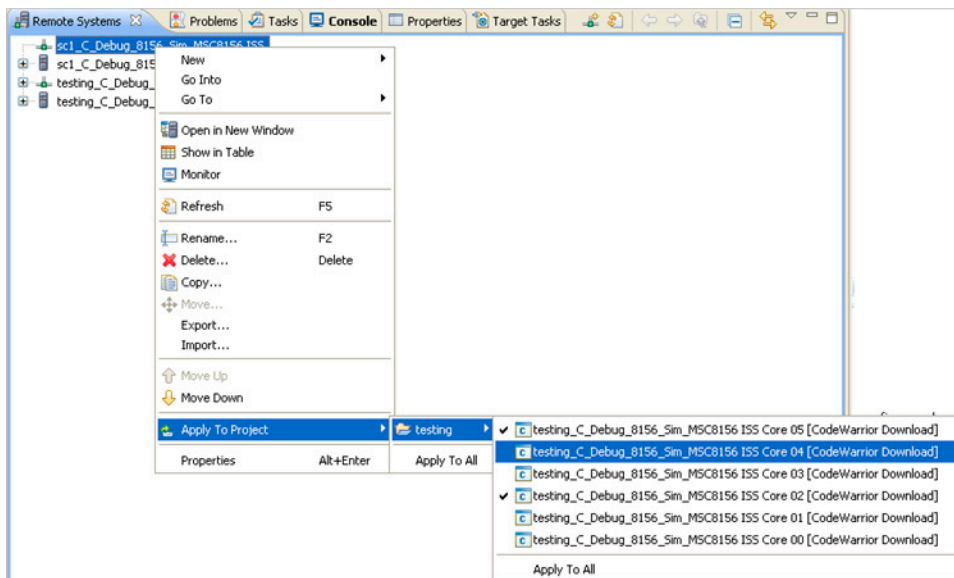
1. Switch to the **Remote Systems** view.
2. Right-click in the view and select **Import** from the context menu.
The **Open** dialog box appears.
3. Select a remote system file.
4. Click **Open**.

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:

1. Switch to the **Remote System** view.
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.

Figure 2.83 Remote System - Apply to Project



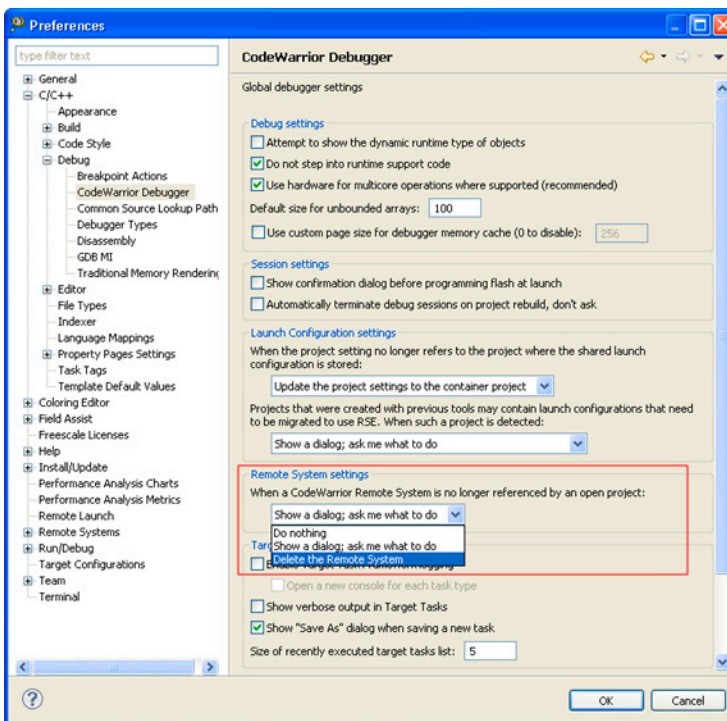
3. Click on the item to apply the target selection in a project ([Figure 2.83](#)).
The projects items are displayed with a check mark to show if the item has the target selected.
4. Click **Apply To All** option to apply the target selection for all projects.

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 ([Figure 2.84](#)).

Figure 2.84 CodeWarrior Debugger - Remote System Settings



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**.

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:

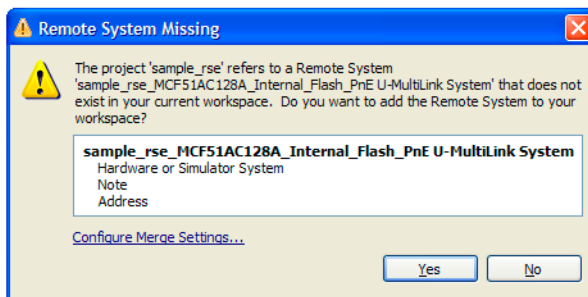
1. [Remote System Missing](#)
2. [Remote System Changed](#)

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.

[Figure 2.85](#) shows **Remote System Missing** dialog box.

Figure 2.85 Remote System Missing dialog box



NOTE By default, the Remote System Missing dialog is not displayed unless the Remote System Merge 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 Merge Settings** link to directly change the Remote System Merge preferences, to avoid automatically displaying this dialog box in future.

Remote System Merge Preferences

You can set preferences for Remote System to handle differing Remote Systems and missing Remote System.

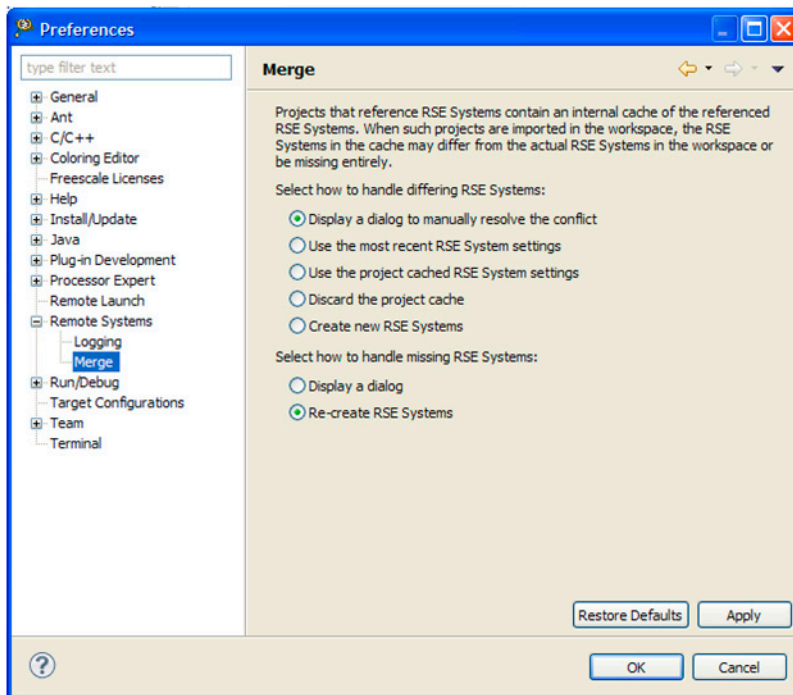
Projects that reference Remote Systems contain an internal cache of the referenced Remote Systems. When such projects are imported in the workspace, the Remote Systems in the cache may differ from the actual Remote Systems in the workspace or be missing entirely.

To configure merge settings:

1. Click **Window > Preferences**.

The **Preferences** dialog box appears.

Figure 2.86 Remote System Merge Preferences dialog box



IDE Extensions

Target Management via Remote System Explorer

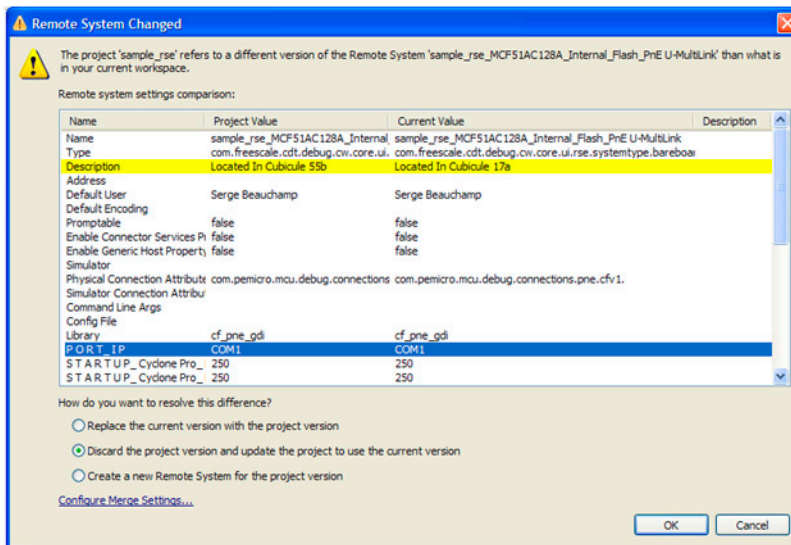
2. Expand the **Remote System** tree control from the left-pane of the **Preferences** dialog box.
You can configure the way the dialog appear by changing the Remote System Merge dialog as shown in [Figure 2.86](#).
3. Select **how to handle differing RSE System** options.
4. Select **how to handle missing RSE Systems** options.
5. Click **OK** to apply changes.

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.

[Figure 2.87](#) shows **Remote System Changed** dialog box.

Figure 2.87 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**.

You can click on the **Configure Merge Settings** link to directly change the Remote System Merge preferences, to avoid automatically displaying this dialog box in future. For details refer [Remote System Merge Preferences](#).

NOTE You may keep the file containing the referenced remote systems `ReferencedRSESystems.xml` in a version control system for future use.

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.

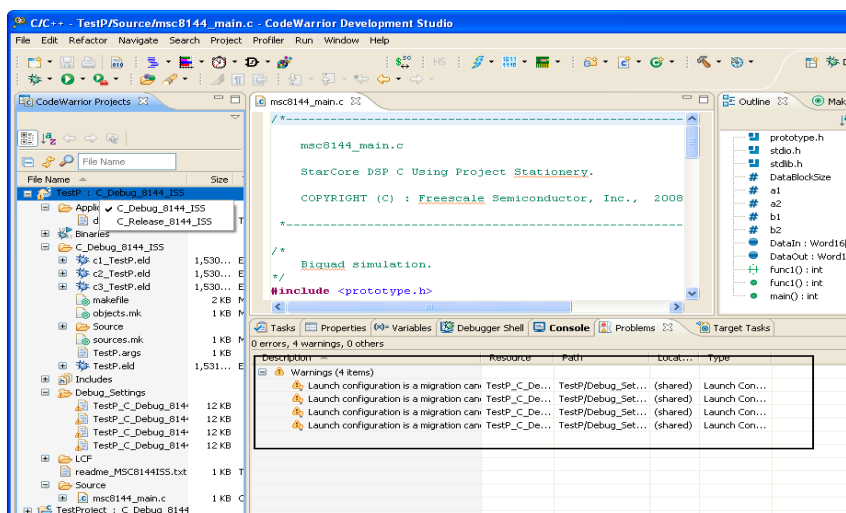
Display of Launch Configurations Needing Migration

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.

IDE Extensions

Target Management via Remote System Explorer

Figure 2.88 Migration Candidates in Problems View



To configure migration preference:

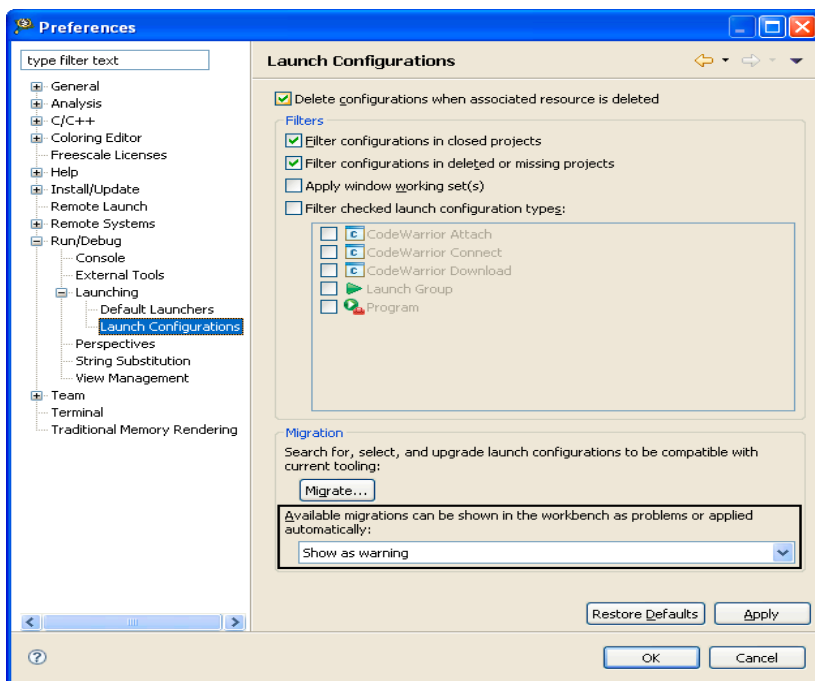
1. Click **Window > Preferences**.

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.

Figure 2.89 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.

Migrating Launch Configurations

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

- [Migration using Smart Migration](#)
- [Migration Using Quick Fix](#)

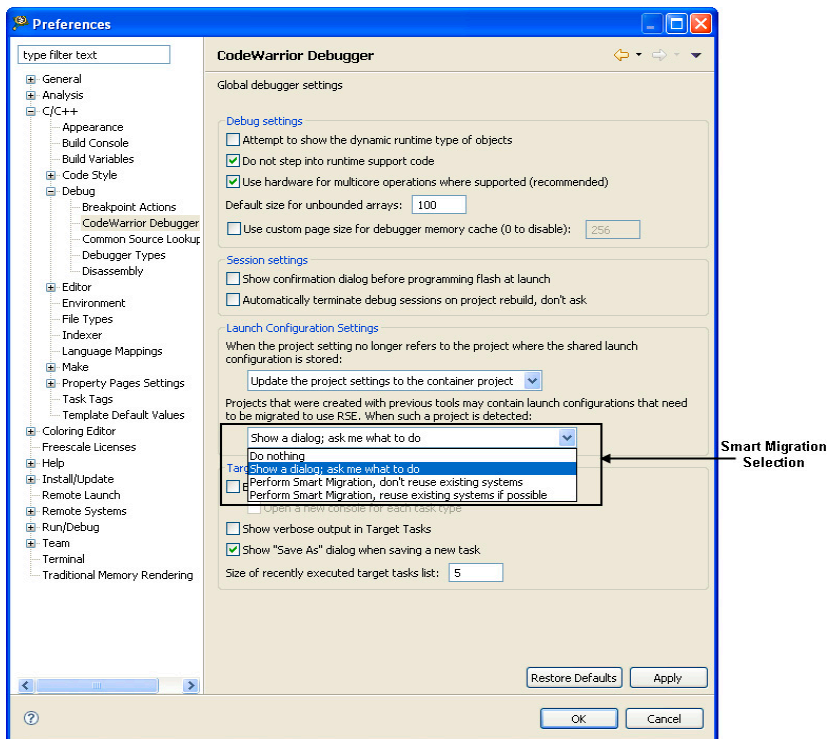
Migration using Smart Migration

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.
3. Select **CodeWarrior Debugger**.

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

Figure 2.90 Configuring Smart Migration



4. Select a migration option from the Smart Migration Selection drop-down list box.
 - 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 2.91](#)) to migrate all migration candidates.

To migrate using Smart Migration

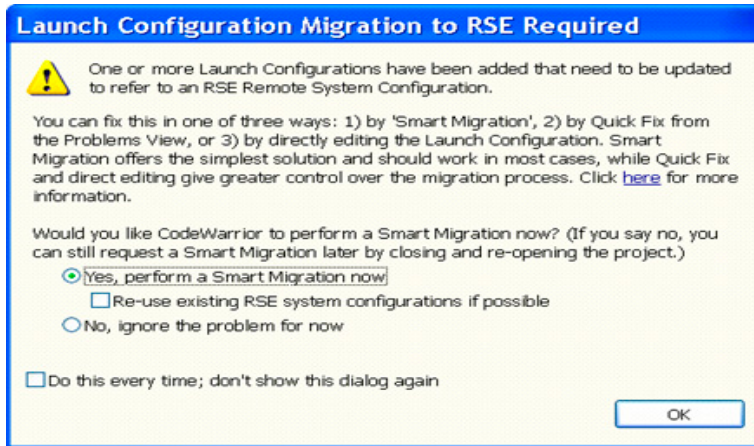
IDE Extensions

Target Management via Remote System Explorer

1. Open the Project in CodeWarrior

The **Launch Configuration Migration to RSE Required** dialog box appears.

Figure 2.91 Launch Configuration Migration using Smart Migration



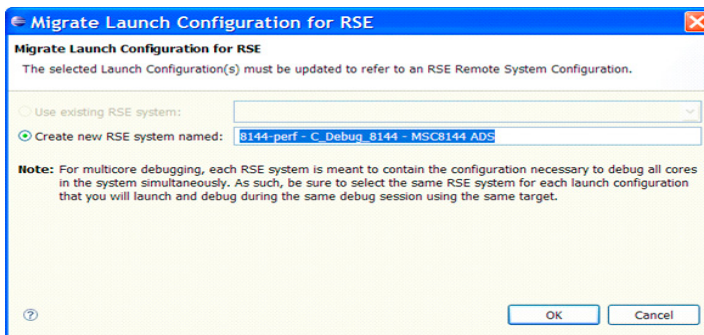
2. Select **Yes, Perform a Smart Migration now** option.
3. Check the **Re-use existing RSE system configuration if possible** option to reuse existing Remote System configurations. Otherwise, Smart Migration migrates launch configurations without re-using existing Remote System configurations.

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.

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 2.92 Migrate Launch Configuration for RSE Dialog Box



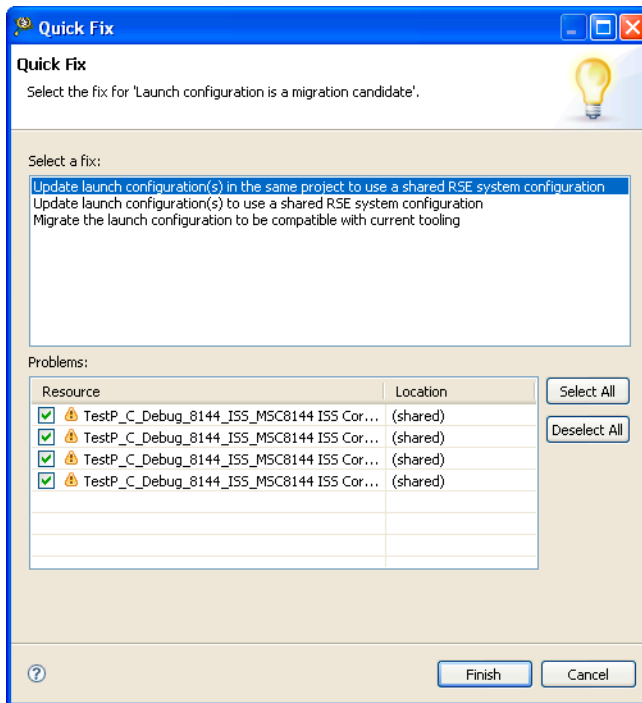
6. Specify a name for the launch configuration in the **Create new RSE system named:** text box.
7. Click **OK**.

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.

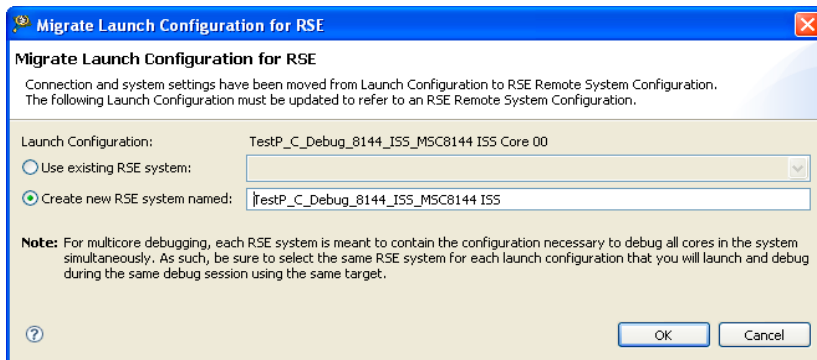
Figure 2.93 Quick Fix Dialog Box



3. Select a fix from the **Select a Fix** list box.
 - Update lunch 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 lunch 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 2.94 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.

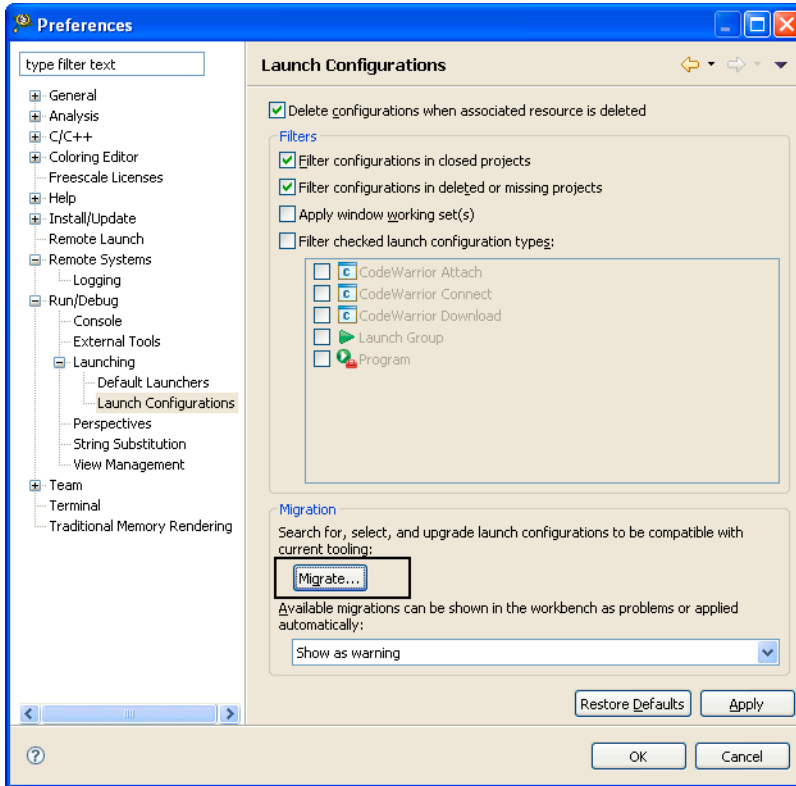
NOTE Remote systems compatible with the selected launch configuration are only listed in this drop-down list box.

- 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.

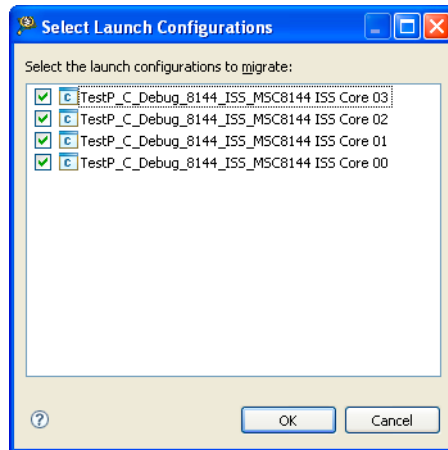
Figure 2.95 Preferences Dialog Box



4. Click **Migrate**.

The **Select Launch Configurations** dialog box appears.

Figure 2.96 Select Launch Configurations Dialog Box



5. Select the launch configurations to migrate.
6. Click **OK**.
The **Migrate Launch Configuration for RSE** dialog box ([Figure 2.94](#)) 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**.

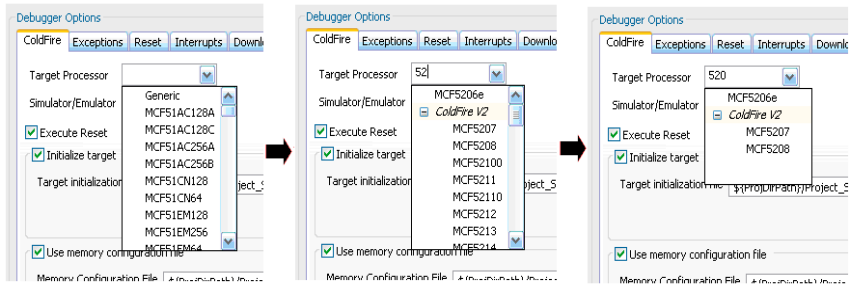
Target Processor Selection

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. [Figure 2.97](#) shows the use of target processor selection combo box.

IDE Extensions

Target Tasks View

Figure 2.97 Target Processor Selection



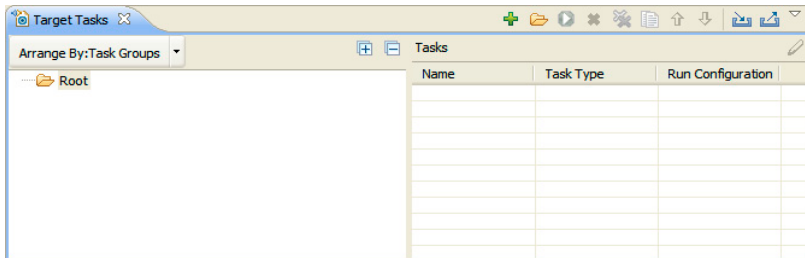
Target Tasks View

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** ([Figure 2.98](#)) appears in the **Debug** perspective.

Figure 2.98 Target Task View



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**.

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**.

Flash File to Target

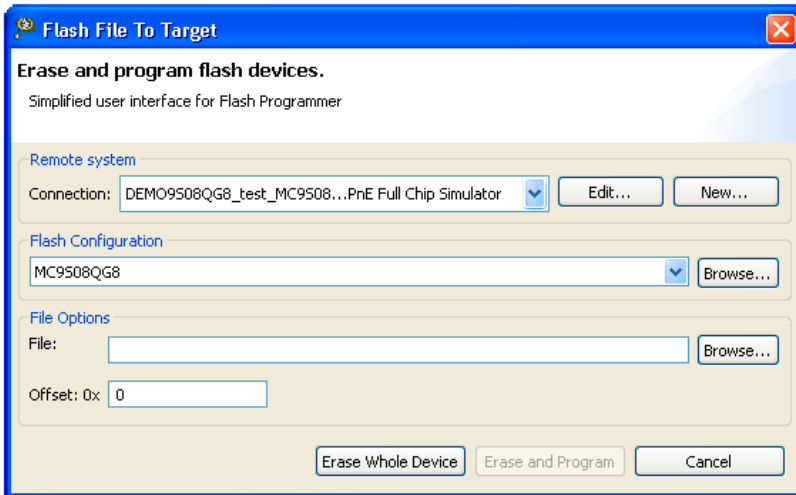
This CodeWarrior feature enables you to perform flash operation. You don't need any project for using **Flash File to Target** feature, only a valid **Remote System** is required. This feature enables you to perform these basic flash operations:

- [Erasing Flash Device](#)
- [Programming a File](#)

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

[Figure 2.99](#) shows the **Flash File to Target** dialog box.

Figure 2.99 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** 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*.
- **File Options** 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.
- **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 [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.

Erasing Flash Device

To erase a flash device, follow these steps:

1. Click the **Flash Programmer** menu button from the IDE toolbar
2. Click the **Flash File to Target** option from the drop-down menu.
The **Flash File to Target** dialog box appears.
3. 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.

4. Select a flash configuration from the **Flash Configuration** drop-down list.
5. Click the **Erase Whole Device** button.

Programming a File

1. Click the **Flash Programmer** menu button from the IDE toolbar
2. Click the **Flash File to Target** option from the drop-down menu.
The **Flash File to Target** dialog box appears.
3. 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.

4. Select a flash configuration from the **Flash Configuration** drop-down list.
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.
7. Click the **Erase and Program** button.

Viewing CodeWarrior Plug-ins

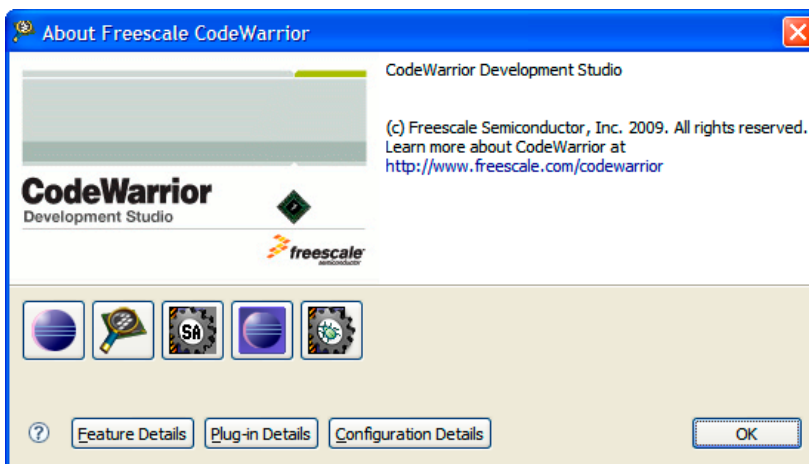
To view the currently installed CodeWarrior features and the associated plug-ins:

1. Select **Help > About Freescale CodeWarrior** from the IDE menu bar.
The **About Freescale CodeWarrior** dialog box ([Figure 2.100](#)) appears.

IDE Extensions

Viewing CodeWarrior Plug-ins

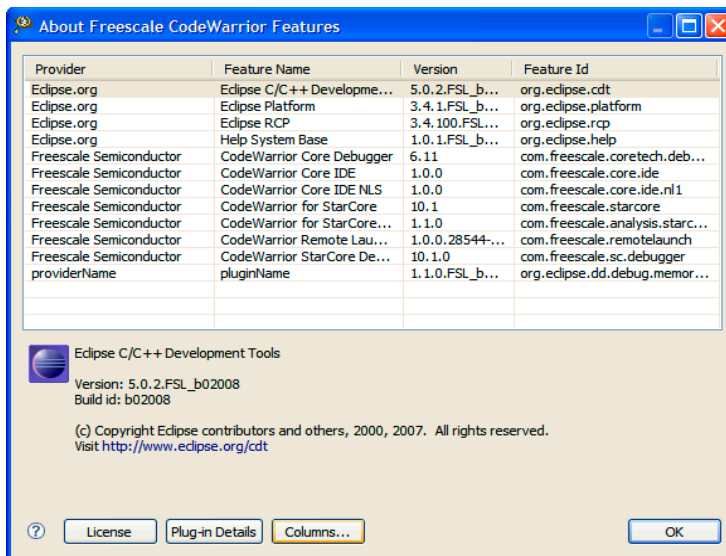
Figure 2.100 About Freescale CodeWarrior Dialog Box



2. Click the **Feature Details** button.

The **About Freescale CodeWarrior Features** dialog box ([Figure 2.101](#)) appears.

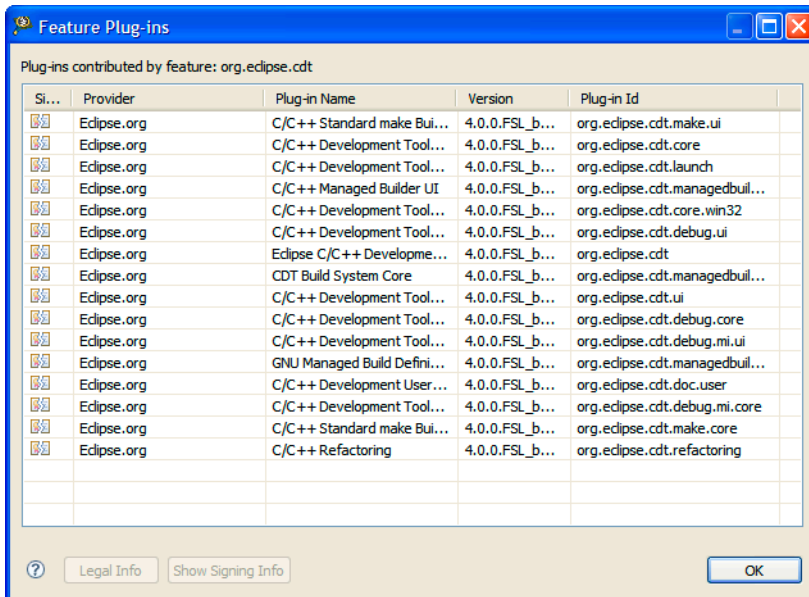
Figure 2.101 About Freescale CodeWarrior Features Dialog Box



3. 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 CodeWarrior Core IDE feature and click the **Plug-in Details** button, the **Feature Plug-ins** dialog box (Figure 2.102) containing the list of plug-ins associated with the CodeWarrior Core IDE feature appears.

Figure 2.102 Feature Plug-ins Dialog Box





IDE Extensions

Viewing CodeWarrior Plug-ins

Debugger

This chapter explains how to work with the debugger to control program execution.

The topics in this chapter are:

- [About Debugger](#)
- [Automated Builds](#)
- [Breakpoints](#)
- [Build While Debugging](#)
- [CodeWarrior Debugger Settings](#)
- [Command-Line Debugger Shell](#)
- [Context Menus](#)
- [Debug Perspective](#)
- [Debug View](#)
- [Disassembly View](#)
- [Environment Variables in Launch Configuration](#)
- [Launch Group](#)
- [Load Multiple Binaries](#)
- [Memory View](#)
- [Memory Browser View](#)
- [Multicore Debugging](#)
- [Multicore Groups](#)
- [Multicore Reset](#)
- [Path Mapping](#)
- [Redirecting Standard Output Streams to Socket](#)
- [Refreshing Data During Run Time](#)
- [Registers View](#)
- [Register Details Pane](#)
- [Remote Launch](#)
- [Stack Crawls](#)

Debugger

About Debugger

- [Symbolics](#)
- [System Browser View](#)
- [Target Connection Lost](#)
- [Target Initialization Files](#)
- [TAP Remote Connections](#)
- [Variables](#)
- [Watchpoints](#)

About Debugger

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,
- manipulate target memory, and
- configure target-hardware subsystems.

Automated Builds

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. Copy `ecd.exe` file from the
`<CWInstallDir>\eclipse\plugins\com.freescale.core.ide.commandLineDriver_2.0.0.FSL_{build_number}` folder to the
`<CWInstallDir>\eclipse\` folder.
2. Invoke the `ecd` command line.

3. 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.

ecd.exe Tool Commands

The `ecd.exe` tool commands are listed below:

build

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

Syntax

```
ecd.exe -build [-verbose] [-cleanAll] [ -project path [  
- config name | -allConfig] -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 out the C/C++ Managed Build project setting(s).

Syntax

```
ecd.exe -getOptions -project path [-config name] [-file path] [-option option-id]
```

Parameters

`-config`

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

`-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).

`-option`

The `option id` of the setting. If the option setting isn't specified, all options will be 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.

`-Option-id`

The `option id` of the setting.

generateMakefiles

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

Syntax

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

Parameters

`-config`

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.

setOptions

Change a C/C++ Managed Build project setting.

Syntax

```
ecd.exe -setOptions -project path [-config name] [-file path] (-set | -prepend | -append | -insert) option-id option-value
```

Parameters

`-config`

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

`-file`

The file 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.

`-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.

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. Supports also leaving the Workbench UI open with the `-noclose` flag.

Syntax

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

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. [Table 3.1](#) defines these states.

Table 3.1 Breakpoint States

State	Icon	Description
Enabled		Indicates that the breakpoint is currently enabled. The debugger halts the program execution at an enabled breakpoint. Click the icon to disable the breakpoint.
Disabled		Indicates that the breakpoint is currently disabled. The debugger does not halt program execution at a disabled breakpoint. Click the icon to enable the breakpoint.

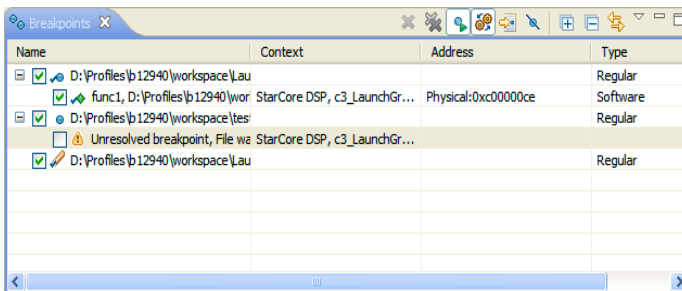
Breakpoints View

The **Breakpoints** view ([Figure 3.1](#)) 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



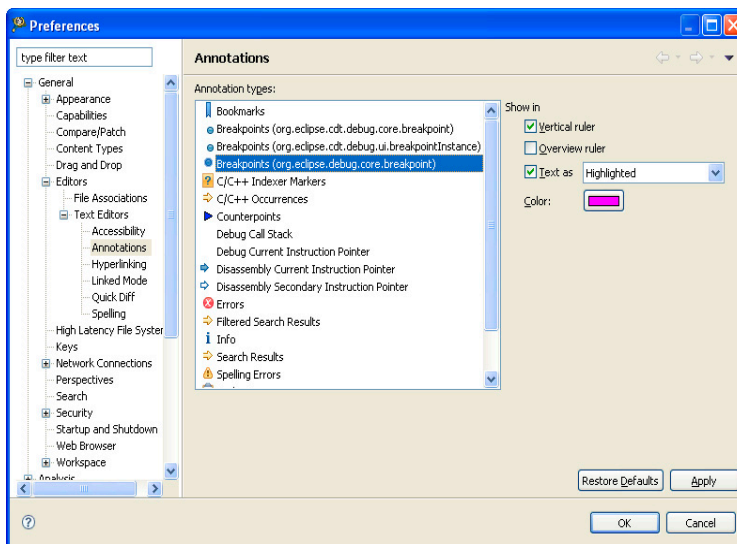
Breakpoint Annotations

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.breapoint`) from the **Annotation types** list box.
4. Specify settings for the selected annotation.
5. Click **Apply**.
6. Click **OK**.

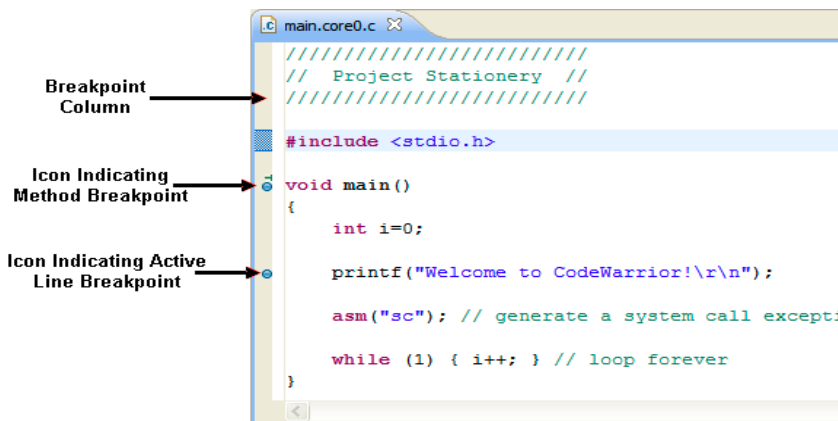
Regular Breakpoints

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

- **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).

[Figure 3.3](#) 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.

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.

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.





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. [Table 3.2](#) describes these states.

Table 3.2 Special Breakpoint States

State	Hardware Icon	Software Icon	Description
Enabled			Indicates that the breakpoint is currently enabled. The debugger halts program execution at an enabled breakpoint. Click the icon to disable the breakpoint.
Disabled			Indicates that the breakpoint is currently disabled. The debugger does not halt program execution at a disabled breakpoint. Click the icon to enable the breakpoint.

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.

Working with Breakpoints

This topic describes the following:

- [Modify Breakpoint Properties](#)
- [Restricting Breakpoints to Selected Targets and Threads](#)
- [Limiting New Breakpoints to Active Debug Context](#)
- [Grouping Breakpoints](#)
- [Disabling Breakpoints](#)
- [Enabling Breakpoints](#)

- [Removing Breakpoints](#)
- [Removing All Breakpoints](#)
- [Skipping All Breakpoints](#)

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.

[Figure 3.4](#) shows the **Properties for** dialog box. [Table 3.4](#) describes each breakpoint property.

Figure 3.4 Properties for Dialog Box

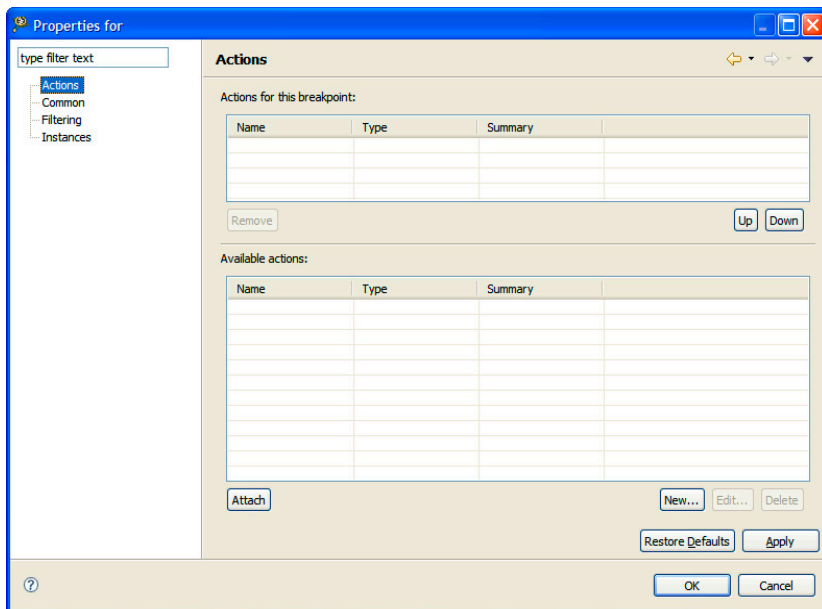


Table 3.3 Breakpoint Properties

Option	Description
Actions	<p>Enables you to attach one or more breakpoint actions to a single breakpoint.</p> <p>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 Actions for this breakpoint table.</p>
Common	<p>Displays common properties of a breakpoint. Additionally, you can define a condition that determines when the breakpoint will be encountered.</p> <p>A condition for a breakpoint can be any logical expression that returns true or false value.</p>
Filtering	<p>Enables you to restrict the breakpoint to the selected targets and threads.</p>
Instances	<p>Displays real-time breakpoint information that helps identify the address and the way a breakpoint is installed on a target.</p>

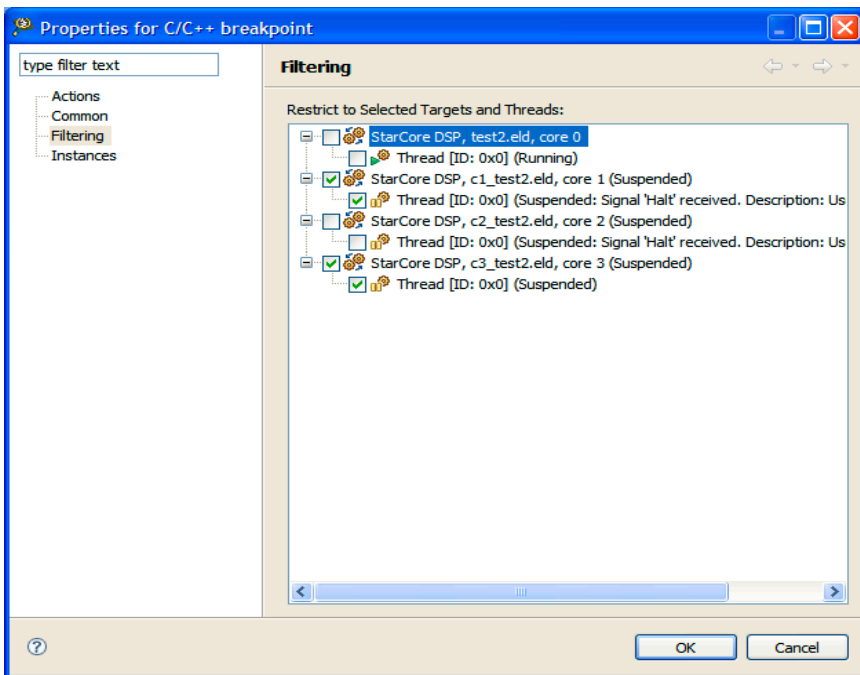
Restricting Breakpoints to Selected Targets and Threads

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** ([Figure 3.5](#)).

Figure 3.5 Restrict Breakpoint to Selected Targets & Threads



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.

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.

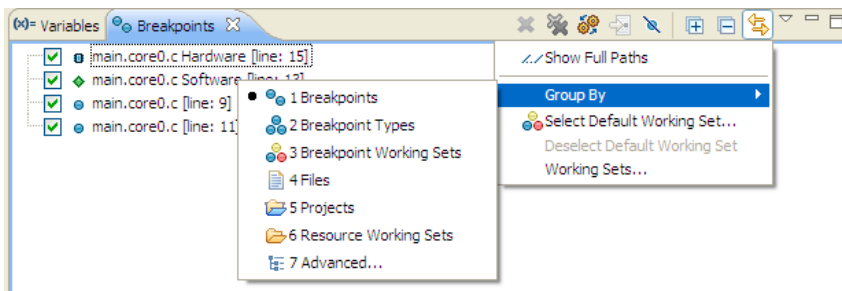
Grouping Breakpoints

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 3.6 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.


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 [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.


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 [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.

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.


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.

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.

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.

[Table 3.4](#) lists and describes breakpoint actions.

Table 3.4 Breakpoint Actions

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

Breakpoint Actions Preferences Page

You use the **Breakpoint Actions** preferences page ([Figure 3.7](#)) to manage a global list of breakpoint actions available in a workspace. Each action is an instance of a specific action type.

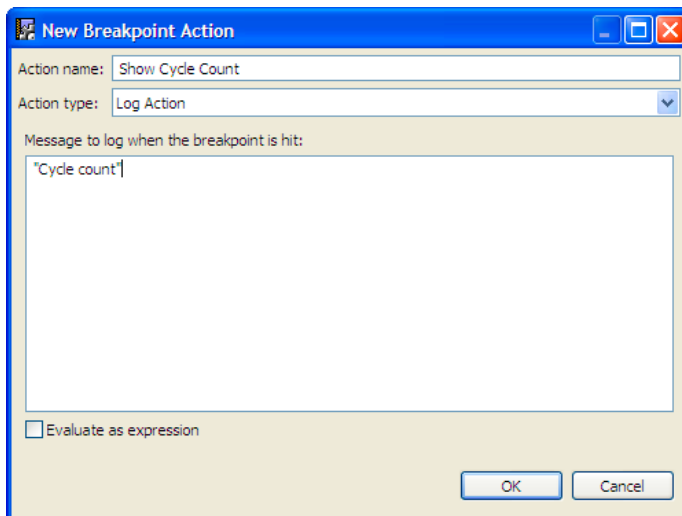
The **Breakpoint Actions** preferences page:

- 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 ([Figure 3.7](#)) appears.

Figure 3.8 New Breakpoint Action Dialog Box



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.

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.

- 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.

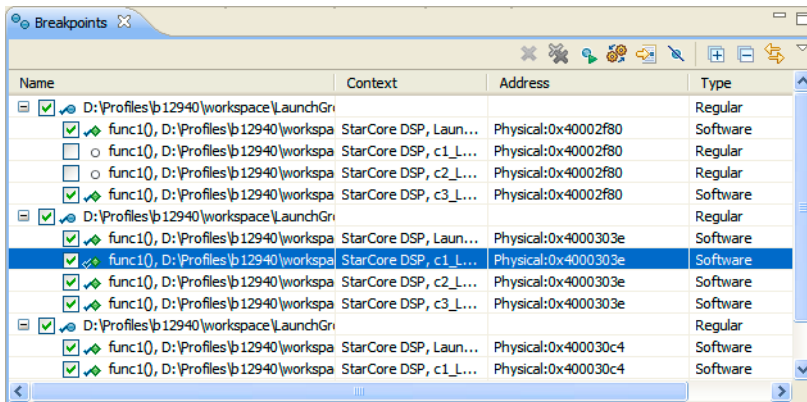
Selecting Breakpoint Template

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.

Figure 3.9 Selecting Breakpoint Template



4. Clear the check box for the core for which you do not want the breakpoint applied.

Build While Debugging

The debug session locks the debugged elf when **Create and Use Copy of Executable** option is not checked, see [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.

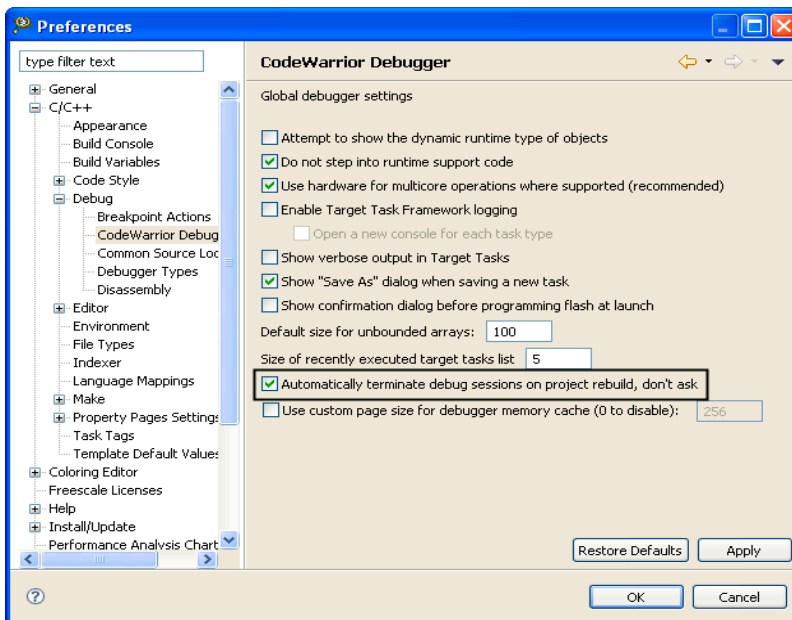
This feature enables automatic termination of the debug sessions when initiating a build that produces executables locked by those debug sessions.

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 ([Figure 3.10](#)) appears in the right-panel of the **Preferences** dialog box.

Figure 3.10 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.

CodeWarrior Debugger Settings

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.

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.

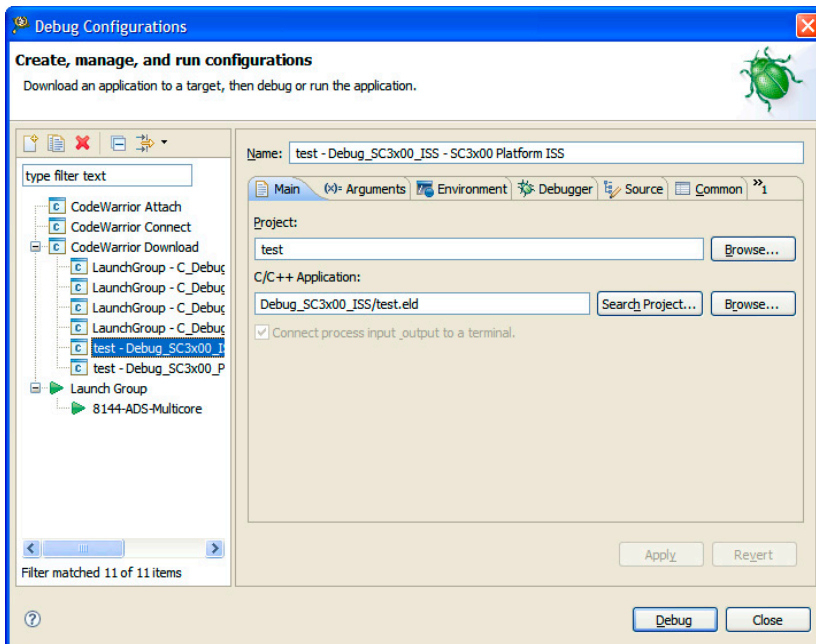
To change debugger settings:

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 ([Figure 3.11](#)) appears.

Debugger

CodeWarrior Debugger Settings

Figure 3.11 Debug Configurations Dialog Box



The left panel of the **Debug Configurations** dialog box lists the debug configurations that apply to the current project.

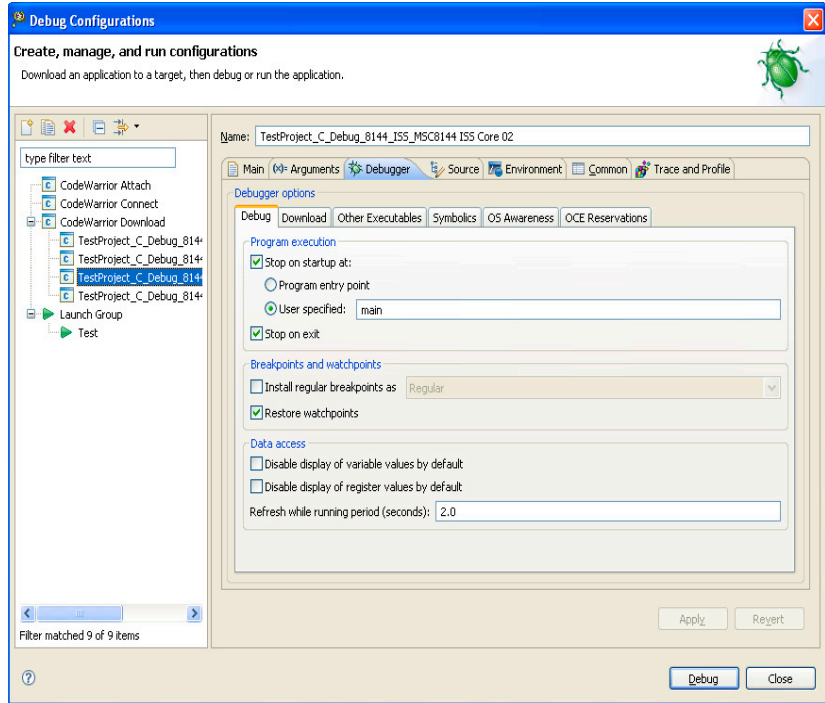
3. Expand the **CodeWarrior Download** configuration.
4. From the expanded list, click the name of the debug configuration you want to modify.

The **Debug Configurations** dialog box shows the settings for the selected configuration.

5. Click the **Debugger** tab.

The **Debugger** page ([Figure 3.12](#)) appears.

Figure 3.12 Debug Configuration Dialog Box — Debugger Page



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.

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.

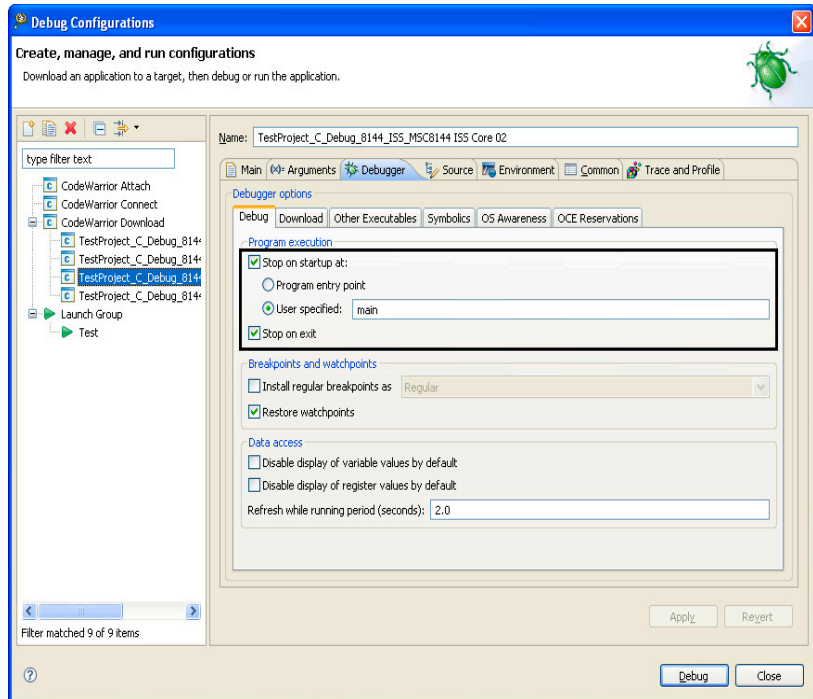
Stopping Debugger at Program Entry Point

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 Download** 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 ([Figure 3.13](#)) appears.

Figure 3.13 Stop Debugger At Program Entry Point



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

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. For more information, see [“Debugger Shell” on page 253](#).

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.

Context Menus

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, Control + click, or click and hold on an item to open a context menu for that item. The context menu appears, displaying menu commands applicable to the selected item.

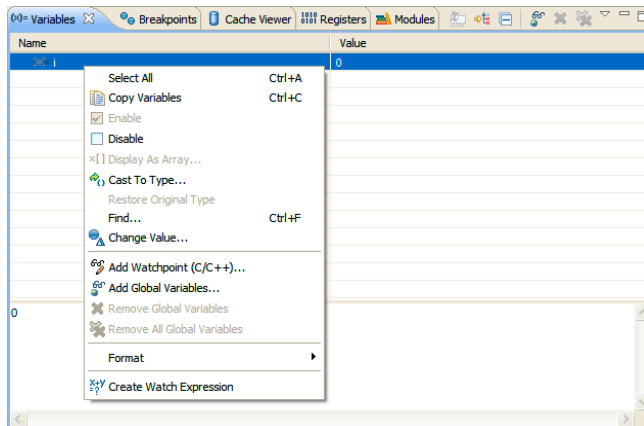
Examples of situations in which the debugger displays a context menu are:

- 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.

[Figure 3.14](#) shows the context menu a variable view displays.

Figure 3.14 Context Menu



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.

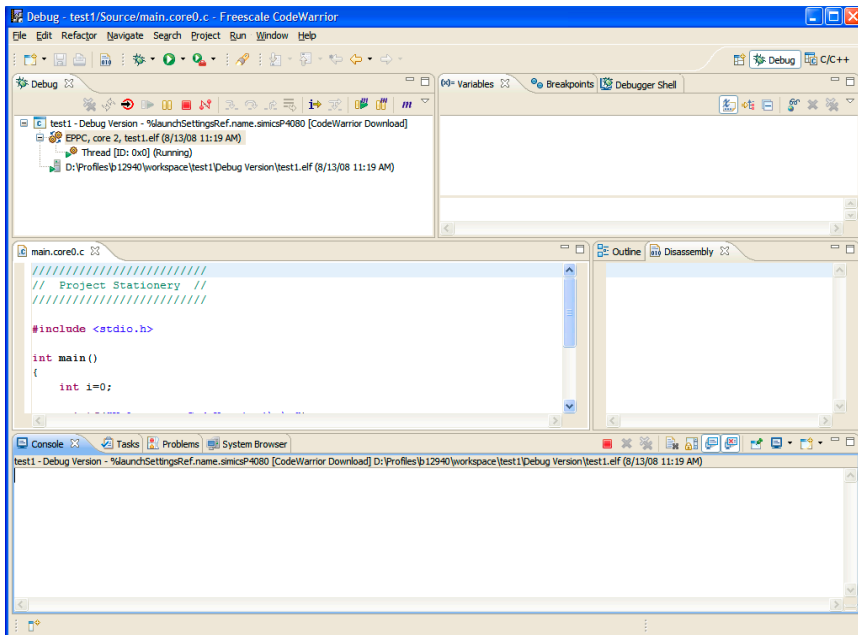
The Debug perspective displays this information:

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

[Figure 3.15](#) shows a **Debug** perspective.

Figure 3.15 Debug Perspective



Debug View

Views support editors and provide alternate presentations as well as ways to navigate the information in your Workbench. The **Debug** view ([Figure 3.16](#)) is a part of the Breakpoints. The **Debug** view 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*.

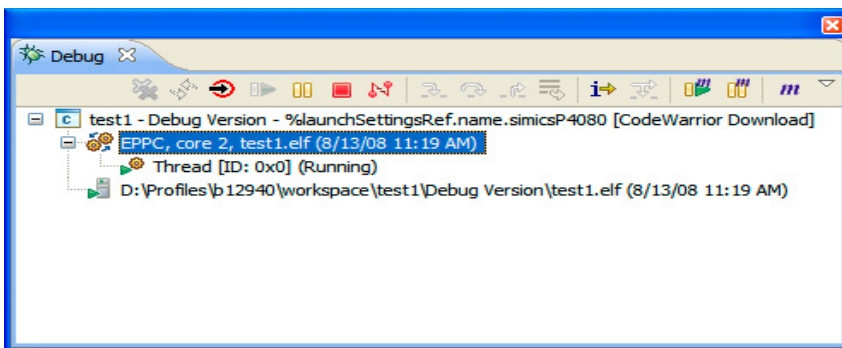
Use the **Debug** view to perform these tasks:

- 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 [Common Debugging Actions](#).

Figure 3.16 Debug View



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:

- [Starting the Debugger](#)
- [Stepping Into Routine Call](#)
- [Stepping Out of Routine Call](#)
- [Stepping Over Routine Call](#)
- [Stopping Program Execution](#)
- [Resuming Program Execution](#)
- [Running a Program](#)
- [Disconnecting a Core](#)
- [Restarting Debugger](#)
- [Debugging in Instruction Stepping Mode](#)
- [Changing the Program Counter Value](#)

Starting the Debugger

Use the `debug` command in the [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 [Debug View](#) toolbar to start the debugger.

The IDE opens a new [Debug View](#).

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

Stepping Into Routine Call

Use the `step` command in the [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 [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.

Stepping Out of Routine Call

Use the `Step Return` command in the [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 [Debug View](#) toolbar to step out of a routine.

The current routine executes and returns to its caller; then program execution stops.

Stepping Over Routine Call

Use the `next` command in the [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,
- watchpoint,
- or an eventpoint that stops execution.



Select **Run > Step Over** or click the **Step Over** button (shown at left) in the [Debug View](#) toolbar to step over a routine.

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

Stopping Program Execution

Use the `kill` command in the [Command-Line Debugger Shell](#) to stop program execution during a debugging session.



Select **Run > Terminate** or click the **Terminate** button (shown at left) in the [Debug View](#) toolbar to stop program execution.

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

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

Resuming Program Execution

Use the `go` command in the [Command-Line Debugger Shell](#) to resume execution of a suspended debugging session.



Select **Run > Resume** or click the **Debug** button (shown at left) in the [Debug View](#) toolbar to resume program execution.

The suspended session resumes.

Running a Program

Use the `run` command in the [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 [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.

Disconnecting a Core



Click the **Disconnect** button (shown at left) in the [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.

Restarting Debugger

Use the `restart` command in the [Command-Line Debugger Shell](#) after stopping program execution. The debugger goes back to the beginning of the program and begins execution again. This behavior is equivalent to killing execution and then starting a new debugging session.



Select **Run > Restart** or click the **Restart** button (shown at left) in the [Debug View](#) toolbar to restart the debugger.

Debugging in Instruction Stepping Mode

Use the `stepi` command in the [Command-Line Debugger Shell](#) to debug a program in instruction stepping mode. In this mode, you can debug the program in [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 [Debug View](#) toolbar.

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.

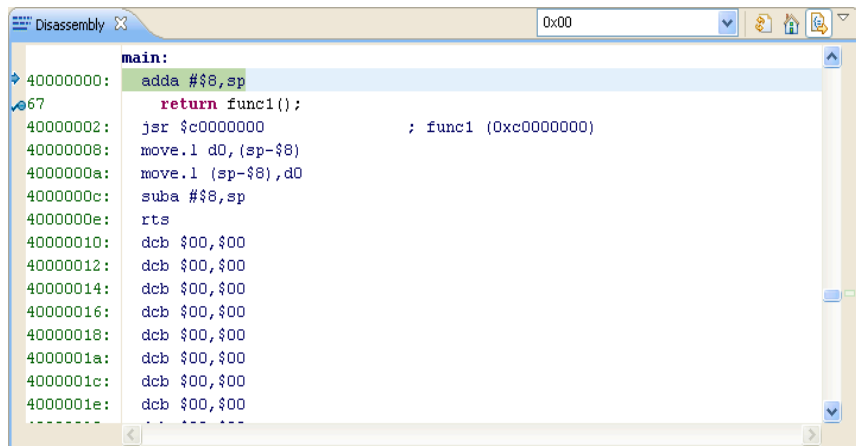
Disassembly View

The **Disassembly** view ([Figure 3.17](#)) 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

Figure 3.17 Disassembly View



Environment Variables in Launch Configuration

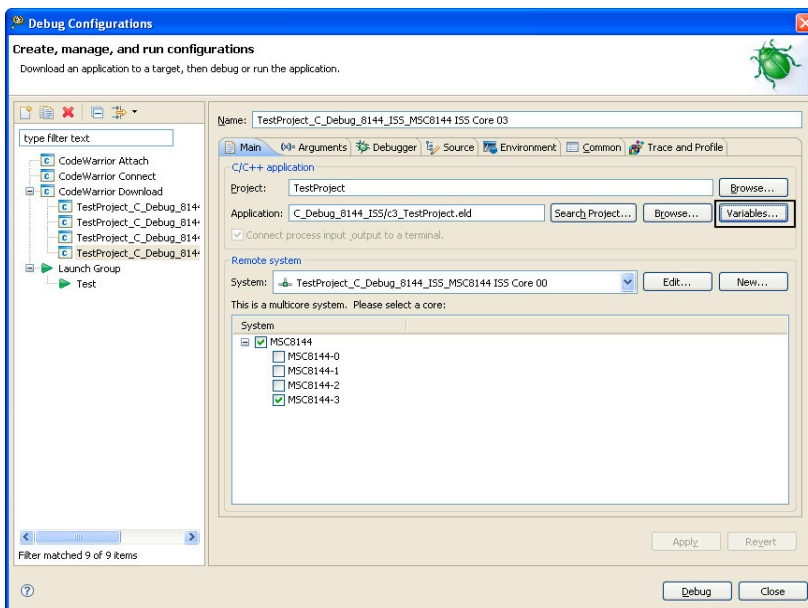
CodeWarrrior enables you to use environment or eclipse variables to specify the path of the launch executable.

To specify an environment or eclipse variable:

1. Click **Run > Debug Configuration**.

The **Debug Configurations** dialog box ([Figure 3.18](#)) appears.

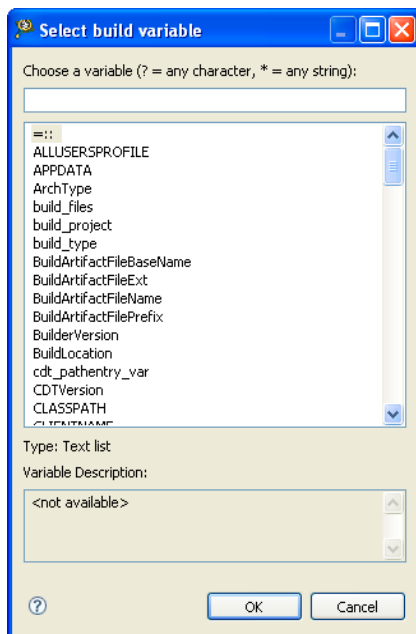
Figure 3.18 Debug Configurations Dialog Box



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

The **Select build variable** dialog box ([Figure 3.19](#)) appears.

Figure 3.19 Select Build Variable Dialog Box



4. Select a variable from the variable list.
5. Click **OK**.

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.

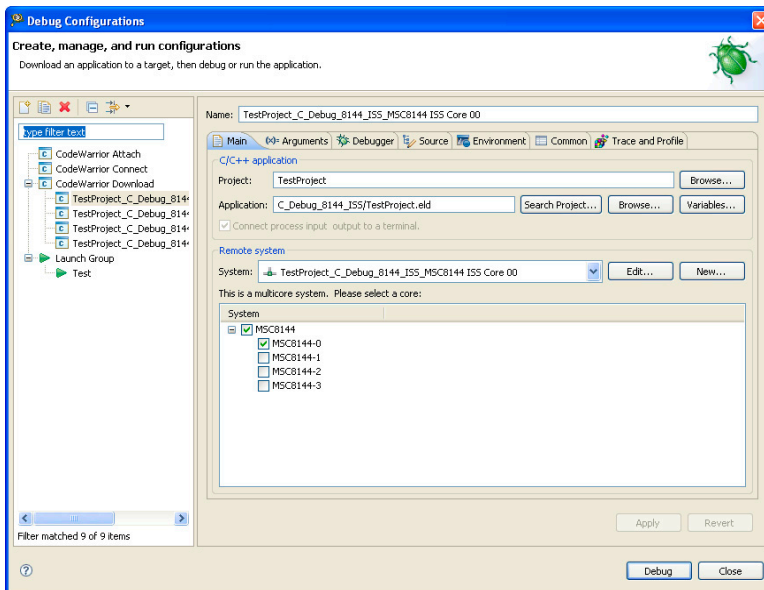
Creating a Launch Group

To create a launch group:

1. Select **Run > Debug Configurations**.

The **Debug Configurations** dialog box ([Figure 3.20](#)) appears.

Figure 3.20 Debug Configurations Dialog Box



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 ([Figure 3.21](#)) of the **Debug Configurations** dialog box.

Figure 3.21 Launch Group Configuration Panel Controls

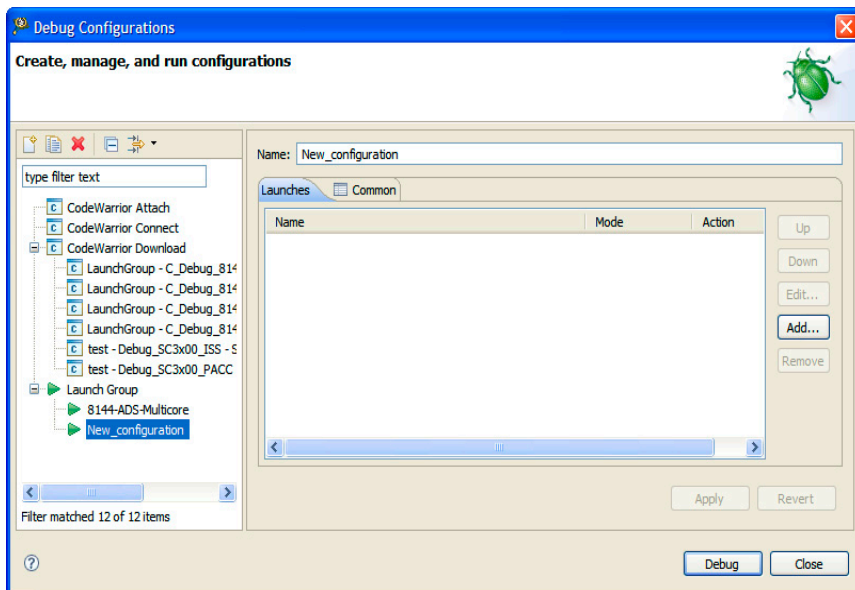


Table 3.5 Launch Group Configuration Panel Controls

Control	Description
Name	Specify a name for the launch group
Up button	Click to move up the selected launch configuration
Down button	Click to move down the selected launch configuration
Edit button	Click to edit the selected entry in the launch group
Add button	Click to add a launch configuration to the launch group
Remove button	Click to remove a launch configuration from the launch group

- Specify a name for the launch group configuration in the Name text box.

5. Click **Add**.

The **Add Launch Configuration** dialog box ([Figure 3.22](#)) appears.

Figure 3.22 Add Launch Configuration Dialog Box

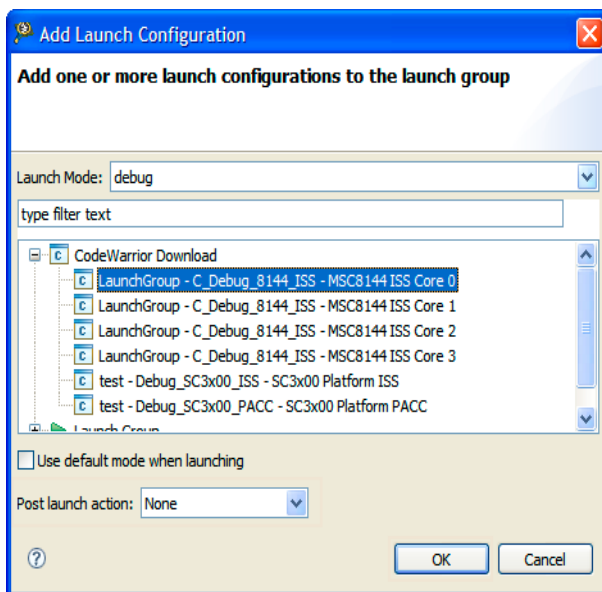
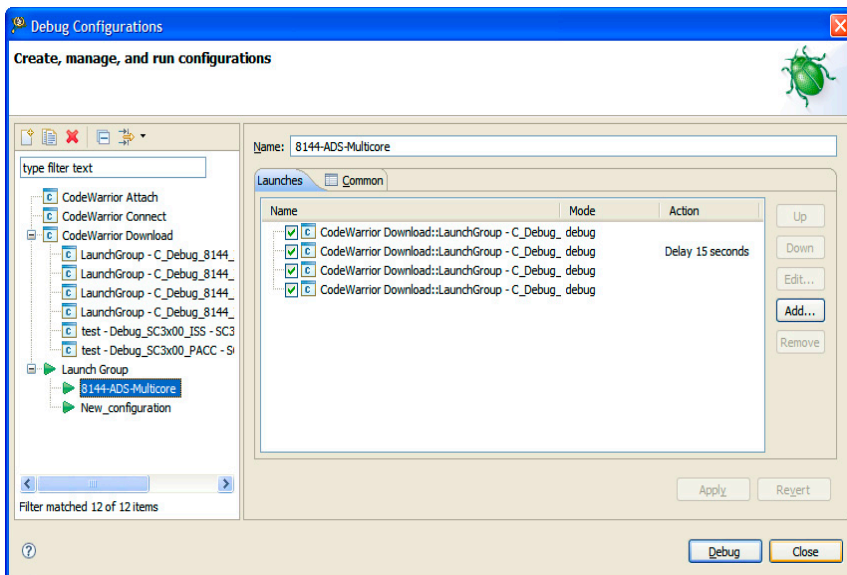


Table 3.6 Add Launch Configuration Dialog Box Options

Option	Description
Launch Mode	<p>Enables you to specify launch mode for the selected launch configuration. This can also be used to filter launch configurations.</p> <p>debug — specifies that the launch configuration will be launched in debug mode.</p> <p>run — specifies that the launch configuration will be launched in run mode.</p> <p>profile — specifies that the launch configuration will be launched in profile mode.</p>
Use default mode when launching	<p>Checking this option indicates that the child launch configuration should be launched in the mode used to initiate the launch group launch.</p>
Post launch action	<p>Enables you to specify a post launch action for the selected launch configuration.</p> <p>None — the debugger immediately moves on to launch the next launch configuration.</p> <p>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.</p> <p>Delay — the debugger waits for specified number of seconds before moving on to the next launch configuration.</p>

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 ([Figure 3.23](#)).

Figure 3.23 Launch Group

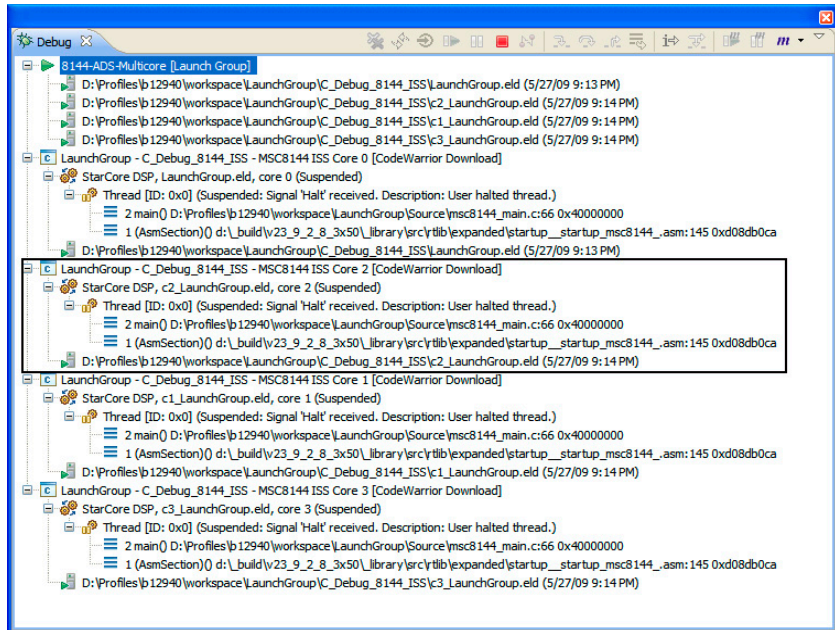


Launching the Launch Group

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.

[Figure 3.24](#) shows the result of a launch group launch in the **Debug** view.

Figure 3.24 Launch Group in Debug View



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.

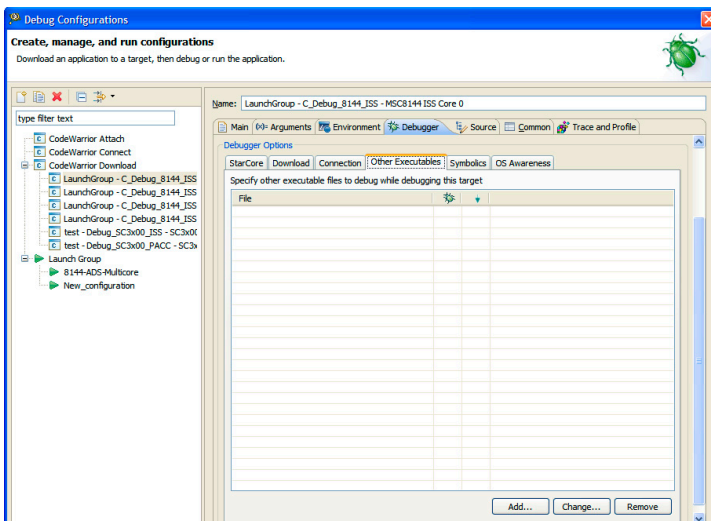
2. Expand the **CodeWarrior Download** configuration.
3. From the expanded list, select the debug configuration that you want to modify.

[Figure 3.25](#) shows the **Debug Configurations** dialog box with the settings for the debug configuration you selected.

Debugger

Load Multiple Binaries

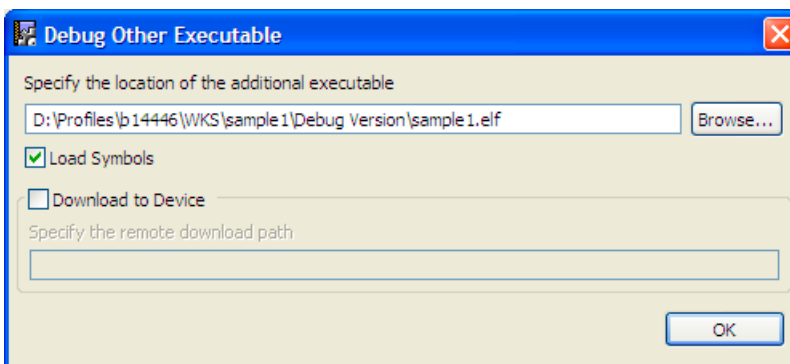
Figure 3.25 Debug Configurations Dialog Box





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 (Figure 3.26) enables you to specify additional ELF files to download or debug in addition to the main executable file associated with the launch configuration.

Figure 3.26 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.

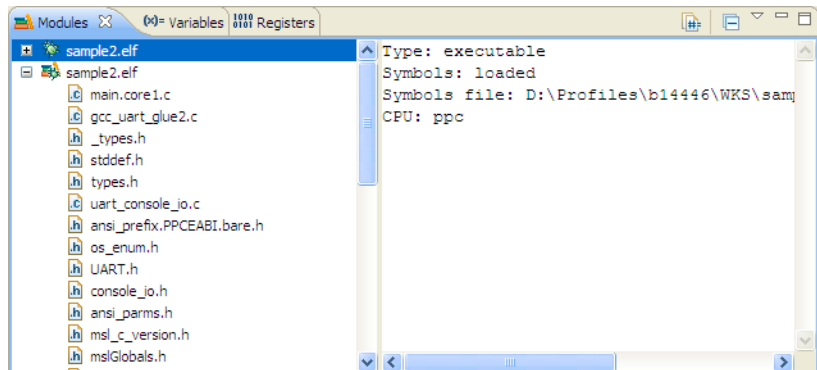
Viewing Binaries

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 [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 ([Figure 3.27](#)) appears.

Figure 3.27 Modules View



2. Click on the application executable to view its details.
An executable can also be expanded in the modules view (to show its symbols) regardless of whether the executable has been targeted or not in the **Debug Other Executables** panel.

Debugger

Memory View

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.

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:

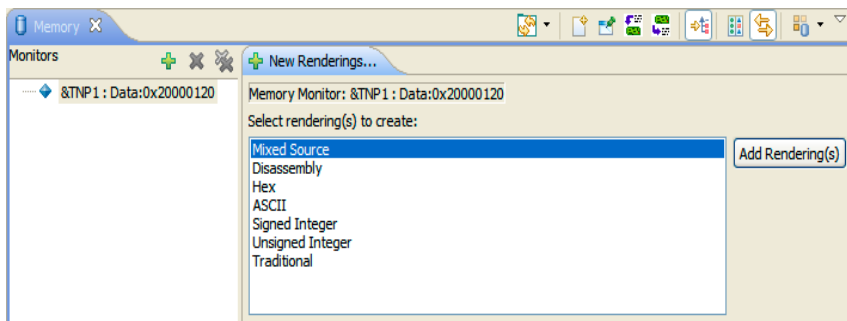
- **Monitors** panel — Displays the list of memory monitors added to the debug session currently selected in the **Debug** view
- **Renderings** panel — Display memory renderings.

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.

Opening Memory View

To open the **Memory** view ([Figure 3.28](#)), click the **Memory** tab of the **Debug** perspective. Alternatively, from the IDE menu bar, select **Window > Show View > Memory**.

Figure 3.28 Memory View



Adding Memory Monitor

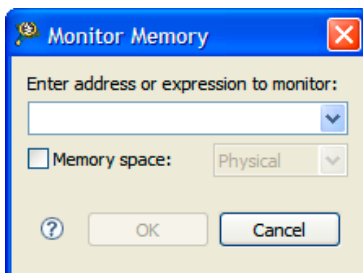
To add a memory monitor to **Memory** view:

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 ([Figure 3.29](#)) appears.

Figure 3.29 Monitor Memory Dialog Box



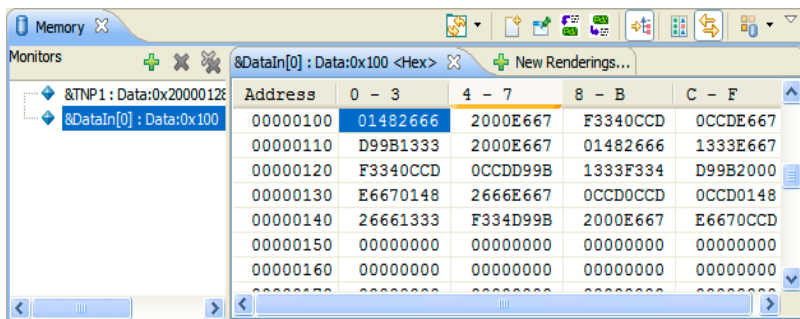
4. Specify information about the memory monitor:
 - To enter a memory space and literal address, simply enter an address.
 - To enter an expression, type in the expression. If you enter a literal address as the expression, use the prefix 0x to indicate hexadecimal notation, or use no prefix to indicate 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 3.30](#)).

Figure 3.30 Added Memory Monitor



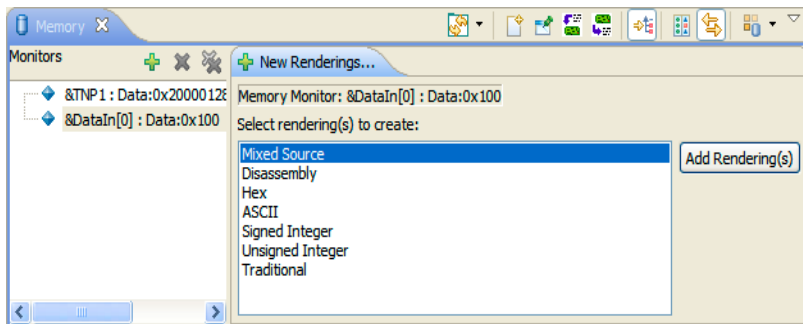
Adding Memory Renderings

To add a memory rendering to the **Memory** view:

1. Open the **Memory** view, see [Opening Memory View](#).
2. Add a memory monitor, see [Adding Memory Monitor](#).
3. Click the **New Renderings** tab in the **Renderings** panel.

The **New Renderings** tab ([Figure 3.31](#)) displays the different rendering types that can be added in the **Renderings** panel.

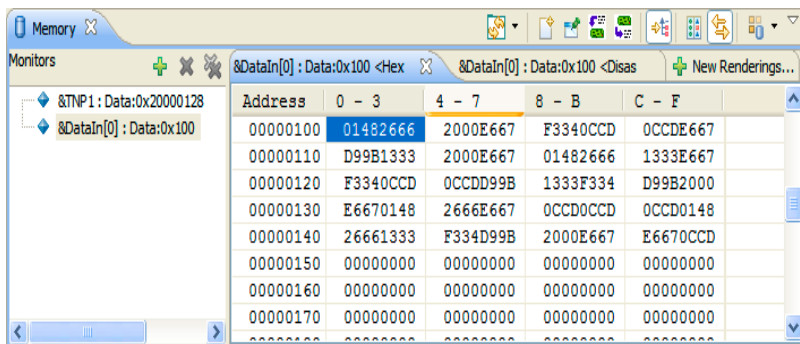
Figure 3.31 Add Rendering



4. Select a rendering from the **Select rendering(s) to create** list.
5. Click the **Add Rendering(s)** button.

The rendering is added to the **Renderings** panel in the **Memory** view ([Figure 3.32](#)).

Figure 3.32 Added Rendering



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:

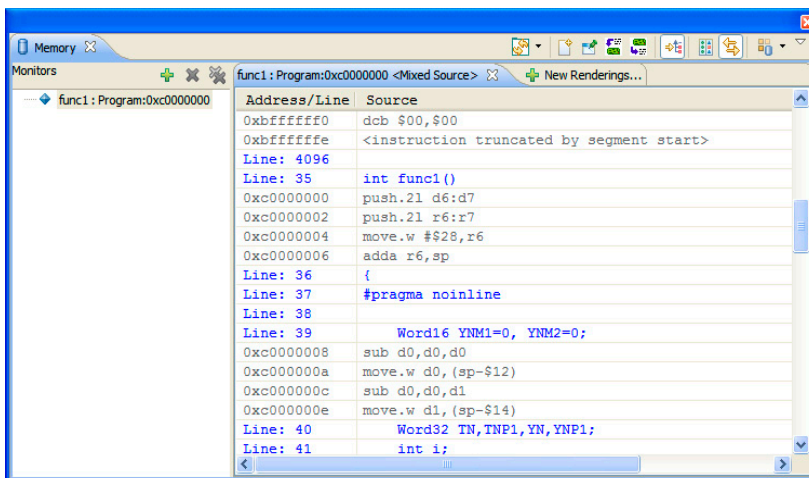
1. Open the **Memory** view, see [Opening Memory View](#).
2. Add a memory monitor, see [Adding Memory Monitor](#).
3. Click the **New Renderings** tab in the **Renderings** panel.

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 ([Figure 3.33](#)).

Figure 3.33 Mixed Source Rendering

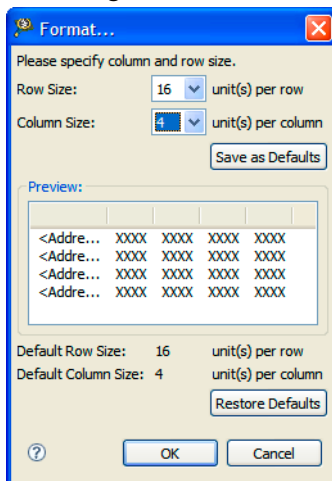


Setting Memory Access Size

To set memory access size:

1. Open the **Memory** view, see [Opening Memory View](#).
2. Right-click on a memory monitor in the **Memory** view.
The context menu appears.
3. Select **Format** from the context menu.
The **Format** dialog box ([Figure 3.34](#)) appears.

Figure 3.34 Format Dialog Box



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.

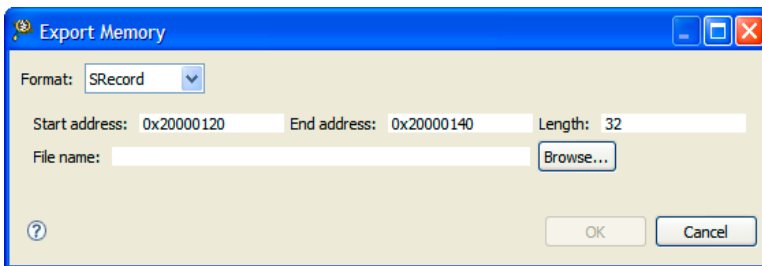
6. Click **OK**.

Exporting Memory

To export memory data:

1. Open the **Memory** view, see [Opening Memory View](#).
2. Click the **Export** button in the **Memory** view toolbar.
The **Export Memory** dialog box ([Figure 3.35](#)) appears.

Figure 3.35 Export Memory Dialog Box



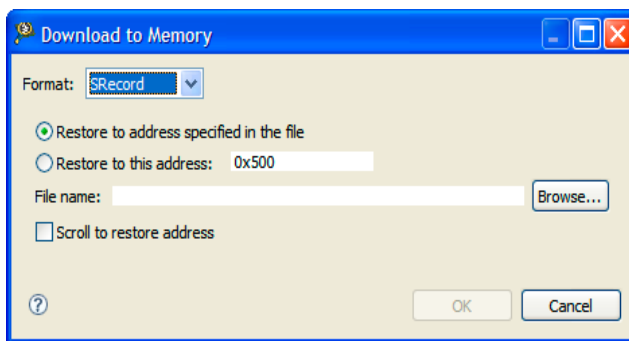
- **Format** drop-down list — Enables you to select the format in which the memory data is exported.
 - SRecord — Exports memory data in Motorola S-record format.
 - Plain Text — Exports memory data in ASCII format.
 - RAW Binary — Exports memory data in binary format.
 - **Start address** text box — 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**.

Importing Memory

To import memory data:

1. Open the **Memory** view, see [Opening Memory View](#).
2. Click the **Import** button in the **Memory** view toolbar.
The **Download to Memory** dialog box ([Figure 3.36](#)) appears.

Figure 3.36 Download to Memory Dialog Box



- **Format** drop-down list — Enables you to select the memory format.
 - **Restore to address specified in the file** option — If selected, the imported memory is restored to the memory location specified in the memory file.
 - **Restore to this address** text box — Enables you to specify a memory address to store the imported memory. The imported memory is restored to the memory location specified in the text box.
 - **File name** text box — Enables you to specify the file name to import memory. Click the **Browse** button to select a file from your system.
 - **Scroll to restore address** check box — If checked, the content in the memory view scroll to the restore point after the export operation is completed.
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 memory 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 complete.
 8. Click **OK**.

Setting Watchpoint in Memory View

To set a watchpoint using the **Memory** view:


Debugger

Memory Browser View


1. Select **Windows > Open Perspective > Debug** from the IDE menu bar to switch to the **Debug** perspective.
2. Select **Window > Show View > Memory**.
The **Memory** view displays.
3. Select a range of bytes in the **Memory Renderings** panel of the **Memory** view.
4. Right-click and select **Add Watchpoint (C/C++)** from the context menu that appears.

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.

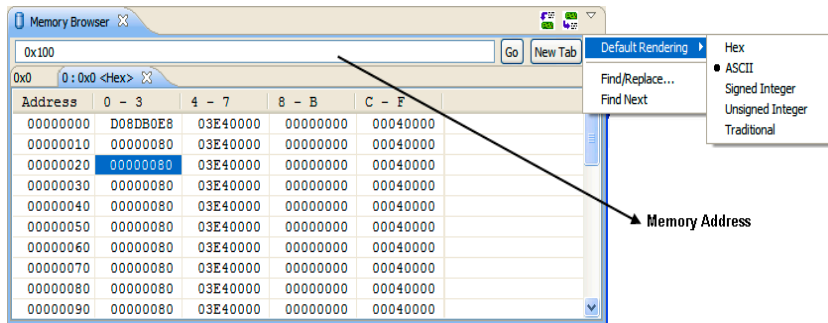
NOTE All watchpoints clear automatically when the target program terminates or the debugger terminates the program. Your watchpoints are reset the next time the program runs.

Memory Browser View

The **Memory Browser** view lets you monitor your process memory. This view also enables you to browse through the memory rendering.

To open the **Memory Browser** view ([Figure 3.37](#)), click the **Memory Browser** tab of the **Debug** perspective. Alternatively, from the IDE menu bar, select **Window > Show View > Memory Browser**.

Figure 3.37 Memory Browser View



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.

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.

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

Multicore Suspend

To suspend execution of a core:

Debugger

Multicore Debugging

1. Enable multicore groups for multicore operations (see [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 [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.

Multicore Resume

To resume execution of a core:

1. Enable multicore groups for multicore operations (see [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 [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.

Multicore Terminate

To terminate execution of a core:

1. Enable multicore groups for multicore operations (see [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 [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.

Multicore Restart

To restart execution of a core:

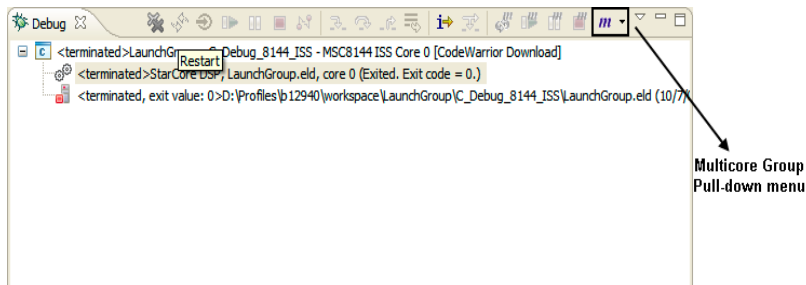
1. Enable multicore groups for multicore operations (see [Multicore Groups](#)).
2. In the **Debug** view, select a thread that corresponds to a core for bareboard debugging.
3. Click **Multicore Restart**.

Alternatively, in the [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.

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 ([Figure 3.38](#)) in the **Debug** view toolbar enables you to create new multicore groups, see [Creating a Multicore Group](#). For more information on multicore debugging, see [Multicore Debugging](#).

Figure 3.38 Multicore Groups



The **Multicore Groups** pull-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 [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 Multicore System Types** — Opens the **Multicore Types** dialog box to add or remove multicore system types, see [Editing a System Type](#).
- **Edit Multicore Groups** — Opens the **Multicore Groups** dialog box to create multicore groups, see [Creating a Multicore Group](#). You can also use this option to modify existing multicore groups, see [Modifying a Multicore Group](#).

The **Multicore Groups** pull-down menu also shows the list of groups that are shown in the **Multicore Groups** dialog box.

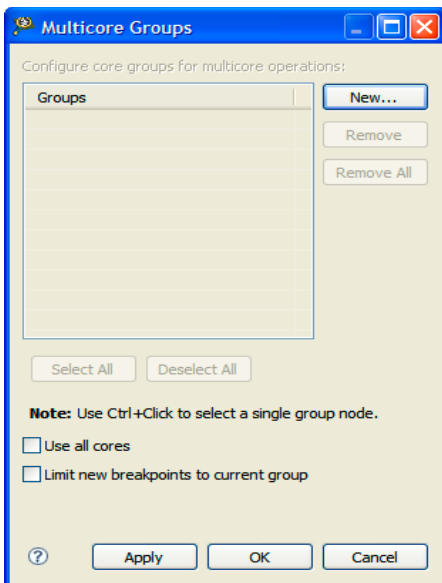
Creating a Multicore Group

To create a multicore group:

1. Click the **Multicore Groups** button from the **Debug** view toolbar.

The **Multicore Groups** dialog box ([Figure 3.39](#)) appears.

Figure 3.39 Multicore Groups



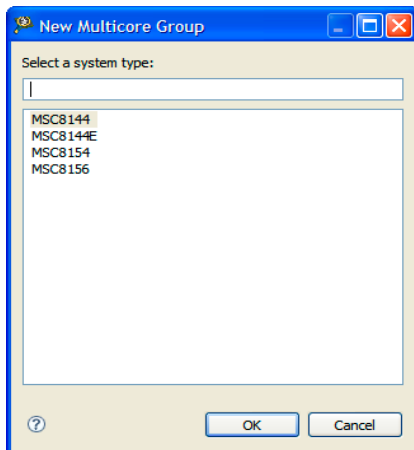
- **New** button — Creates a new group using the **New Multicore Group** dialog box. The initial name of the group is the system type name unless the name is already in use. If the name is already in use then an index is appended to the group name. The

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.
 - **Select All** button — Selects all groups, processors and cores.
 - **Deselect All** button — Deselects all groups, processors and cores.
 - **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 ([Figure 3.40](#)) appears.

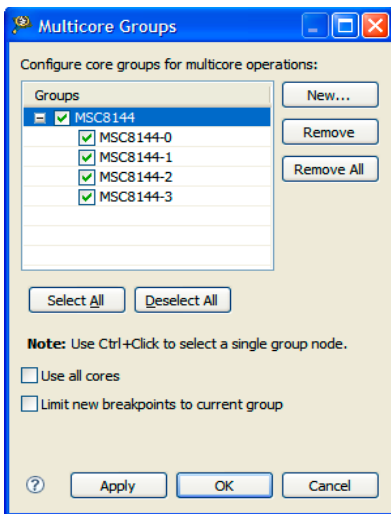
Figure 3.40 New Multicore Group



3. Select a system type from the list.
4. Click **OK**.

The group appears in the **Multicore Groups** dialog box ([Figure 3.41](#)).

Figure 3.41 Added Multicore Group



5. Repeat **Steps 2 - 4** to add more core groups for multicore operations.
6. Click **OK**.

Modifying a Multicore Group

You can also modify an existing multicore group.

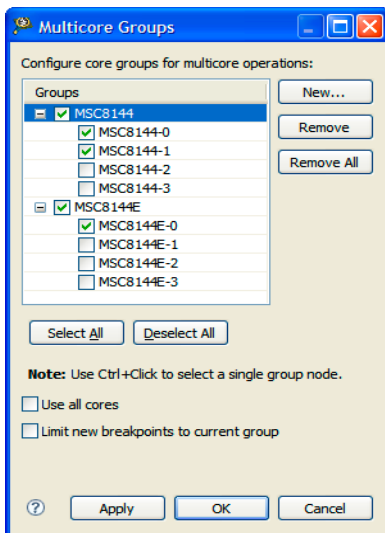
NOTE You are not allowed to enable a group that overlaps with another group.

To modify a multicore group:

1. Select the **Edit Multicore Groups** option from the **Multicore Groups** pull-down menu in the **Debug** view toolbar.

The **Multicore Groups** dialog box ([Figure 3.42](#)) appears.

Figure 3.42 Modify a Multicore Group



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**.

Editing a System Type

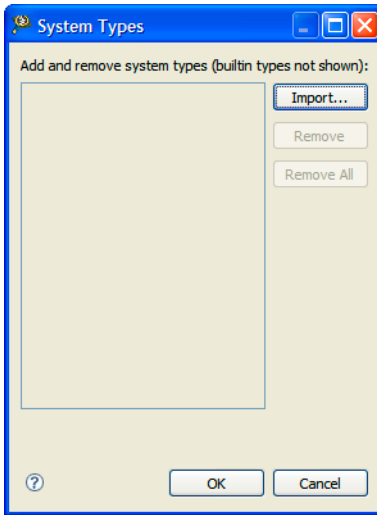
You can add custom system types by importing system types from:

- JTAG configuration files
- Device Tree Blob files (for Power Architecture)

To add a system type:

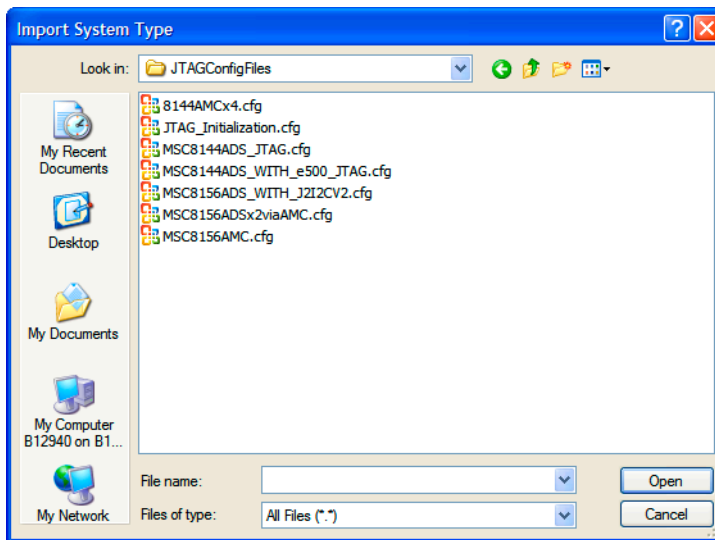
1. Click the **Edit System Types** option from the **Multicore Groups** pull-down menu. The **System Types** dialog box ([Figure 3.43](#)) appears.

Figure 3.43 System Types



- **Import** — Creates a custom system type by importing it from a configuration file.
 - **Remove** — Removes a system type from the list.
 - **Remove All** — Removes all system types from the list.
2. Click **Import**.
- The **Import System Type** dialog box ([Figure 3.44](#)) appears.

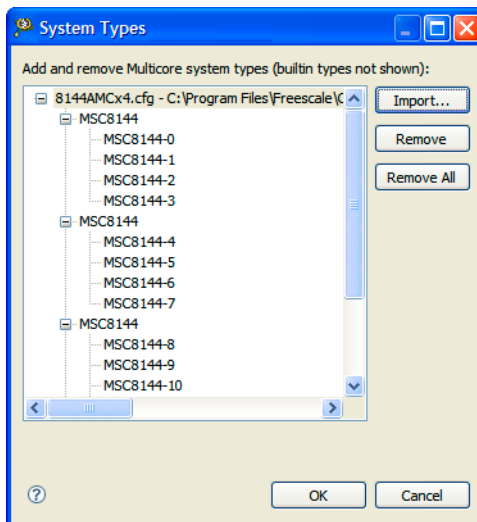
Figure 3.44 Import System Type



3. Select a multicore system type configuration file, then click **Open**.

The multicore system type appears in the **System Types** dialog box ([Figure 3.45](#)).

Figure 3.45 Added System Type



4. Click **OK**.

Using Multicore Group Debugging Commands

Multicore Group features can also be accessed from the Debugger Shell command line. [Table 3.7](#) lists and defines different multicore group debugging commands.

Table 3.7 Multicore Group Debugging Commands

Command	Description
<code>mc::type</code>	Syntax <code>mc::type</code> Lists the available system types.
<code>mc::type import</code>	Syntax <code>mc::type import <filename></code> Imports a new system type specified using the filename.
<code>mc::type remove</code>	Syntax <code>mc::type remove <filename> <type-index> ...</code> Removes the specified imported system type or types. Built-in system types cannot be removed and will return an error.
<code>mc::type removeall</code>	Syntax <code>mc::type removeall</code> Removes all imported system types.
<code>mc::group</code>	Syntax <code>mc::group</code> Lists the defined groups.
<code>mc::group new</code>	Syntax <code>mc::group new <type-name> <type-index> [<name>]</code> 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.

Table 3.7 Multicore Group Debugging Commands

Command	Description
<code>mc::group rename</code>	<p>Syntax</p> <pre>mc::group rename <name> <group-index> <new-name></pre> <p>Renames an existing group. Specifying a duplicate name results in an error.</p>
<code>mc::group remove</code>	<p>Syntax</p> <pre>mc::group remove <name> <group-index> ...</pre> <p>Removes the specified group or groups.</p>
<code>mc::group removeall</code>	<p>Syntax</p> <pre>mc::group removeall</pre> <p>Removes all groups.</p>
<code>mc::group enable disable</code>	<p>Syntax</p> <pre>mc::group enable disable <index> ... all</pre> <p>Enables or disables nodes in the group tree.</p>

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.

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.

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.

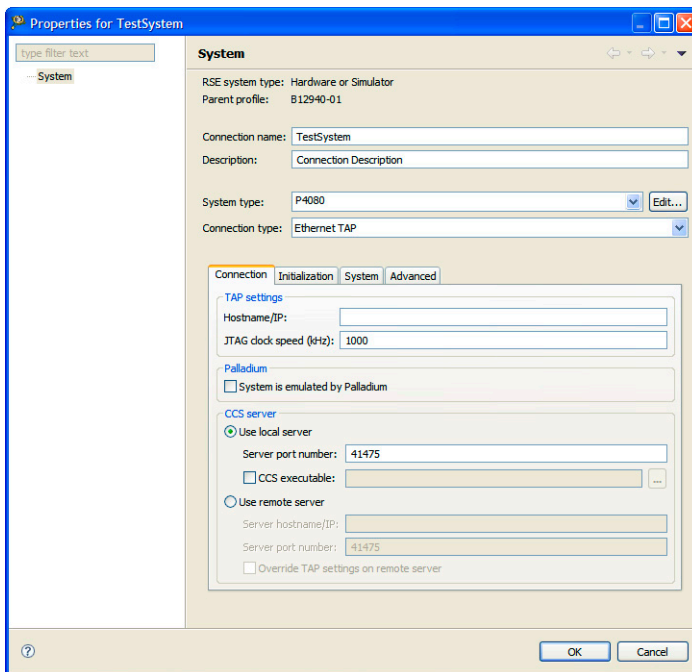
NOTE The `system reset` action is applicable for initial launch only.

To specify reset setting for cores in a multicore environment:

1. Go to [Remote Systems View](#).
2. Right-click a remote system and select **Properties** from the context menu.

The **Properties for <Remote System>** dialog box ([Figure 3.46](#)) appears.

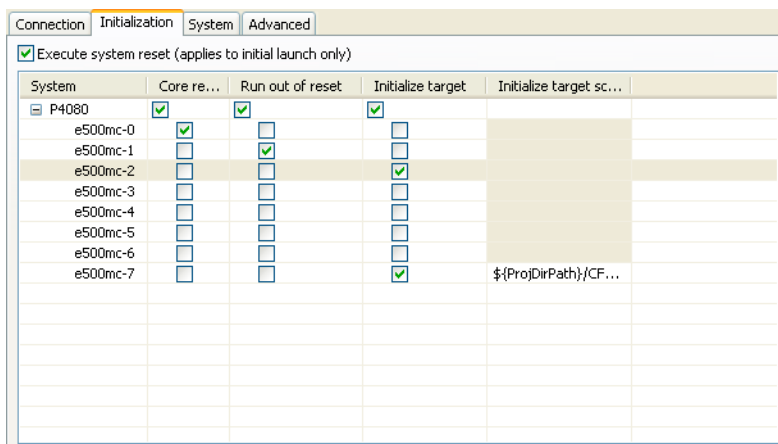
Figure 3.46 Properties for <Remote System> Dialog Box



3. Click the **Initialization** tab.

The initialization settings page ([Figure 3.47](#)) appears.

Figure 3.47 Initialization Settings Page



- **Remote System Explorer (RSE) Configurator initial launch** — The initial launch session for the RSE, 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**.

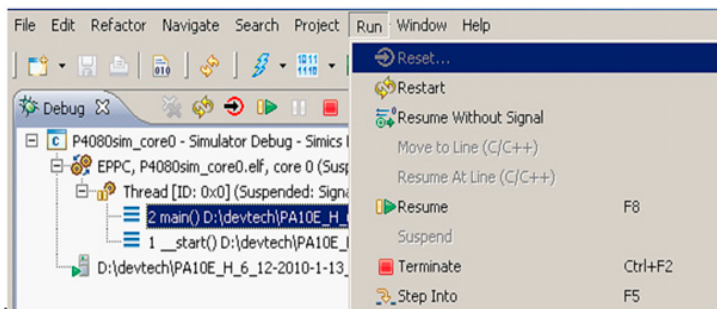
NOTE Initialization files are executed only for cores selected for debug.

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 3.48](#)).

Figure 3.48 On Demand Reset



Path Mapping

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:

- [Automatic Path Mapping](#)
- [Manual Path Mapping](#)

Automatic Path Mapping

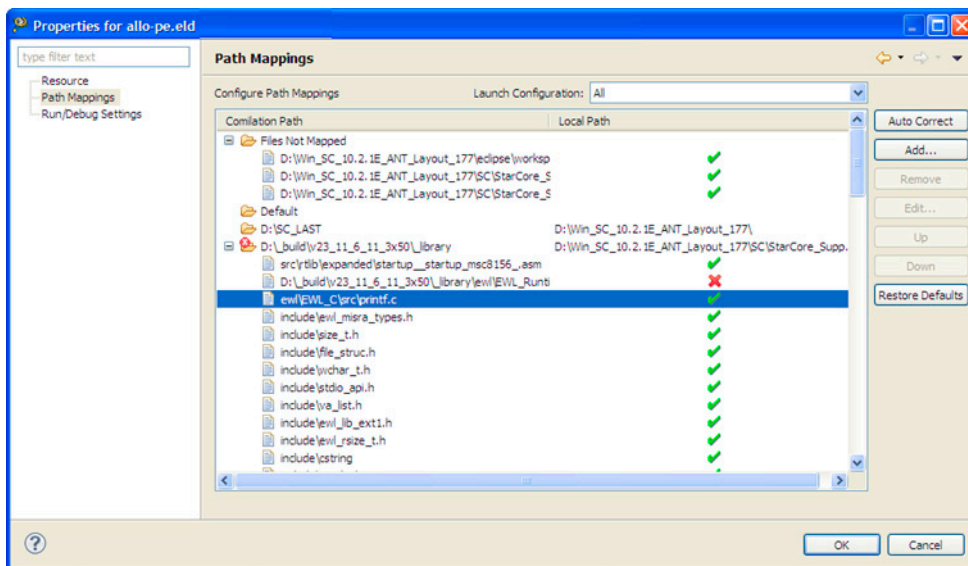
The Automatic Path Mapping feature focuses on reducing as much as possible the manual steps required by the user to setup the path mapping settings in order to support source level debugging.

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 ([Figure 3.49](#)) appears. The Path Mapping Configuration page displays every path mapping settings for the launch configurations associated with a project.

Figure 3.49 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

Debugger

Path Mapping

column, a green (✓) is displayed if the file exists after being mapped by the destination path or a red (✗) 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.

[Table 3.8](#) describes various options available in the Path Mappings page.

Table 3.8 Automatic Path Mappings Options

Options	Description
Auto Correct	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.
Add	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.
Remove	The Remove button allows you to remove any path mapping or default entry.
Edit	The Edit button allows you to change the values of the selected path mapping entry. Editing non-path mapping entry is not supported.
Up	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.

Table 3.8 Automatic Path Mappings Options

Options	Description
Down	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.
Restore Defaults	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 plugin.

NOTE 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.

Manual Path Mapping

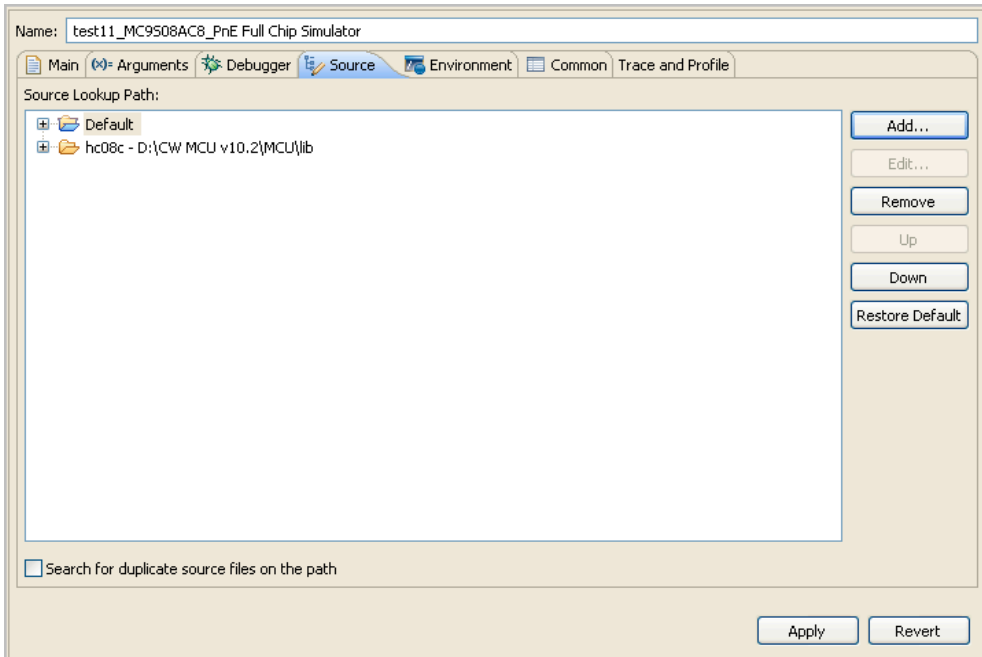
You need to specify the source lookup path in terms of the compilation path and the local file-system path for the newly imported executable file. The CodeWarrior debugger uses both of these paths to debug the executable file. The compilation path is the path to the original project that built the executable 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.

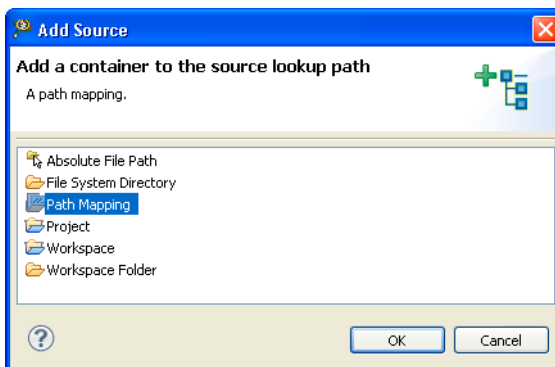
The corresponding page ([Figure 3.50](#)) appears.

Figure 3.50 Debug Configurations - Source Page



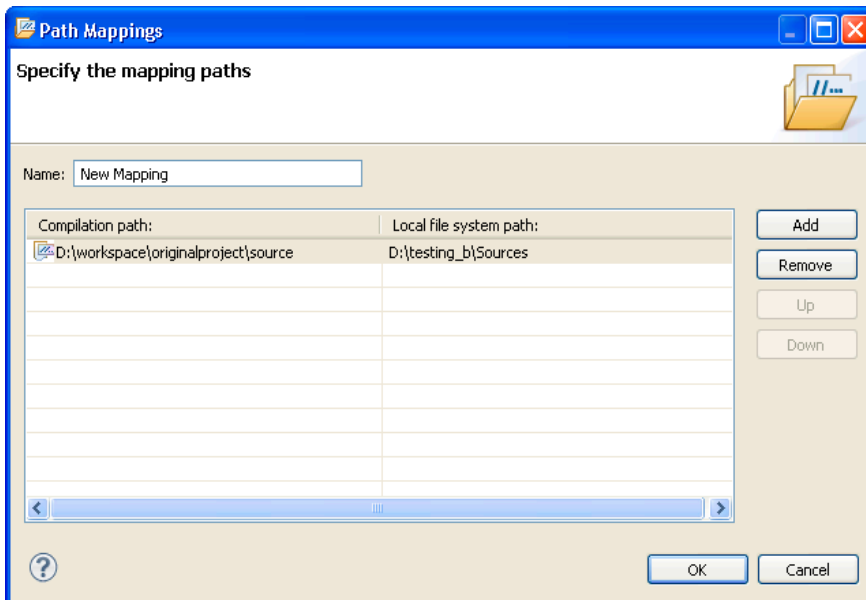
2. Click **Add**.
 The Add Source dialog box appears.
3. Select **Path Mapping** ([Figure 3.51](#)).

Figure 3.51 Add Source dialog box



4. Click **OK**.
The **Path Mappings** dialog box appears.

Figure 3.52 Path Mappings dialog box



5. Specify the Path mappings name in **Name** text box.
6. Click **Add**.

- 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.

- 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.

- 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.

- 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.

- Click **Apply**.

The IDE saves your changes.

Adding Path Mapping to Workspace

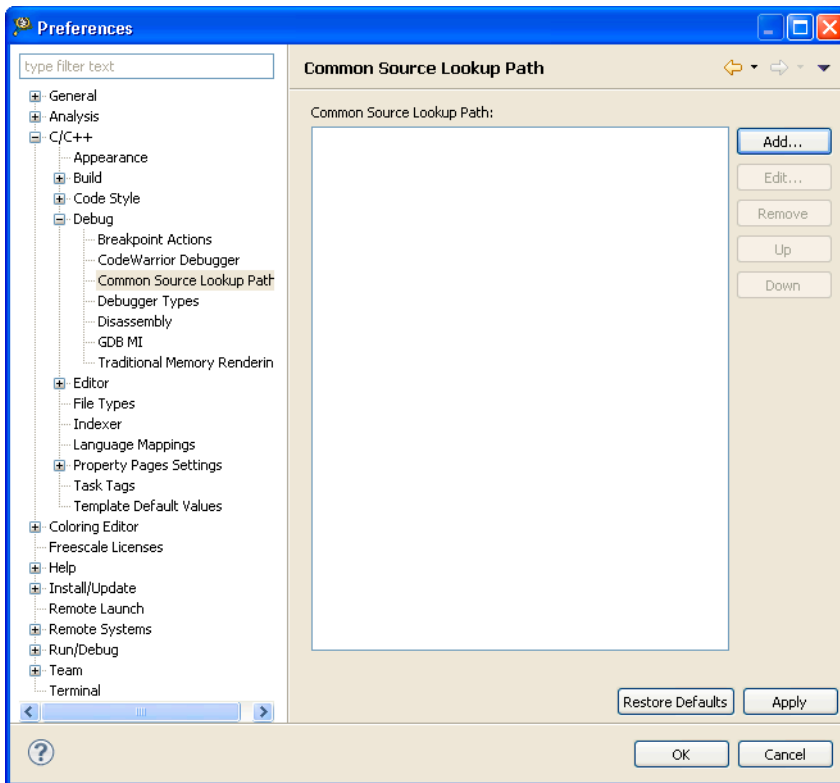
To add a path mapping to the workspace:

- Select **Window > Preferences** from the CodeWarrior IDE menu bar.

The Preferences dialog box appears.

- Expand **C/C++ > Debug > Common Source Lookup Path**.

Figure 3.53 Preferences - Common Source Lookup Path



3. Repeat [step 2.](#) to [step 11.](#) from the previous section for adding a path mapping for a single launch configuration.

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.

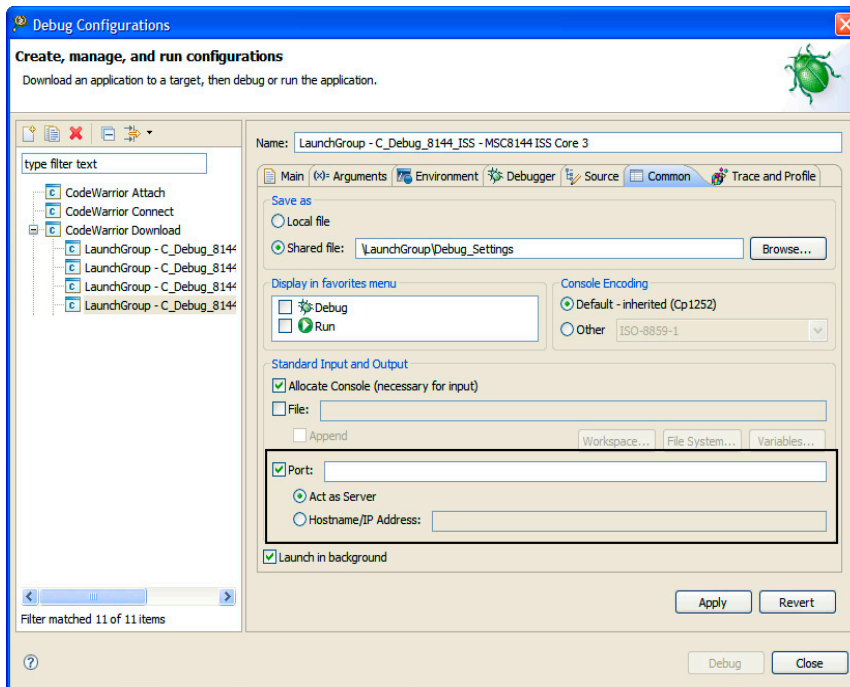
To specify the initial connection redirection settings:

Debugger

Redirecting Standard Output Streams to Socket

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 Download 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 3.54 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.

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 (shown in left) 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 Download Configuration** tree.

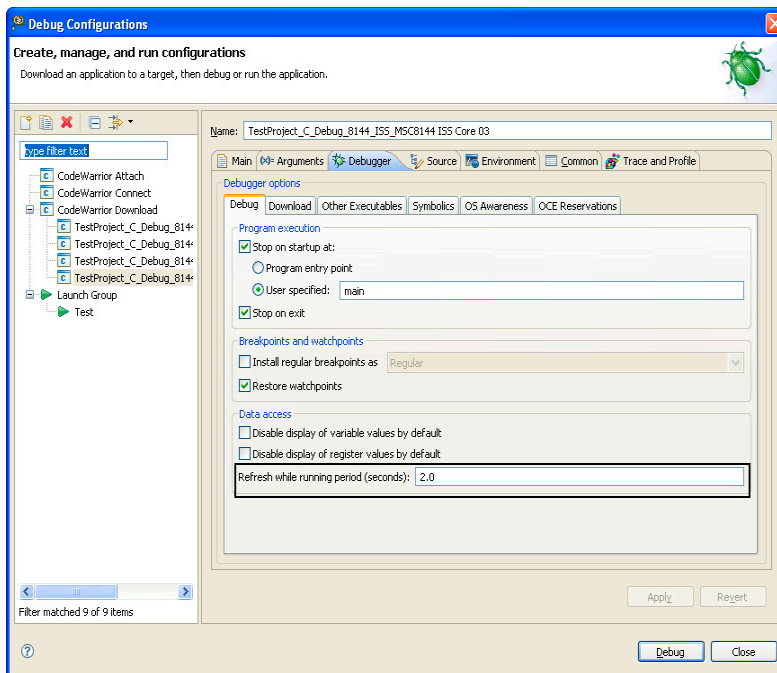
Debugger Registers View

- 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.

- Click the **Debugger** tab.
- Click the **Debug** tab from the **Debugger Options** group.

Figure 3.55 View Refresh Settings



- Type the refresh interval in the **Refresh while running period (seconds)** text box.
- Click **Apply**.

The changes are applied to the selected debug configuration.

Registers View

The **Registers** view ([Figure 3.56](#)) 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:

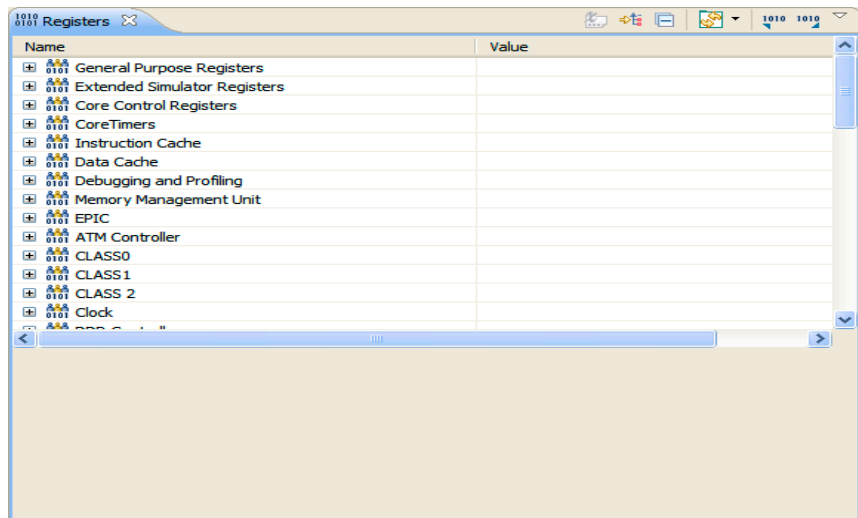
- add, edit, or remove groups of registers
- view register details, such as explanations of a register's bit fields and values
- change register values
- import/export register data

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*).

Figure 3.56 Registers View



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.

Viewing Registers

To view registers content:

1. Open the **Registers** view ([Figure 3.56](#)).
2. Expand a register group.

Expanding a group shows its content by register name and the content of each register in the group.

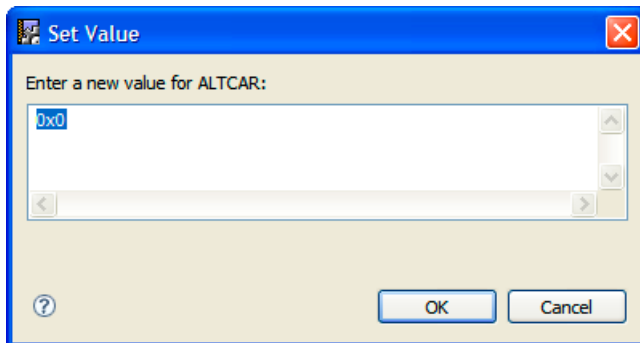
Changing Register Values

To change the value of a register:

1. Open the **Registers** view ([Figure 3.56](#)).
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 3.57 Set Value Dialog Box



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.

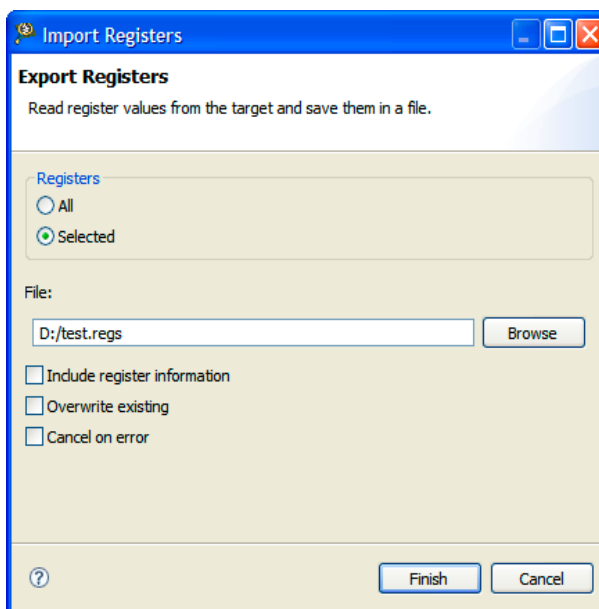
Exporting Registers

To export register data to a file:

1. Open the **Registers** view ([Figure 3.56](#)).
2. Click the **Export registers** button  in the **Registers** view toolbar.

The **Export Registers** dialog box appears.

Figure 3.58 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**.

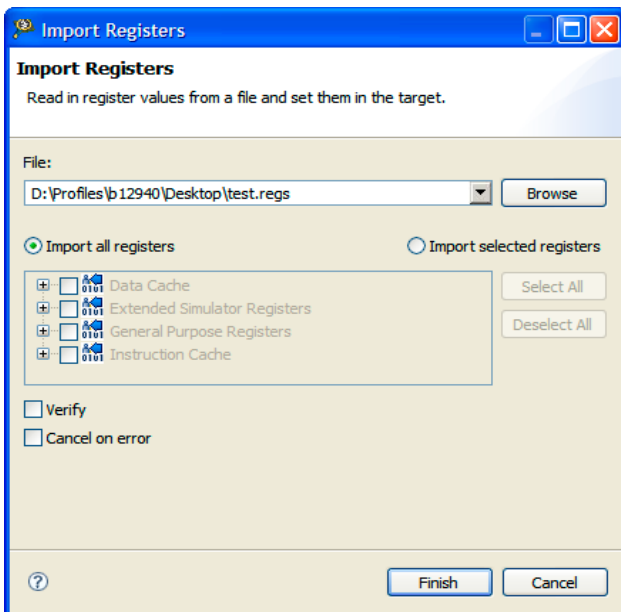
Importing Registers

To import register data from a file:

1. Open the **Registers** view ([Figure 3.56](#)).
2. Click the **Import registers** button  in the **Registers** view toolbar.

The **Import Registers** dialog box appears.

Figure 3.59 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**.

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.

Register Details Pane

The **Register Details** pane ([Figure 3.60](#)) shows detailed information for a selected register. 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 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.

Debugger

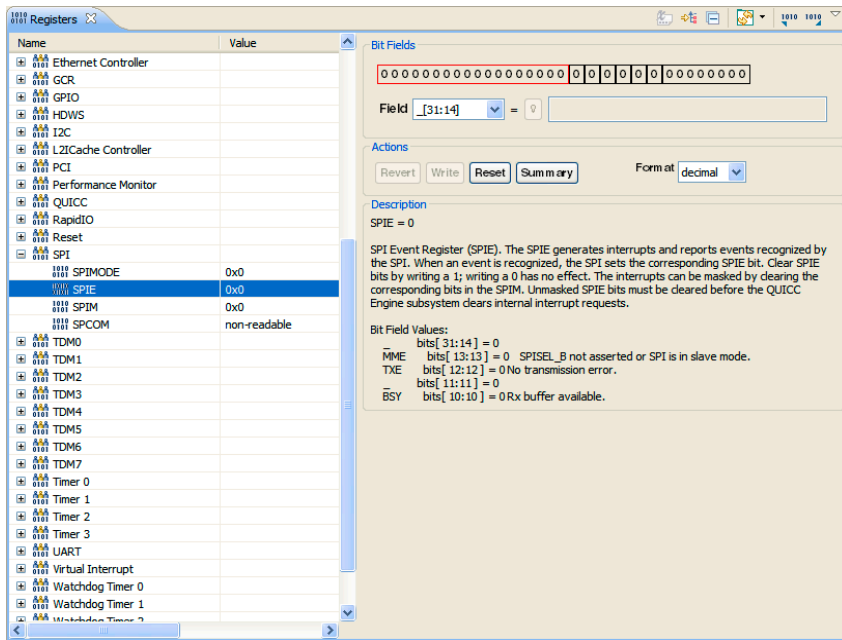
Register Details Pane

- 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 3.60 Register Details View



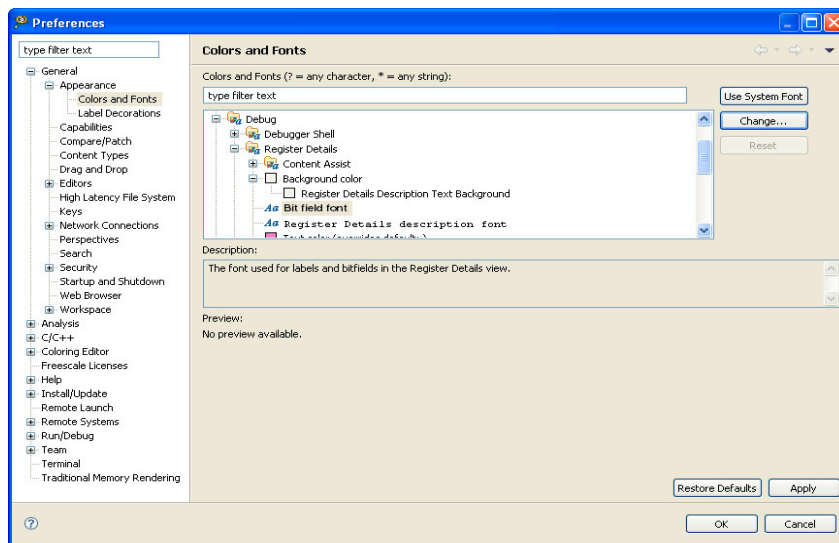
NOTE If the **Registers** view loses focus, all pending changes are discarded. For more information, see the *<Product> Targeting Manual*.

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 3.61 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**.

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:

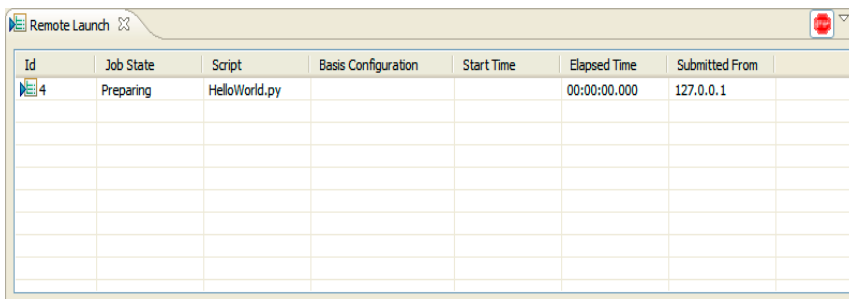
- The submissions web page
- Java and/or Python Clients

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.

The **Remote Launch** view ([Figure 3.62](#)) 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.

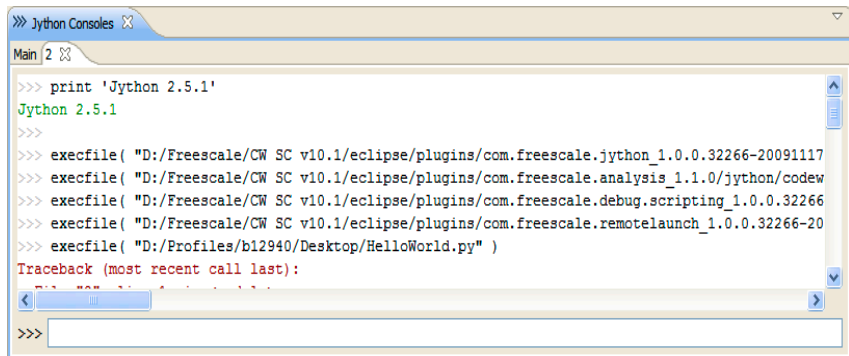
Figure 3.62 Remote Launch View



Id	Job State	Script	Basis Configuration	Start Time	Elapsed Time	Submitted From
4	Preparing	HelloWorld.py			00:00:00.000	127.0.0.1

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 3.63 Jython Consoles View



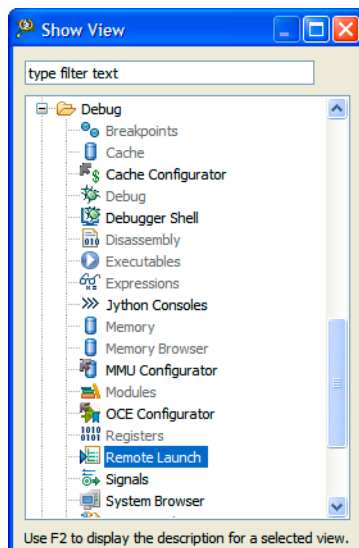
Remote Launch View

To display the **Remote Launch** view:

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

The **Show View** dialog box ([Figure 3.64](#)) appears.

Figure 3.64 Show View Dialog Box



2. Expand the **Debug** tree control.
3. Select **Remote Launch**.

4. Click **OK**.

Stack Crawls

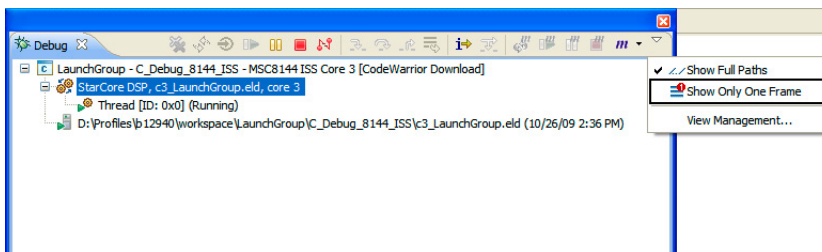
CodeWarrior allows you to limit the depth of stack crawls in the debugger. You can limit the stack crawl depth in two ways:

- [One Frame Mode](#)
- [Global Preference](#)

One Frame Mode

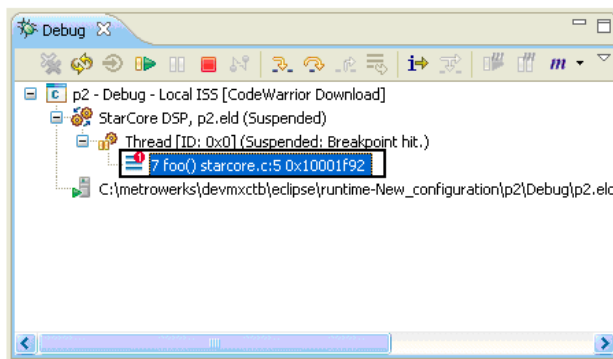
In the one frame mode, only the topmost frame is retrieved by the debugger engine and displayed in the **Debug** view and in the debugger shell. [Figure 3.65](#) shows selecting the one-frame mode from the **Debug** view.


Figure 3.65 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. [Figure 3.66](#) shows the stack crawl in a one frame mode.

Figure 3.66 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.

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:

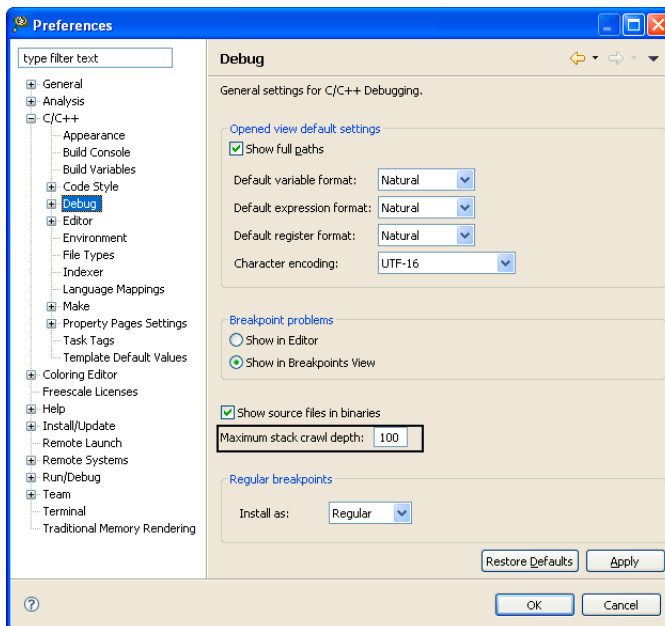
1. Select **Window -> Preferences** from the IDE menu bar.

The **Preferences** dialog box appears.

2. Expand the **C/C++** group and select **Debug** group.

General C/C++ debug settings ([Figure 3.67](#)) appears in the left-panel of the **Preferences** dialog box.

Figure 3.67 Preferences Dialog Box



3. Type the maximum frame depth in the **Maximum stack crawl depth** text box.

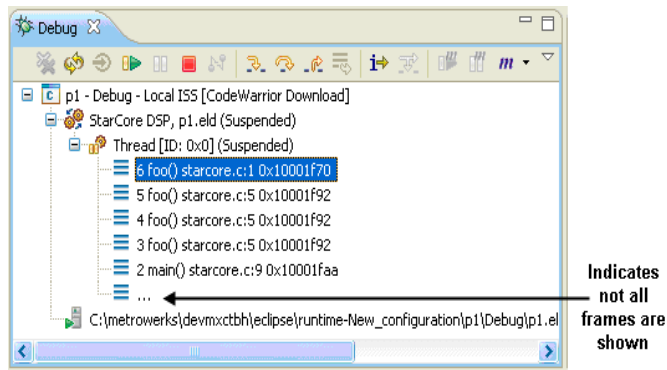
NOTE The upper limit for maximum frame depth is 100.

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 3.68](#)). This label indicates that frames are being omitted from display.

Figure 3.68 Exceeding Stack Crawl Depth



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.

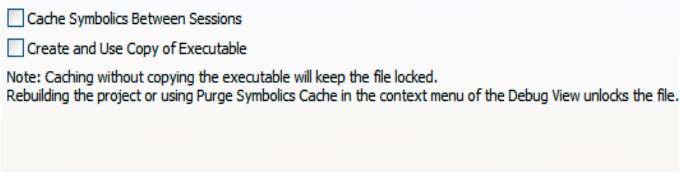
2. Expand the **CodeWarrior Download** configurations.
3. From the expanded list, select the debug configuration that you want to modify.
4. Select the **Debugger** tab to view the corresponding debugger settings page.

Debugger

System Browser View

5. Select the **Symbolics** tab in the **Debugger Options** group on the page.
The **Symbolics** page ([Figure 3.69](#)) appears.

Figure 3.69 Symbolics Page



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.

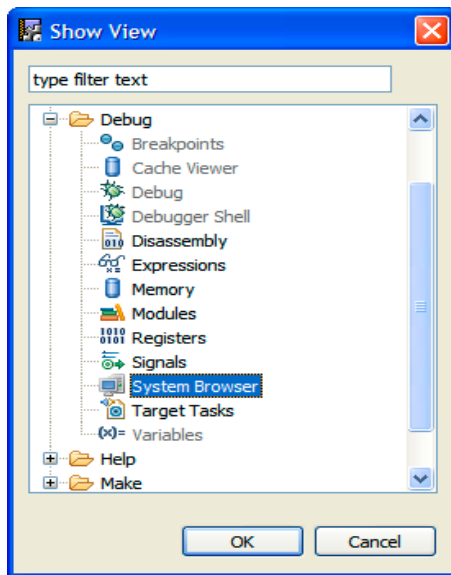
Opening the System Browser View

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 ([Figure 3.70](#)) appears.

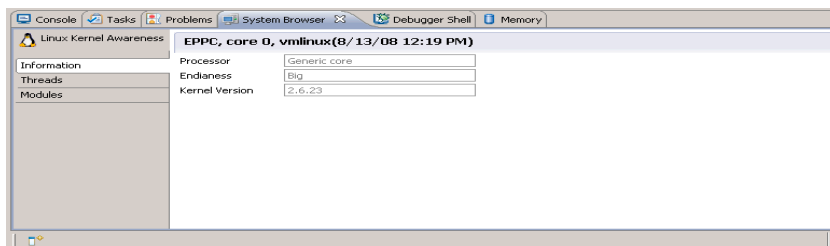
Figure 3.70 Show View Dialog Box



3. Expand the **Debug** group.
4. Select **System Browser**.
5. Click **OK**.

The **System Browser** view ([Figure 3.71](#)) appears.

Figure 3.71 System Browser View



NOTE The **System Browser** view shows information only when there is an OS running on the target being debugged.

Target Connection Lost

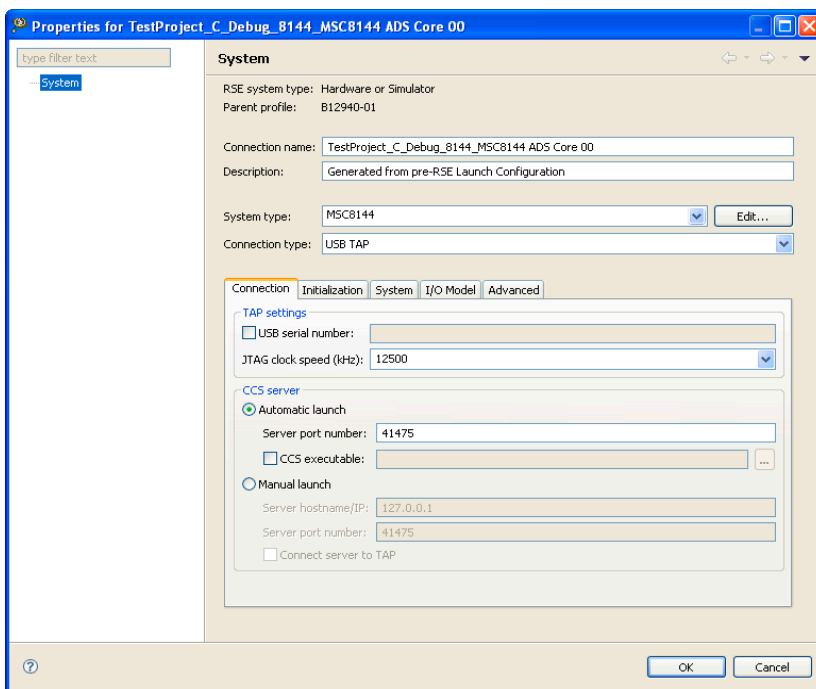
This feature enables you to configure the debugger's behavior when connection to the target is lost, such as low power modes, target power switched off, target changed communication speed, or disconnected run control. This feature enables you to configure the debugger to close the connection or automatically reconnect with a specified timeout value.

To configure target connection lost settings for debugger:

1. Open Remote System view.
2. Right-click a remote system name and select **Properties** from the context menu.

The **Properties for <Remote System>** dialog box ([Figure 3.12](#)) appears.

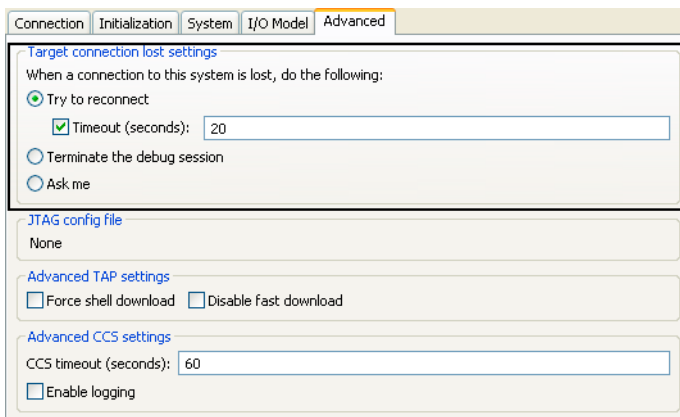
Figure 3.72 Properties for <Remote System> dialog box



3. In the right-pane, select **Advanced** tab.

The advanced connection settings appear under the **Advanced** tab.

Figure 3.73 Advanced Settings



4. Specify the target connection lost settings to suit your needs.
 - **Try to reconnect** — Select this option to specify timeout interval.
 - **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**.

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).

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.

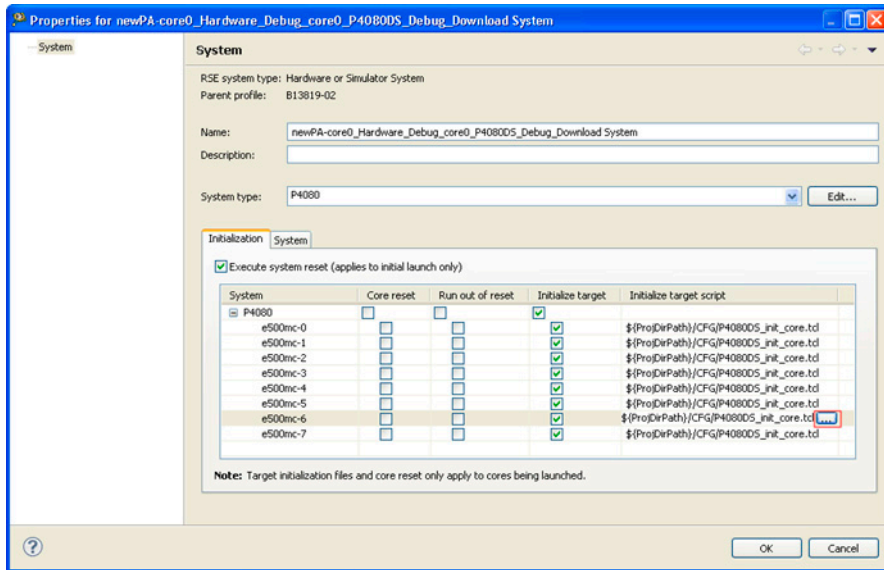
The debugger executes the commands in the target initialization file using the target connection protocol, such as a JTAG run-control device.

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 [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 **System type** drop-down list.
The Properties for <remote system> window ([Figure 3.74](#)) appears.
4. Select **initialization** tab.

Figure 3.74 Properties for <Remote System> — Initialization Settings

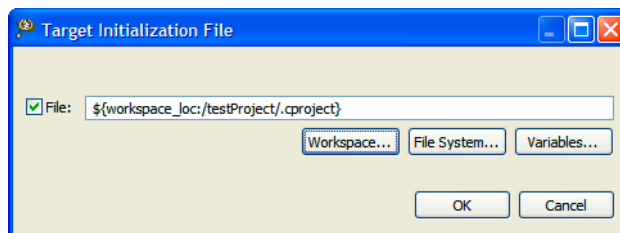


5. Click the **ellipsis** button in the **Initialize target script** column corresponding to the core for which you want to select the target initialization file.

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 (Figure 3.75) appears.

Figure 3.75 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**.

TAP Remote Connections

The CodeWarrior and Ethernet TAP and USB TAP hardware use emulation technology to control and provide visibility into your target system. They let you control and debug software running in-target, with minimal intrusion into target operation. You use the OnCE connector on your target hardware to interface with the TAP hardware.

Creating an Ethernet TAP Remote Connection

Before you can debug programs using the Ethernet TAP, you must create a remote connection for it. To create an Ethernet TAP remote connection:

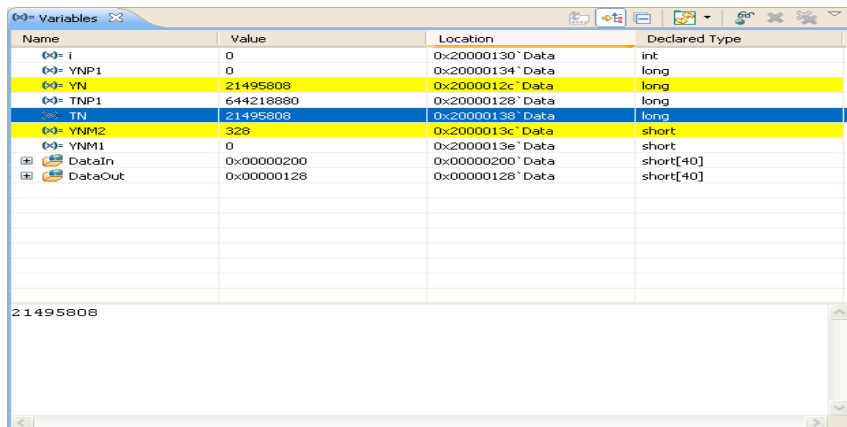
1. Open the CodeWarrior project that you want to configure to use the remote connection.
2. Select **Run > Debug** from the IDE menu bar.
The **Debug Configuration** dialog box appears. The left side of this window has a list of debug configurations that apply to the current application.
3. Expand CodeWarrior Download configuration.
4. From the expanded list, click the name of the debug configuration for which you want to modify debugger settings.
5. The **Debug Configuration** dialog box shows the settings for the configuration that you selected.
6. Click the Debugger tab.
7. Use the **Target Processor** drop-down list to specify MSC8144 ADS or MSC8144 AMC-S.
8. In the Debugger Options group, click **Connection** tab.
9. In the Physical connection pane, Select Ethernet Tap from the Connection drop-down list. Make sure to provide the IP Address in the **Hostname/IP Address** box.
10. Click **Apply**.

NOTE In the same way you can create a USB Tap Connection.

Variables

The **Variables** view (Figure 3.76) 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 3.76 Variables View



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**.
- 5.

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

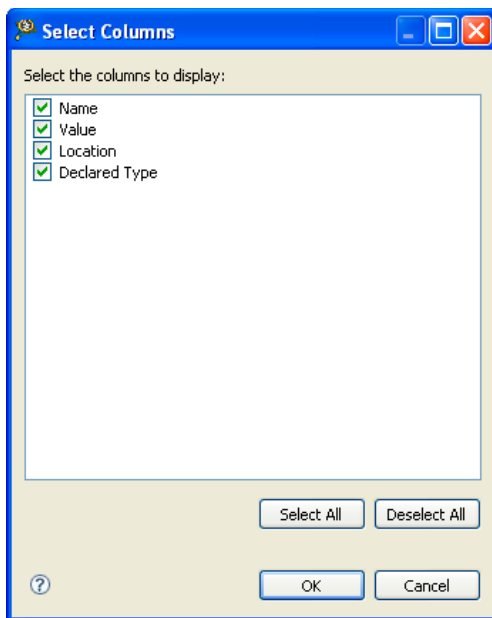
("0x...") value if the variable is in memory; if it is in a register, \$<register-name> will be returned.

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 ([Figure 3.77](#)) appears.

Figure 3.77 Select Columns



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.

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

Fractional Variable Formats

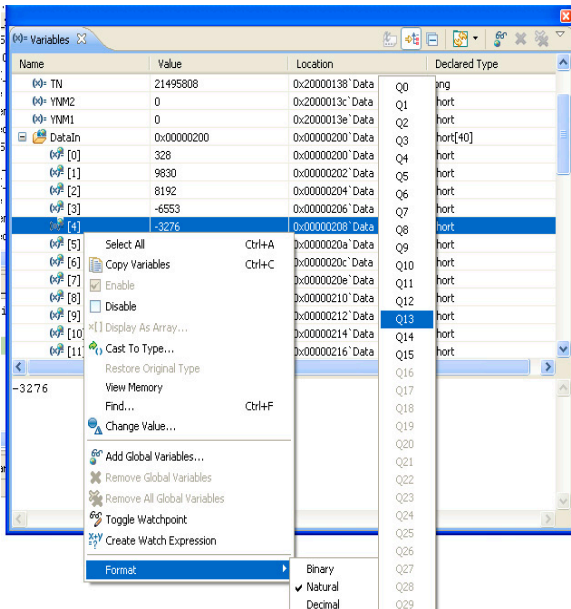
In addition to the Natural, Binary, Decimal, and Hexadecimal variable formats, CodeWarrior supports an additional class of custom fractional formats called *Qn*. *Qn* is a floating point representation of a fractional or fixed point number where n signifies the number of fractional bits (the number of 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 [“Opening the Variables View” on page 241](#).
2. Right-click a variable in the **Variable** view.
The context menu appears.

Figure 3.78 Selecting Fractional Format



3. Select **Format > Custom > 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

Adding Global Variables

You can add global variables to the **Variables** view.

To add global variable:

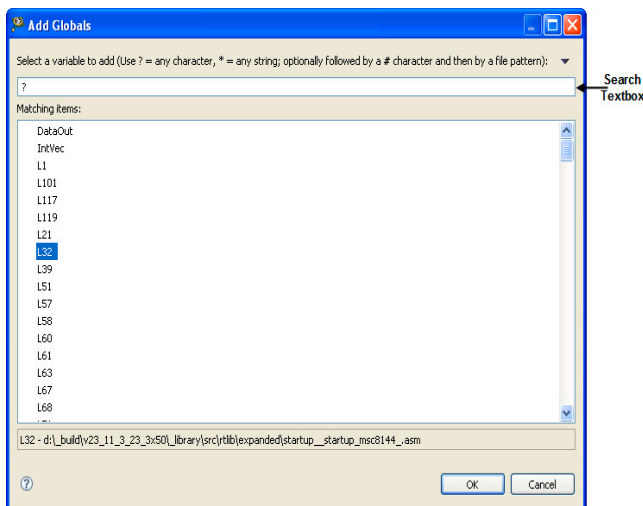
1. Select **Project > Debug** from the IDE menu bar.
A debugging session starts.

- In the **Variables view** toolbar, click the **Add Global Variables** button .

The **Add Globals** dialog box ([Figure 3.79](#)) appears.

TIP You can also add a global variable using the **Add Global Variable** command from the context menu.

Figure 3.79 Add Globals Dialog Box



- Specify a search criteria in the Search Textbox to filter the list of variables.
- 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.

- Click **OK**.

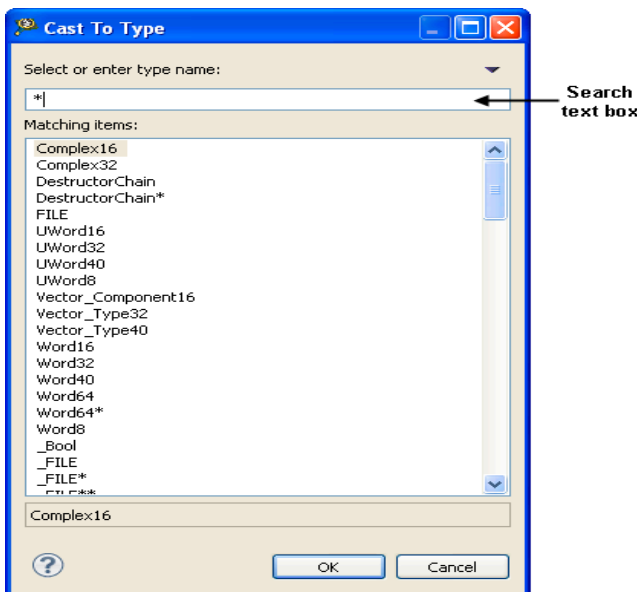
Cast to Type

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 [“Opening the Variables View” on page 241](#).
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 3.80 Cast to Type Dialog Box



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**.

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:

- Add Watchpoint dialog box
- Breakpoints view
- Memory view
- Variables view

NOTE Not all targets support setting a watchpoint on a memory range. For example, if a target has only one or two debug watch registers, you cannot set a watchpoint on 50 bytes.

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.

Setting a Watchpoint

Use the **Add Watchpoint** dialog box ([Figure 3.70](#)) 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 — an empty area inside the view.

- 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.

Creating Watchpoint

Use the **Add Watchpoint** dialog box ([Figure 3.70](#)) to create a watchpoint. The debugger sets the watchpoint according to the settings that you specify in this dialog box. [Table 3.9](#) describes each option.

Figure 3.81 Add Watchpoint Dialog Box

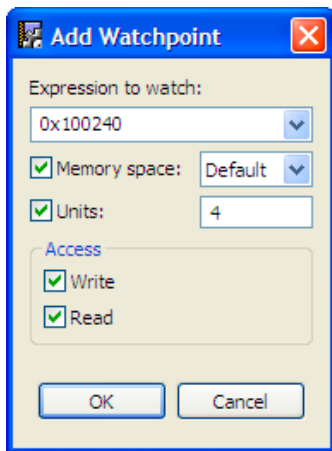


Table 3.9 Add Watchpoint Dialog Box Options

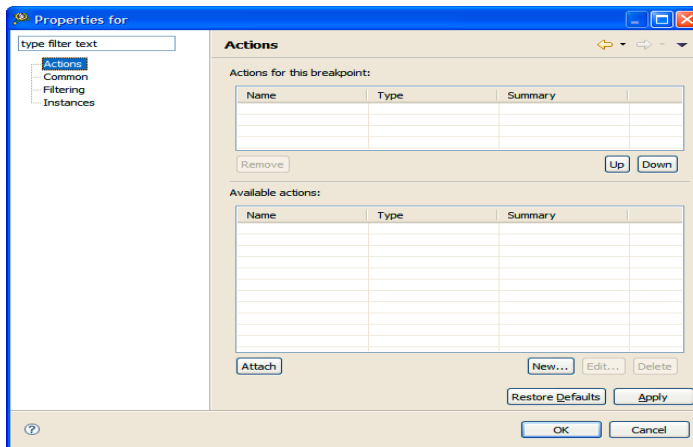
Option	Description
Expression to Watch	<p>Enter an expression that evaluates to an address on the target device. The debugger displays an error message when the specified expression evaluates to an invalid address.</p> <p>You can enter these types of expressions:</p> <ul style="list-style-type: none"> • 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. <p>The Add Watchpoints dialog box does not support entering expressions that evaluate to registers.</p>
Memory Space	<p>Check this box if you want to specify the memory space in which the watchpoint is set.</p> <p>The dropdown menu to the right of the check box lists each memory space available for the active debug context.</p> <p>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.</p>
Units	<p>Checked—Enter the number of addressable units that the watchpoint monitors.</p> <p>Cleared—Set the watchpoint on the entire range of memory occupied by the variable.</p>
Write	<p>Checked—The watchpoint monitors write activity on the specified memory space and address range.</p> <p>Cleared—The watchpoint does not monitor write activity.</p>
Read	<p>Checked—The watchpoint monitors read activity on the specified memory space and address range.</p> <p>Cleared—The watchpoint does not monitor read activity.</p>

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 ([Figure 3.82](#)) appears.

Figure 3.82 Properties for Dialog Box

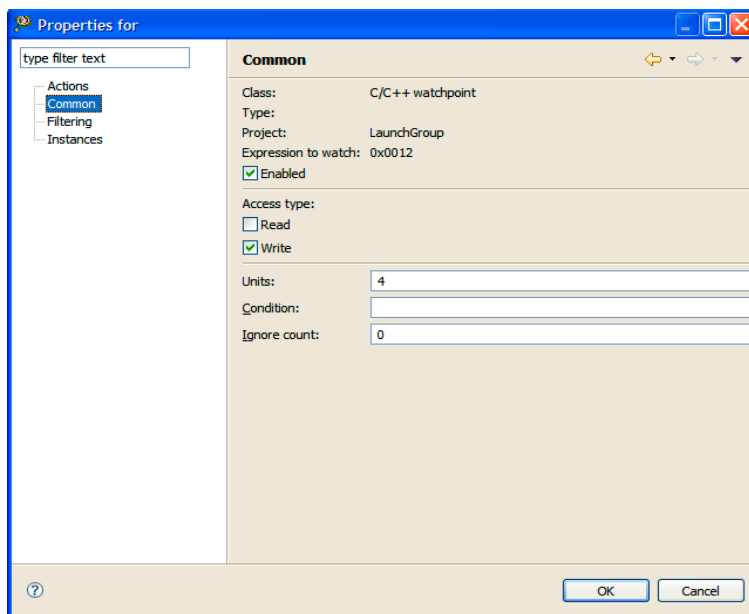


Modifying Watchpoint Properties

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 ([Figure 3.82](#)) appears.
3. Select **Common** from the left pane of the **Properties for C/C++ Watchpoint** window.
The right pane ([Figure 3.83](#)) displays the common properties for the watchpoint.

Figure 3.83 Common Watchpoint Properties



4. Edit the values in the fields.
5. Click **OK**.

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.

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.

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.

Remove All Watchpoints

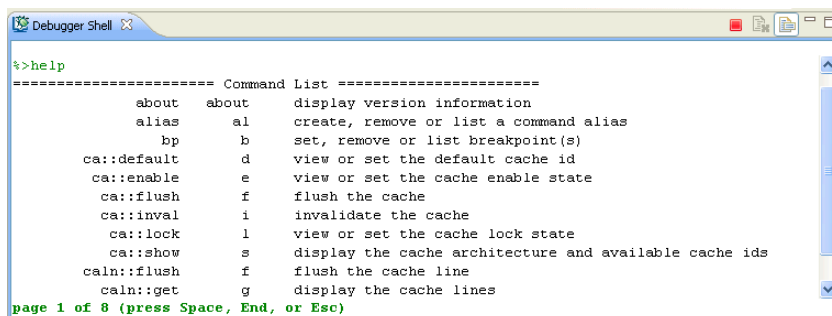
To remove all watchpoints, right-click inside the **Breakpoints** view and select **Remove All** from the context menu that appears. The **Breakpoints** view reflects your changes.

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.

You use the **Debugger Shell** view (Figure 4.1) 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



```
Debugger Shell
%>help
===== Command List =====
      about  about  display version information
      alias  al     create, remove or list a command alias
      bp     b     set, remove or list breakpoint(s)
      ca::default d   view or set the default cache id
      ca::enable e   view or set the cache enable state
      ca::flush f   flush the cache
      ca::inval i   invalidate the cache
      ca::lock l   view or set the cache lock state
      ca::show s   display the cache architecture and available cache ids
      caln::flush f  flush the cache line
      caln::get g  display the cache lines
page 1 of 8 (press Space, End, or Esc)
```

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**.

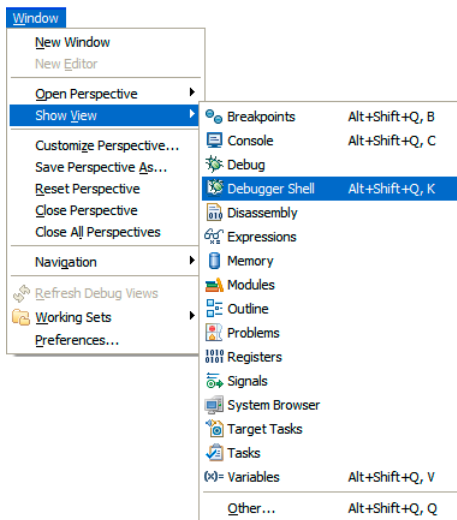
The **Debugger Shell** view appears ([Figure 4.1](#)).

NOTE Alternatively, select **Window > Show View > Other**. Expand the **Debug** tree control in the **Show View** dialog box, select **Debugger Shell**, and click **OK**.

Debugger Shell

Executing Previously Issued Commands


Figure 4.2 Show View - Debugger Shell



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.

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.

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.

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.

Debugger Shell

Debugger Shell Commands

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

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

[Table 4.1](#) lists and defines examples of the `alias` command.

Table 4.1 alias Command-Line Debugger Command — Examples

Command	Description
alias	Lists current aliases.

Table 4.1 alias Command-Line Debugger Command — Examples

Command	Description
<code>alias ls dir</code>	Issue the <code>dir</code> command when <code>ls</code> is typed.
<code>alias ls</code>	Remove the alias <code>ls</code> .

bp

Sets a breakpoint, removes a breakpoint, or lists the current breakpoints.

Syntax

```
bp
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 <count>}
bp #<id> cond <c-expr>
```

Examples

[Table 4.2](#) lists and defines examples of the `bp` command.

Table 4.2 bp Command-Line Debugger Command — Examples

Command	Description
<code>bp</code>	Lists all breakpoints.
<code>bp -hw fn</code>	Set hardware breakpoint at function <code>fn()</code> .
<code>bp -auto</code> <code>file.cpp 101 1</code>	Set an auto breakpoint on file <code>file.cpp</code> at line 101, column 1.
<code>bp -auto</code> <code>file.cpp 101</code> <code>"int</code> <code>foo<int>()" 1</code>	Set an auto breakpoint on file <code>file.cpp</code> at line 101 on function template instance <code>"int foo<int>()"</code> , column 1.
<code>bp fn off</code>	Remove the breakpoint at function <code>fn()</code> .

Debugger Shell

Debugger Shell Commands

Table 4.2 bp Command-Line Debugger Command — Examples

Command	Description
<code>bp 10343</code>	Set a breakpoint at memory address 10343.
<code>bp #4 off</code>	Remove the breakpoint number 4.
<code>bp #4 disable</code>	Disable the breakpoint number 4.
<code>bp #4 ignore 3</code>	Set ignore count to 3 for breakpoint number 4.
<code>bp #4 cond x == 3</code>	Set the condition for breakpoint number 4 to fire only if <code>x == 3</code> .
<code>bp #4.1 off</code>	Remove the breakpoint instance number 4.1.
<code>bp #4.1 ignore 3</code>	Set ignore count to 3 for breakpoint instance number 4.1.
<code>bp #4.1 cond x == 3</code>	Set the condition for breakpoint instance number 4.1 to fire only if <code>x == 3</code> .

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

[Table 4.3](#) lists and defines examples of the `cd` command.

Table 4.3 cd Command-Line Debugger Command — Examples

Command	Description
cd	Display current directory.
cd c:	Changes to the C: drive root directory.
cd d:/mw/0622/ test	Changes to the specified D: drive directory
cd c:p*s	Changes to any C: drive directory whose name starts with p and ends with s .

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>] <value>
      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

[Table 4.4](#) lists and defines parameters of the change command.

Debugger Shell

Debugger Shell Commands

Table 4.4 change Command-Line Debugger Command — Parameters

Command	Description
<ms>	On architectures supporting multiple memory spaces, specifies the memory space in which <addr> 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 "config MemIdentifier" is used.
<addr>	Target address in hex format.
<count>	Number of memory cells.
x<cell-size>	Memory is displayed in units called cells, where each cell consists of <cell-size> bytes. If unspecified, the setting "config MemWidth" is used.
h<access-size>	Memory is accessed with a hardware access size of <access-size> bytes. If unspecified, the setting "config MemAccess" is used.
{8,16,32,64}bit	Sets both <cell-size> and <access-size>.
<a1>{..<a2> #<n>}	Specifies a range of memory either by two endpoints, <a1> and <a2>, or by a startpoint and a byte count, <a1> and <n>. This alternate syntax is provided mainly for backwards compatibility. The new form of <addr> and <count> should be easier to use and thus preferred.
{r nr}	If multiple registers are specified, then the prefix r: causes a recursive, depth-first traversal of the register hierarchy. The prefix nr: prevents recursion. If unspecified, recursion is the default. Note that different levels of the register hierarchy are represented in the manner of a path with forward-slashes '/' used to delimit the register groups. A name that contains a slash itself can be represented with an escape backward-slash '\' followed by the forward-slash. Further note that a backslash in a doubly-quoted Tcl string is itself an escape character -- in this case two backslashes are required. Alternatively, you may use curly braces '{' and '}' to denote your string in which case just one backslash is necessary.
<reg>	A register name or a register group name.
..<reg>	The end point for a range of registers to access.
<n>	Number of registers.

Table 4.4 change Command-Line Debugger Command — Parameters

Command	Description
all	Specifies all registers.
v:	If this option appears with no <var> following it, then all variables pertinent to the current scope are printed.
<var>	Symbolic name of the variable to print. Can be a C expression as well.
v	This alternate syntax is provided mainly for backward compatibility.
-sl-ns	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 "config MemSwap" is used.
%<conv>	Specifies the type of the data. Possible values for <conv> are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.
%x	Hexadecimal.
%d	Signed decimal.
%u	Unsigned decimal.
%f	Floating point.
%[E<n>]F	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".
%s	ASCII.

Examples

The examples assume the following settings:

- radix x
- config MemIdentifier 0
- config MemWidth 32
- config MemAccess 32
- config MemSwap off

Debugger Shell

Debugger Shell Commands

[Table 4.5](#) lists and defines **Memory** examples of the `change` command.

Table 4.5 change Command-Line Debugger Command — Memory Examples

Command	Description
<code>change 10000 10</code>	Change memory range 0x10000-3 to 0x10 (because radix is hex).
<code>change 1:10000 20</code>	Change memory range 0x10000-3, memory space 1, to 0x20.
<code>change 10000 16 20</code>	Change each of 16 cells in the memory range 0x10000-3f to 0x20.
<code>change 10000 16x1h8 31</code>	Change each of 16, 1-byte cells to 0x31, using a hardware access size of 8-bytes per write.
<code>change 10000 - s %d 200</code>	Change memory range 0x10000-3 to c8000000.

[Table 4.6](#) lists and defines **Register** examples of the `change` command.

Table 4.6 change Command-Line Debugger Command — Register Examples

Command	Description
<code>change R1 123</code>	Change register R1 to 0x123.
<code>change R1..R5 5432</code>	Change registers R1 through R5 to 0x5432.
<code>change "General Purpose Registers/R1" 100</code>	Change register R1 in the General Purpose Register group to 0x100.

[Table 4.7](#) lists and defines **Variable** examples of the `change` command.

Table 4.7 change Command-Line Debugger Command — Variable Examples

Command	Description
<code>change myVar 10</code>	Change the value of variable <code>myVar</code> to 16 (0x10)

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. [Table 4.8](#) summarizes cache commands.

Table 4.8 Global Cache Commands

Command	Description
<code>cmdwin::ca::default</code>	Set specified cache as default
<code>cmdwin::ca::enable</code>	Enable/disable cache
<code>cmdwin::ca::flush</code>	Flushes cache
<code>cmdwin::ca::inval</code>	Invalidates cache
<code>cmdwin::ca::lock</code>	Lock/Unlock cache
<code>cmdwin::ca::show</code>	Show the architecture of the cache

Syntax

```
command [<cache ID>] [on | off]
```

Parameters

<cache ID>

Selects the cache that the command affects.

[on | off]

Changes a cache state.

Debugger Shell

Debugger Shell Commands

cmdwin::caln

The `cmdwin::caln` commands manage cache line operations. They affect memory elements within a designated cache. [Table 4.9](#) summarizes these commands.

Table 4.9 Cache Line Commands

Command	Description
<code>cmdwin::caln::get</code>	Display cache line
<code>cmdwin::caln::flush</code>	Flush cache line
<code>cmdwin::caln::inval</code>	Invalidate cache line
<code>cmdwin::caln::lock</code>	Locks/unlocks cache line
<code>cmdwin::caln::set</code>	Writes specified data to cache line

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

[Table 4.15](#) lists and defines examples of the `cmdwin::caln` commands.

Table 4.10 copy Command-Line Debugger Command — Examples

Command	Description
<code>cmdwin::caln:get 2</code>	Displays the second cache line.
<code>cmdwin::caln:flush 2</code>	flushes line 2 of the default cache.
<code>cmdwin::caln:set 2 = 0 1 1 2 3 5 8 13</code>	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.

config

Configures the command window.

Syntax

```
config <option> [<sub-option>] <value>
```

```
config
```

Options

<none>

With no options, config displays the current configuration settings.

[Table 4.11](#) lists and defines Display options of the config command.

Table 4.11 config Command-Line Debugger Command — Display Options

Command	Description
<code>hexPrefix <prefix></code>	Sets the string to be used as the prefix for hex values.
<code>binPrefix <prefix> 3</code>	Sets the string to be used as the prefix for binary values.
<code>showCommas off on</code>	When set on, decimal data is displayed with commas inserted every three digits. Hex and binary data is displayed with a colon inserted every four digits.

Debugger Shell

Debugger Shell Commands

Table 4.11 config Command-Line Debugger Command — Display Options

Command	Description
hexPadding on off	When set on, hex values are padded with leading zeros.
decPadding on off	When set on, decimal values are padded with leading zeros.
mem/Identifier <mem-space-id>	Sets the string to be used for the main memory space prefix.
memCache off on	With memCache off, the Command Window will always read target memory. This setting is useful if your target memory may change while the target is paused. With memCache on, the Command Window will cache target memory reads while your target is paused. This setting will improve the performance of the Command Window.
memReadMax <max-bytes>	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.
memSwap off on	When set, memory values are swapped on cell boundaries by default.
memWidth <bits> factory	Specifies the default width for display of memory data. Initially, the default width may vary depending on the active debugger; once the user has changed the value, the new value is used for all active debuggers. The initial behavior can be restored with the keyword "factory".
memAccess <bits>	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.
memFixedIntBits s <bits>	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 $[-256, 256)$.

Table 4.11 config Command-Line Debugger Command — Display Options

Command	Description
<pre>variable <sub- option> [on off]</pre>	<p>Enables or disables certain fields in the output of the "evaluate" command. If neither on nor off are specified, then the field is enabled. Possible values for <sub-option> are:</p> <ul style="list-style-type: none"> • echo - the variable name • location - the address of the variable • size - the size of the variable is bytes • type - the variable type
<pre>variable format <format></pre>	<p>Controls the output format of the "evaluate" command. Possible values for <format> are:</p> <ul style="list-style-type: none"> • - Default • d Signed • u Unsigned • x h Hex • c Char • s CString • p PascalString • f Float • e Enum • i Fixed Fract • b Binary Boolean SignedFixed • o w Unicode

[Table 4.12](#) lists and defines Run Control options of the `config` command.

Debugger Shell

Debugger Shell Commands

Table 4.12 Conf Command-Line Debugger Command — Run Control Options

Command	Description
<pre>autoThreadSwitch off interactive-only on</pre>	<p>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.</p> <p>If enabled, automatic thread switching is done in the following cases:</p> <ul style="list-style-type: none"> • 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.
<pre>debugTimeout <seconds></pre>	<p>The maximum amount of time to wait for a debug command to finish. You can also hit ESC to stop waiting.</p>
<pre>runControlSync off script-only on</pre>	<p>Sets how to synchronize run control commands. If set to "on", then all run control commands will wait until a thread stopped event. If set to "off", then all run control commands will return immediately. If set to "script-only", then all run control commands will wait while running a script but will return immediately while running interactively.</p>
<pre>setPCNextValidLine on off</pre>	<p>Controls the behavior of the setpc command in the case that the specified file line number has no source code. If set to "on", the PC is set to the next line number containing source. If set to "off", an error is shown.</p>

Examples

```
config
```

Display the current config status.

[Table 4.13](#) lists and defines Display examples of the `config` command.

Table 4.13 Conf Command-Line Debugger Command — Display Examples

Command	Description
<code>config hexPrefix 0x</code>	Show hexadecimal numbers with "0x" prefix.
<code>config ShowCommas on</code>	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.
<code>config HexPadding off</code>	Show hex and binary numbers with leading zeros, as in 0x0000. config MemIdentifier 0. Use "0" as the default memory space for memory commands.
<code>config MemCache off</code>	Turn off caching of target memory. AFFECTS COMMAND WINDOW ONLY.
<code>config MemReadMax 2048</code>	Limit memory commands to 2048 (decimal) bytes.
<code>config MemSwap on</code>	Swap memory on cell boundaries before accessing the target.
<code>config MemWidth 16</code>	Displays and writes 16bit values.
<code>config MemWidth factory</code>	Reset the MemWidth to factory settings.
<code>config MemAccess 8</code>	Uses an 8-bit access size for reading and writing target memory.
<code>config MemFixedIntBits 8</code>	Sets the fixed point range to $[-2^8, 2^8)$, or $[-256, 256)$.
<code>config var echo on</code>	Include the variable name in the output of the "evaluate" command.

Debugger Shell

Debugger Shell Commands

Table 4.13 Conf Command-Line Debugger Command — Display Examples

Command	Description
<pre>config var format d</pre>	Set the default display format of the “evaluate” command to decimal. The format may be one of the following strings or the corresponding character abbreviation: Default('-'), Signed('d'), Unsigned('u'), Hex('h' 'x'), Char('c'), CString('s'), PascalString('p'), Float('f'), Enum('e'), Fixed('i'), Fract(no abbrev), Binary('b'), Boolean(no abbrev), SignedFixed(no abbrev), Unicode('o' 'w').
<pre>config var type off</pre>	Exclude the variable type name in the output of the “evaluate” command.
<pre>config var location on</pre>	Include the memory address in the output of the “evaluate” command.
<pre>config var size on</pre>	Include the variable size in the output of the “evaluate” command.

[Table 4.14](#) lists and defines Run Control examples of the `config` command.

Table 4.14 Conf Command-Line Debugger Command — Run Control Examples

Command	Description
<pre>config AutoThreadSwitch interactive-only</pre>	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.
<pre>config DebugTimeout 10</pre>	Wait up to 10 seconds for debug command to finish.
<pre>config RunControlSync on</pre>	Run control commands will wait for thread-stopped event.
<pre>config SetPCNextValid Line on</pre>	If <code>setpc</code> is called for a file line number with no source code, the line number is automatically increased to the next line with source code.

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

[Table 4.15](#) lists and defines examples of the `copy` command.

Table 4.15 copy Command-Line Debugger Command — Examples

Command	Description
<code>copy 00..1f 30</code>	Copy memory addresses 00 through 1f to address 30.
<code>copy 20#10 50</code>	Copy 10 memory locations beginning at memory location 20 to memory beginning at location 50.

debug

Launches a debug session.

Syntax

```
debug [[-index] <index> | [-name] <debug-config-name>]
```

Examples

[Table 4.16](#) lists and defines examples of the `debug` command.

Debugger Shell

Debugger Shell Commands

Table 4.16 debug Command-Line Debugger Command — Examples

Command	Description
<code>debug</code>	Start debugging using the default launch configuration, which is the last debugged configuration if one exists and index 0 otherwise.
<code>debug -index 3</code>	Start debugging using the launch configuration at index 3. Type 'launch' for the current set of launch configurations.
<code>debug -name 3</code>	Start debugging using the launch configuration named '3'. Type 'launch' for the current set of launch configurations.
<code>debug 3</code>	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.
<code>debug {My Launch Config}</code>	Start debugging using the launch configuration named 'My Launch Config'. Type 'launch' for the current set of launch configurations.

dir

Lists directory contents.

Syntax

```
dir [path|files]
```

Examples

[Table 4.17](#) lists and defines examples of the `dir` command.

Table 4.17 dir Command-Line Debugger Command—Examples

Command	Description
<code>dir</code>	Lists all files of the current directory.
<code>dir *.txt</code>	Lists all current-directory files that have the <code>.txt</code> file name extension.

Table 4.17 dir Command-Line Debugger Command—Examples

Command	Description
<code>dir c:/tmp</code>	Lists all files in the tmp directory on the C: drive.
<code>dir /ad</code>	Lists only the subdirectories of the current directory.

disassemble

Disassembles the instructions of the specified memory block.

Syntax

```
disassemble
disassemble pc | [<ms>:]<addr> [<count>]
disassemble 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

[Table 4.18](#) lists and defines examples of the `disassemble` command.

Table 4.18 disassemble Command-Line Debugger Command—Examples

Command	Description
<code>disassemble</code>	Lists the next block of instructions.
<code>disassemble reset</code>	Reset the next block to the PC and the instruction count to one screenful.
<code>disassemble pc</code>	Lists instructions starting at the PC.
<code>disassemble pc 4</code>	Lists 4 instructions starting at the PC. Sets the instruction count to 4.
<code>disassemble 1000</code>	Lists instructions starting at address 0x1000 .
<code>disassemble p:1000 4</code>	Lists 4 instructions from memory space <code>p</code> , address 1000. Sets the instruction count to 4.

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.

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

[Table 4.19](#) lists and defines parameters of the `display` command.

Table 4.19 display Command-Line Debugger Command — Options

Command	Description
<ms>	On architectures supporting multiple memory spaces, specifies the memory space in which <addr> is to be found. Refer the help for the option <code>-ms</code> of <code>display</code> or <code>mem</code> for more information on memory spaces. If unspecified, the setting "config MemIdentifier" is used.
<addr>	Target address in hex format.
<count>	Number of memory cells.
x<cell-size>	Memory is displayed in units called cells, where each cell consists of <cell-size> bytes. If unspecified, the setting "config MemWidth" is used.
h<access-size>	Memory is accessed with a hardware access size of <access-size> bytes. If unspecified, the setting "config MemAccess" is used.
{8,16,32,64}bit	Sets both <cell-size> and <access-size>.
-np	Don't print anything to the display, only return the data.
-ms	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.
<a1>{.. <a2> #<n>}< td=""> <td>Specifies a range of memory either by two endpoints, <a1> and <a2>, or by a startpoint and a byte count, <a1> and <n>. This alternate syntax is provided mainly for backwards compatibility. The new form of <addr> and <count> should be easier to use and thus preferred.</td> </a2> #<n>}<>	Specifies a range of memory either by two endpoints, <a1> and <a2>, or by a startpoint and a byte count, <a1> and <n>. This alternate syntax is provided mainly for backwards compatibility. The new form of <addr> and <count> should be easier to use and thus preferred.

Debugger Shell

Debugger Shell Commands

Table 4.19 display Command-Line Debugger Command — Options

Command	Description
{r nr}	If multiple registers are specified, then the prefix r: causes a recursive, depth-first traversal of the register hierarchy. The prefix nr: prevents recursion. If unspecified, recursion is the default. Note that different levels of the register hierarchy are represented in the manner of a path with forward-slashes '/' used to delimit the register groups. A name that contains a slash itself can be represented with an escape backward-slash '\' followed by the forward-slash. Further note that a backslash in a doubly-quoted Tcl string is itself an escape character -- in this case two backslashes are required. Alternatively, you may use curly braces '{' and '}' to denote your string in which case just one backslash is necessary.
<reg>	A register name or a register group name.
..<reg>	The end point for a range of registers to access.
<n>	Number of registers.
all	Specifies all registers.
-d	Print detailed data book information.
-nr	Print only register groups, i.e. no registers.
-nv	Print only register groups and register names, i.e. no values.
-np	Don't print anything to the display, only return the data.
regset	Display the register group hierarchy.
v:	If this option appears with no <var> following it, then all variables pertinent to the current scope are printed.
<var>	Symbolic name of the variable to print. Can be a C expression as well.
-s -ns	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 "config MemSwap" is used.
%<conv>	Specifies the type of the data. Possible values for <conv> are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.

Table 4.19 display Command-Line Debugger Command — Options

Command	Description
<code>%x</code>	Hexadecimal.
<code>%d</code>	Signed decimal.
<code>%u</code>	Unsigned decimal.
<code>%f</code>	Floating point.
<code> %[E<n>]F</code>	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".
<code>%s</code>	ASCII.

Examples

The examples assume the following settings:

- radix x
- config MemIdentifier 0
- config MemWidth 32
- config MemAccess 32
- config MemSwap off

[Table 4.20](#) lists and defines examples of the `display` command.

Table 4.20 display Command-Line Debugger Command — Examples

Command	Description
<code>display 10000</code>	Display memory range 0x10000-3 as one cell.
<code>display 1:10000</code>	Display memory range 0x10000-3, memory space 1, as one cell.
<code>display 10000 16</code>	Display memory range 0x10000-3f as 16 cells.
<code>display 10000 16x1h8</code>	Display 16, 1-byte cells, with a hardware access size of 8-bytes per read.
<code>display 10000 8bit</code>	Display one byte, with a hardware access size of one byte.

Debugger Shell

Debugger Shell Commands

Table 4.20 display Command-Line Debugger Command — Examples (*continued*)

Command	Description
<code>display 10000 -np</code>	Return one cell, but don't print it to the Command Window .
<code>display 10000 -s</code>	Display one cell with the data endian-swapped.
<code>display 10000 %d</code>	Display one cell in decimal format.
<code>display -ms</code>	Display the available memory spaces, if any.
<code>display -regset</code>	List all the available register sets on the target chip.
<code>display R1</code>	Display the value of register R1.
<code>display "General Purpose Registers/R1"</code>	Display the value of register R1 in the General Purpose Register group.
<code>display R1 -d</code>	Display detailed "data book" contents of R1, including bitfields and definitions.
<code>display "nr:General Purpose Registers/R1" 25</code>	Beginning with register R1 , display the next 25 registers. Register groups are not recursively searched.
<code>display myVar -s %d</code>	Display the endian-swapped contents of variable myVar in decimal.

evaluate

Display variable or expression.

Syntax

```
evaluate [#<format>] [-1] [<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

[Table 4.21](#) lists and defines examples of the `evaluate` command.

Table 4.21 evaluate Command-Line Debugger Command — Examples

Command	Description
<code>evaluate</code>	List the types for all the variables in current and global stack.
<code>evaluate i</code>	Return the value of variable 'i'
<code>evaluate #b i</code>	Return the value of variable 'i' formatted in binary
<code>evaluate -l 10</code>	Return the address for line 10 in the current file
<code>evaluate -l myfile.c,10</code>	Return the address for line 10 in file myfile.c
<code>evaluate -l +10</code>	Return the address to an offset of 10 lines starting from the current line

Debugger Shell

Debugger Shell Commands

Table 4.21 evaluate Command-Line Debugger Command — Examples (*continued*)

Command	Description
<code>evaluate -l myfile.c:mysym bol</code>	Return the address of the symbol 'mysymbol' defined in file 'myfile.c'.
<code>evaluate -l mysymbol</code>	Return the address of the global symbol 'mysymbol'.
<code>evaluate -l mysymbol +10</code>	Return the address of the 10'th line belonging to the global symbol 'mysymbol'.
<code>evaluate -l myfile.c:mysym bol</code>	Return the address of the local symbol 'mysymbol' defined in the file 'myfile.c'.
<code>evaluate -l myfile.c:mysym bol 10</code>	Return the address of the 10'th line belonging to the local symbol.

finish

Execute until the current function returns.

Syntax

`finish`

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

[Table 4.22](#) lists and defines examples of the `fl::blankcheck` command.

Table 4.22 fl::blankcheck Command-Line Debugger Command — Examples

Command	Description
<code>blankcheck all</code>	Test if the flash device is in the blank state. All sectors will be tested regardless of the enabled list maintained by the "device" command.
<code>blankcheck list</code>	Test whether the sectors in the enabled list are in the blank state.

fl::checksum

Calculate a checksum.

Syntax

```
checksum [-host | -range <addr> <size> | -dev]
```

Options

[Table 4.23](#) lists and defines options of the `fl::checksum` command.

Table 4.23 fl::checksum Command-Line Debugger Command — Options

Command	Description
<code><none></code>	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.
<code>-host</code>	Calculate the checksum for the host image file contents corresponding to the settings of the <code>fl::image</code> command.

Debugger Shell

Debugger Shell Commands

Table 4.23 fl::checksum Command-Line Debugger Command — Options

Command	Description
-range	<addr> <size> Calculate the checksum for the target memory contents specified by a beginning address <addr> and number of bytes <size>, both given in hex. The target is defined by the fl::target command.
-dev	Calculate the checksum for the entire flash contents. The flash is defined by the fl::device command. The target is defined by the fl::target command.

Examples

```
checksum
```

Calculate a checksum.

fl::device

Define the flash device.

Syntax

```
device
```

```
device <setting> ...
```

```
device ls
```

```
device ls org [<dev>]
```

```
device ls sect [[<dev>] <org>]
```

Options

[Table 4.24](#) lists and defines options of the fl::device command.

Table 4.24 fl::device Command-Line Debugger Command — Options

Command	Description
<none>	With no options, lists the current settings.
<setting>	Used to set a configuration setting. Possible values are: -d <dev>, -o <org>, -a <addr> [<end>] , -se all <index> ... , -sd all <index> ...
-d <dev>	Set the device to <dev>.
-o <org>	Set the organization to <org>.
-a <addr> [<end>]	Set the start <addr> and optional end <end> address for the device, both given in hex.
-se all <index> ...	Enable sectors for "erase" and "blankcheck". Sectors are specified with a zero-based index.
-sd all <index> ...	Disable sectors for "erase" and "blankcheck". Sectors are specified with a zero-based index.
ls	Lists all the supported devices.
ls org [<dev>]	List the organizations for a particular device. The device may be specified with <dev>, otherwise the current device is used.
ls sect [[<dev>] <org>]	List the sectors for a particular device and organization. The organization may be specified with <org>, otherwise the current device and organization is used. If <org> is specified, the device may be specified with <dev>, otherwise the current device is used. If <dev> is specified, then 0 is used for the starting address; otherwise, the current device start address is used.

Examples

device

Lists the current settings.

fl::diagnose

Dump flash information.

Debugger Shell

Debugger Shell Commands

Syntax

`diagnose [full]`

Options

`full`

Also dump sector status (programmed/erased). This could take a few minutes for large flashes.

Examples

[Table 4.25](#) lists and defines examples of the `fl::diagnose` command.

Table 4.25 fl::diagnose Command-Line Debugger Command — Examples

Command	Description
<code>diagnose</code>	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.

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.

Syntax

```
fl::dump [all | -range start_addr end_addr] -o <file>
```

Parameter

-all

Dumps content of entire flash.

-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.

Examples

```
dump -all -o myfile -t "Binary/Raw Format"
```

Dump 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

[Table 4.26](#) lists and defines examples of the fl::erase command.

Debugger Shell

Debugger Shell Commands

Table 4.26 fl::erase Command-Line Debugger Command — Examples

Command	Description
erase all	<p>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 "device" command.</p> <p>Erase the device one sector at a time. All sectors will be erased regardless of the enabled list maintained by the "device" command.</p>
erase list	Erase the sectors in enabled list.
erase image	Erase the sectors occupied by the file defined with fl::image.

fl::image

Define the flash image settings.

Syntax

```
image
```

```
image <setting> ...
```

```
image ls
```

Options

[Table 4.27](#) lists and defines options of the fl::image command.

Table 4.27 fl::erase Command-Line Debugger Command — Options

Command	Description
<none>	With no options, lists the current settings.
<setting>	<p>Used to set a configuration setting. Possible values are:</p> <p>-f <file>, -t <type>, -re on off, -r <addr> [<end>], -oe on off, -o <offset></p>
-f <file>	Set the image file.

Table 4.27 fl::erase Command-Line Debugger Command — Options

Command	Description
-t <type>	Set the type of the image file. Possible values are shown by "image ls".
-re on off	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.
-r <addr> [<end>]	Set the start <addr> and optional end <end> address for the restricting flash access, both given in hex. The range must also be enabled by the option "-re".
-oe all <index> ...	If -oe is set to on, the offset setting of this command will be used.
-o	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.

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.

Debugger Shell

Debugger Shell Commands

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

[Table 4.28](#) lists and defines options of the `fl::target` command.

Table 4.28 fl::target Command-Line Debugger Command — Options

Command	Description
<none>	With no options, lists the current settings.
<setting>	Used to set a configuration setting. Possible values are: -c <conn>, -p <proc>, -ie on off , -i <initfile>, -b <addr> [<size>] , -v on off , -l on off
-lc <launch configuration name>	Set the launch configuration that will be used.
-c <conn>	Set the connection to <conn>.
-p <proc>	Set the processor to <proc>.

Table 4.28 fl::target Command-Line Debugger Command — Options

Command	Description
-ie on off	Enables or disables the initfile set by -i.
-i <initfile>	Set the target initialization file to <initfile>. Only used if -ie is on.
-b <addr> [<size>]	Set the target RAM buffer for downloading image data to begin at <addr> with <size> bytes, both given in hex.
-v on off	Set the target memory verification on or off.
-l on off	Enable or disable logging.
ls [p c]	List the supported processors and/or the available connections.

Examples

```
target
```

List the current settings.

fl::verify

Verify the flash device.

Syntax

```
verify
```

fl::write

Write the flash device.

Syntax

```
write
```

Debugger Shell

Debugger Shell Commands

funcs

Displays information about functions.

Syntax

```
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

```
getpid
```

go

Starts to debug your program from the current instruction.

Syntax

```
go [nowait | <timeout_s>]
```

Parameter

<none>

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

[Table 4.29](#) lists and defines examples of the `go` command.

Table 4.29 go Command-Line Debugger Command — Examples

Command	Description
<code>go</code>	Run the default thread.
<code>go nowait</code>	Run the default thread without waiting for a thread break event.
<code>go 5</code>	Run the default thread. If <code>config runControlSync</code> is enabled, then the command will wait for a thread break event for a maximum of 5 seconds.

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

[Table 4.30](#) lists and defines examples of the `help` command.

Debugger Shell

Debugger Shell Commands

Table 4.30 help Command-Line Debugger Command — Examples

Command	Description
help	Lists all debug commands.
help b	Lists help information for the break 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

[Table 4.31](#) lists and defines examples of the `jtagclock` command.

Table 4.31 jtagclock Command-Line Debugger Command — Examples

Command	Description
jtagclock 3	Read the current jtag clock speed for chain position 3.
jtagclock 3 1000	Update the jtag clock speed to 1000kHz for chain position 3.

kill

Close the specified debug session.

Syntax

```
kill [all | <index> ...]
```

Examples

[Table 4.32](#) lists and defines examples of the `kill` command.

Table 4.32 kill Command-Line Debugger Command — Examples

Command	Description
<code>kill</code>	Kills the debug session for the current process.
<code>kill all</code>	Kills all active debug sessions.
<code>kill 0 1</code>	Kills debug sessions 0 and 1.

launch

Lists the launch configurations.

Syntax

```
launch
```

Examples

```
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.

Debugger Shell

Debugger Shell Commands

Syntax

```
displaylinuxlist -list <listName> -function
    <functionWhereListIsVisible> -address <listAddress> -
    type <elementTypeName> [-next <nextPath>]
```

Options

[Table 4.33](#) lists and defines options of the `linux::displaylinuxlist` command.

Table 4.33 linux::displaylinuxlist Command-Line Debugger Command — Options

Command	Description
-l[ist] <listName>	The name of the list (must be global).
-f[unction] <functionWhere ListIsVisible>	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[ype] <elementTypeNa me>	The type of the list elements.
-n[ext] <nextPath>	Specify in order all the structure member names needed to reach the next element.

Examples

[Table 4.34](#) lists and defines examples of the `linux::displaylinuxlist` command.

Table 4.34 linux::displaylinuxlist Command-Line Debugger Command — Examples

Command	Description
<pre>linux::displaylinuxlist - list workqueues -function __create_workqueue -type workqueue_struct -next list next</pre>	Lists the current workqueues.
<pre>linux::displaylinuxlist - address 0xC00703fc -type workqueue_struct -next list next</pre>	List the current workqueues, 0xC00703fc should be the address of workqueues. Available only if the kernel is stopped.

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>
```

Debugger Shell

Debugger Shell Commands

linux::unloadsymbolics

Unloads the symbolics for the specified module.

Syntax

```
unloadsymbolics
```

loadsym

Load a symbolic file.

Syntax

```
loadsym <filename> [PIC load addr (hex)]
```

Examples

[Table 4.35](#) lists and defines examples of the `loadsym` command.

Table 4.35 loadsym Command-Line Debugger Command — Examples

Command	Description
<code>loadsym myapp.elf</code>	Loads the debug information in <code>myapp.elf</code> into the debugger.
<code>loadsym mypicapp.elf 0x40000</code>	Loads the debug information in <code>mypicapp.elf</code> into the debugger; symbolics addresses are adjusted based on the alternate load address of <code>0x40000</code> .

log

Logs the commands or lists entries of a debug session. If issued with no parameters, the command lists all open log files.

Syntax

```
log c|s <filename>
log off [c|s] [all]
log
```

Parameter

c
Command specifier.

s
Lists entry specifier.

<filename>
Name of a log file.

Examples

[Table 4.36](#) lists and defines examples of the log command.

Table 4.36 log Command-Line Debugger Command — Examples

Command	Description
log	Lists currently opened log files.
log s session.log	Log all display entries to file session.log .
log off c	Close current command log file.
log off	Close current command and log file.
log off all	Close all log files.

mc::go

Resumes multiple cores.

Syntax

```
mc::go
```

Debugger Shell

Debugger Shell Commands

Remarks

`mc::go` resumes the selected cores associated with the current thread context (see [“switchtarget”](#)).

mc::kill

Terminate multiple cores.

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
```

Remarks

`mc::stop` stops the selected cores associated with the current thread context (see [“switchtarget”](#)).

mem

Read and write memory.

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

[Table 4.37](#) lists and defines options of the `mem` command.

Debugger Shell

Debugger Shell Commands

Table 4.37 mem Command-Line Debugger Command — Options

Command	Description
[none]	With no option, next block of memory is read.
<ms>	On architectures supporting multiple memory spaces, specifies the memory space in which <addr> 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 "config MemIdentifier" is used.
<addr>	Target address in hex.
<count>	Number of memory cells.
x<cell-size>	Memory is displayed in units called cells, where each cell consists of <cell-size> bytes. If unspecified, the setting "config MemWidth" is used.
h<access-size>	Memory is accessed with a hardware access size of <access-size> bytes. If unspecified, the setting "config MemAccess" is used.
{8,16,32,64}bit	Sets both <cell-size> and <access-size>.
-np	Don't print anything to the display, only return the data.
-ms	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.
-s -ns	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 "config MemSwap" is used.
%<conv>	Specifies the type of the data. Possible values for <conv> are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.
%x	Hexadecimal.
%d	Signed decimal.
%u	Unsigned decimal.

Table 4.37 mem Command-Line Debugger Command — Options

Command	Description
%f	Floating point.
%[E<n>]F	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".
%s	ASCII.

Examples

The examples assume the following settings:

- radix x
- config MemIdentifier 0
- config MemWidth 32
- config MemAccess 32
- config MemSwap off

[Table 4.38](#) lists and defines examples of the mem command.

Table 4.38 mem Command-Line Debugger Command — Examples

Command	Description
mem	Display the next block of memory.
mem 10000	Change memory range 0x10000-3 as one cell.
mem 1:10000	Change memory range 0x10000-3, memory space 1, as one cell.
mem 10000 16	Display memory range 0x10000-3f as 16 cells.
mem 10000 16x1h8	Display 16, 1-byte cells, with a hardware access size of 8-bytes per read.
mem 10000 8bit	Display one byte, with a hardware access size of one byte.
mem 10000 -np	Return one cell, but don't print it to the Command Window.
mem 10000 -s	Display one byte with the data endian-swapped.
mem 10000 %d	Display one cell in decimal format.
mem -ms	Display the available memory spaces, if any.

Debugger Shell

Debugger Shell Commands

Table 4.38 mem Command-Line Debugger Command — Examples

Command	Description
<code>mem 10000 =10</code>	Change memory range 0x10000-3 to 0x10 (because radix is hex).
<code>mem 1:10000 =20</code>	Change memory range 0x10000-3, memory space 1, to 0x20.
<code>mem 10000 16x1h8 =31</code>	Change each of 16, 1-byte cells to 0x31, using a hardware access size of 8-bytes per write.
<code>mem 10000 -s %d =200</code>	Change memory range 0x10000-3 to c8000000.

next

Runs to next source line or assembly instruction in current frame.

Syntax

`next`

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

[Table 4.39](#) lists and defines examples of the `oneframe` command.

Table 4.39 oneframe Command-Line Debugger Command — Examples

Command	Description
<code>oneframe on</code>	Tells the debugger to only query and show one frame in stack crawls.
<code>oneframe off</code>	Turns off one-frame mode.
<code>oneframe</code>	Reveals if one-frame mode is on or off.

pwd

Lists current working directory.

Syntax

```
pwd
```

quitIDE

Quits the IDE.

Debugger Shell

Debugger Shell Commands

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

Hexadecimal

d

Decimal

u

Unsigned decimal

b

Binary

f

Fractional

h

Hexadecimal

Examples

[Table 4.40](#) lists and defines examples of the `radix` command.

Table 4.40 radix Command-Line Debugger Command — Examples

Command	Description
<code>radix</code>	Lists the current setting.

Table 4.40 radix Command-Line Debugger Command — Examples (*continued*)

Command	Description
radix d	Change the setting to decimal.
radix x	Change the setting to hexadecimal.

redirect

Redirect I/O streams of the current target process.

Syntax

```
redirect <stream> <destination>
```

Options

[Table 4.41](#) lists and defines options of the `red` command.

Table 4.41 red Command-Line Debugger Command — Options

Command	Description
<stream>	stdout stderr both
<destination>	stop server <port> socket [<host>] <port>
<port>	TCP/IP port number of destination socket.
<host>	IP4 address or IP6 address or hostname of target system. Assumed to be “localhost” if omitted.
stop	Ends redirection for the specified stream(s).
server	Attempts to establish a server socket on the specified port. Client sockets may connect to this server socket and read the redirected data. Data written by the target while no client is connected is discarded.
socket	Attempts to connect to the specified server socket. All target output data is written to this connection.

Debugger Shell

Debugger Shell Commands

Examples

[Table 4.42](#) lists and defines examples of the `red` command.

Table 4.42 red Command-Line Debugger Command — Examples

Command	Description
<code>redirect stdout server 27018</code>	Redirects output of stdout for the current process to a server socket on local port 27018.
<code>redirect stderr socket logmachine.com 22018</code>	Attempts to connect to the server socket at port 22018 on host "logmachine.com" and redirects output of stderr for the current process to that connection.
<code>redirect both stop</code>	Ends redirection (if any) currently in place for both stdout and stderr for the current process.

refresh

Discard all cached target data and refresh views.

Syntax

```
refresh [all | -p <pid> <pid> ...]
```

Options

[Table 4.43](#) lists and defines options of the `refresh` command.

Table 4.43 refresh Command-Line Debugger Command — Options

Command	Description
<code>[none]</code>	No option will refresh current process only.
<code>all</code>	Refresh all currently debugged processes.
<code>-p <+pid></code>	Specify list of process ID for the processes to be refreshed.

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

[Table 4.44](#) lists and defines options of the reg command.

Table 4.44 reg Command-Line Debugger Command — Options

Command	Description
[none]	No option is equivalent to <code>reg -reset</code> .
<reg-spec> [{r nr} :] <reg> {r nr}	If multiple registers are specified, then the prefix <code>r :</code> causes a recursive, depth-first traversal of the register hierarchy. The prefix <code>nr :</code> prevents recursion. If unspecified, recursion is the default. Note that different levels of the register hierarchy are represented in the manner of a path with forward-slashes <code>/</code> used to delimit the register groups. A name that contains a slash itself can be represented with an escape backward-slash <code>\</code> followed by the forward-slash. Further note that a backslash in a doubly-quoted Tcl string is itself an escape character -- in this case two backslashes are required. Alternatively, you may use curly braces <code>{</code> and <code>}</code> to denote your string in which case just one backslash is necessary.
<reg>	A register name or a register group name.
..<reg>	The end point for a range of registers to access.
<n>	Number of registers.
all	Specifies all registers.
-d	Print detailed data book information.
-nr	Print only register group, i.e. no registers.
-nv	Print only register groups and register names, i.e. no values.
-np	Do not print anything to the display, only return the data.
reset	Display the register group hierarchy.
-s -ns	Specifies whether each value is to be swapped. For memory, specifies whether each cell is to be swapped. With a setting of <code>-ns</code> , target memory is written in order from lowest to highest byte address. Otherwise, each cell is endian swapped. If unspecified, the setting "config MemSwap" is used.
%<conv>	Specifies the type of the data. Possible values for <code><conv></code> are given below. The default conversion is set by the <code>radix</code> command for memory and registers and by the <code>config var</code> command for variables.
%x	Hexadecimal.

Table 4.44 reg Command-Line Debugger Command — Options

Command	Description
%d	Signed decimal.
%u	Unsigned decimal.
%f	Floating point.
%[E<n>]F	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.
%s	ASCII.

Examples

[Table 4.45](#) lists and defines examples of the `reg` command.

Table 4.45 reg Command-Line Debugger Command — Examples

Command	Description
<code>reg -regset</code>	List all the available register sets on the target chip.
<code>reg R1</code>	Display the value of register R1.
<code>reg "General Purpose Registers/R1"</code>	Display value of register R1 in the General Purpose Register group.
<code>reg R1 -d</code>	Display detailed "data book" contents of R1, including bitfields and definitions.
<code>reg "nr:General Purpose Registers/R1" 25</code>	Beginning with register R1, display the next 25 registers. Register groups are not recursively searched.
<code>reg R1 =123</code>	Change register R1 to 0x123.
<code>reg R1..R5 =5432</code>	Change registers R1 through R5 to 0x5432.

Debugger Shell

Debugger Shell Commands

Table 4.45 reg Command-Line Debugger Command — Examples (*continued*)

Command	Description
<code>reg "General Purpose Registers/R1" =100</code>	Change register R1 in the General Purpose Register group to 0x100.
<code>reg export filename</code>	Export all registers from the target to the specified file.
<code>reg export R1 filename</code>	Export the value of register R1 to the specified file.
<code>reg export "General Purpose Registers/R1" 25 filename</code>	Beginning with register R1, export the next 25 registers to the specified file.
<code>reg import filename</code>	Import registers from the specified file.

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

[Table 4.46](#) lists and defines examples of the `reset` command.

Table 4.46 reset Command-Line Debugger Command — Examples

Command	Description
reset	Issue a soft reset if supported, otherwise a hard reset.
reset s	Issue a soft reset.
reset hard	Issue a hard reset.
reset run	Issue a soft reset if supported, otherwise a hard reset. The target is allowed to run after the reset.

restart

Restarts the current debug session.

Syntax

```
restart
```

Examples

```
restart
```

This command will restart the current debug session.

restore

Write file contents to memory.

Syntax

```
restore -h <filename> [[<ms>:]<addr>|+<offset>]
[8bit|16bit|32bit|64bit]
```

```
restore -b <filename> [<ms>:]<addr> [8bit|16bit|32bit|64bit]
```

Options

[Table 4.47](#) lists and defines options of the `restore` command.

Debugger Shell

Debugger Shell Commands

Table 4.47 restore Command-Line Debugger Command — Options

Command	Description
-h -b	Specifies the input file format as hex or binary.
[<ms>:]<addr>	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 <code>config MemIdentifier</code> and <code>mem -ms</code> for more details.
<offset>	Loads the contents of the hex file at an offset of the original location.
8bit 16bit 32bit 64bit	Controls the memory access size.

Examples

[Table 4.48](#) lists and defines examples of the `restore` command.

Table 4.48 restore Command-Line Debugger Command — Examples

Command	Description
<code>restore -h dat.txt</code>	Loads the contents of the hex file <code>dat.txt</code> into memory.
<code>restore -b dat.bin 0x20</code>	Loads the contents of binary file <code>dat.bin</code> into memory beginning at <code>0x20</code> .
<code>restore -h dat.bin +0x20</code>	Load the contents of the binary file <code>dat.lod</code> into memory with an offset of <code>0x20</code> relative to the address saved in <code>dat.bin</code> .

run

Launch a process

Syntax

```
run [[-index] <index> | [-name] <debug-config-name>]
```

Examples

[Table 4.49](#) lists and defines examples of the `run` command.

Table 4.49 run Command-Line Debugger Command—Examples

Command	Description
run	Start a process using the default launch configuration, which is the last run configuration if one exists and index 0 otherwise.
run -index 3	Start a process using the launch configuration at index 3. Type 'launch' for the current set of launch configurations.
run -name 3	Start a process using the launch configuration named '3'. Type 'launch' for the current set of launch configurations.
run 3	Start a process using the launch configuration named '3'. If '3' does not exist then configuration with index 3 will be started. Type 'launch' for the current set of launch configurations.
run {My Launch Config}	Start debugging using the launch configuration named 'My Launch Config'. Type 'launch' for the current set of launch configurations.

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

Debugger Shell

Debugger Shell Commands

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

[Table 4.50](#) lists and defines examples of the `save` command.

Table 4.50 save Command-Line Debugger Command — Examples

Command	Description
<pre>set addressBlock1 "p:10..`31" set addressBlock2 "p:10000#20" save -h \$addressBlock1 \$addressBlock2 hexfile.txt -a</pre>	<p>Dumps contents of two memory blocks to the text file hexfile.txt (in append mode).</p>
<pre>set addressBlock1 "p:10..`31" set addressBlock2 "p:10000#20" save -b \$addressBlock1 \$addressBlock2 binfile.bin -o</pre>	<p>Dumps contents of two memory blocks to the binary file binfile.bin (in overwrite mode).</p>

setpc

Set the value of the program counter register.

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 [+|-]n <symbol>
<source_file> {target}
setpc {[-va|-ve|-vn]} -line [+|-] <symbol> <source_file>
{target}

```

Examples

[Table 4.51](#) lists and defines examples of the `setpc` command.

Table 4.51 setpc Command-Line Debugger Command — Examples

Command	Description
<code>setpc -address 0x1000</code>	Set the PC to address 0x1000.
<code>setpc -line 10</code>	Set the PC to source line 10 in the current source file.
<code>setpc -line 10 myfile.c</code>	Set the PC to source line 10 in source file 'myfile.c'.
<code>setpc -line +10</code>	Set the PC to an offset of 10 lines from the current source line.
<code>setpc -gsymbol my_extern_function</code>	Set the PC to the address of the 'my_extern_function' global symbol.

Debugger Shell

Debugger Shell Commands

Table 4.51 setpc Command-Line Debugger Command — Examples (*continued*)

Command	Description
<code>setpc -lsymbol my_static_func tion myfile.c</code>	Set the PC to the address of the 'my_static_function' local symbol defined in source file 'myfile.c'.
<code>setpc -line +10 my_extern_func tion</code>	Set the PC to the address corresponding to an offset of 10 source lines from the location where 'my_extern_function' global symbol was defined.
<code>setpc -line +10 my_static_func tion myfile.c</code>	Set the PC to the address corresponding to an offset of 10 source lines from the location where 'my_static_function' local symbol was defined, in source file 'myfile.c'.
<code>setpc -line 10 myfile.c mymodule.ko</code>	Set the PC to source line 10 in source file 'myfile.c'. The file 'myfile.c' is used by the target 'mymodule.ko'.

setpicloadaddr

Indicate where a PIC executable is loaded.

Syntax

```
setpicloadaddr [symfile] <PIC load addr (hex) | reset>]
```

Examples

[Table 4.52](#) lists and defines examples of the `setpicloadaddr` command.

Table 4.52 setpicloadaddr Command-Line Debugger Command — Examples

Command	Description
<code>setpicloadaddr 0x40000</code>	Tells the debugger the main executable is loaded at 0x40000.

Table 4.52 setpicloadaddr Command-Line Debugger Command — Examples (continued)

Command	Description
setpicloadaddr myapp.elf 0x40000	Tells the debugger myapp.elf is loaded at 0x40000.
setpicloadaddr myapp.elf reset	Tells the debugger myapp.elf is loaded at the address set in the ELF.

stack

Print the call stack.

Syntax

```
stack [num_frames] [-default]
```

Examples

[Table 4.53](#) lists and defines examples of the `stack` command.

Table 4.53 stack Command-Line Debugger Command — Examples

Command	Description
stack	Prints the entire call stack unless limited with <code>stack -default</code> .
stack 6	Prints the 6 innermost call stack levels.
stack -6	Prints the 6 outermost call stack levels.
stack 6 - default	Limits the number of stack frames shown to the 6 innermost levels.
stack -default	Removes the stack frame limit.

status

Lists the debug status of all existing active targets.

Debugger Shell

Debugger Shell Commands

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.

into|over|out

Controls the type of step operation. If unspecified, **into** is used.

nve

Step non optimized action.

nxt

Step next action.

fwd

Step forward action.

end

Step end of statement action.

aft

Step end all previous action.

Examples

[Table 4.54](#) lists and defines examples of the `step` command.

Table 4.54 step Command-Line Debugger Command—Examples

Command	Description
<code>step</code>	Step into the current source or assembly line.
<code>step over</code>	Step over the current source or assembly line.
<code>step out</code>	Step out of a function.
<code>step asm</code>	Step over a single assembly instruction.

stepi

Execute to the next assembly instruction.

Syntax

`stepi`

stop

Stops a running program (started by a `go`, `step`, or `next` command).

Syntax

`stop`

Examples

[Table 4.55](#) lists and defines examples of the `stop` command.

Table 4.55 stop Command-Line Debugger Command — Examples

Command	Description
<code>stop</code>	Using it after command <code>go/step out/next</code> , this will stop the target program.

Debugger Shell

Debugger Shell Commands

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.

Examples

[Table 4.56](#) lists and defines examples of the switchtarget command.

Table 4.56 switchtarget Command-Line Debugger Command — Examples

Command	Description
switchtarget	list currently available debug sessions.
switchtarget 0	select the thread with index 0
switchtarget -cur	list the index of the current thread.
switchtarget -ResetIndex	reset the index counter to 0, not valid while debugging.
switchtarget -tid	list the thread IDs of the current process of the current connection.

Table 4.56 switchtarget Command-Line Debugger Command — Examples (continued)

Command	Description
<code>switchtarget - pid</code>	list the process IDs of the debugged processes of the current connection.
<code>switchtarget - pid -arch=EPPC</code>	list the process IDs of the debugged processes of EPPC architecture on the current debug system.
<code>switchtarget - pid - conn=Launch-1</code>	list the process IDs of the debugged processes of the Launch-1 connection.
<code>switchtarget - arch - conn=Launch-1</code>	list the architectures debugged on Launch-1 connection.
<code>switchtarget - conn - arch=EPPC</code>	list the name of the connection of EPPC architecture on the current debug system.
<code>switchtarget - pid=0 -tid=0 - arch=EPPC</code>	switch current context to thread 0 of process 0 of EPPC architecture on the current debug system.
<code>switchtarget - pid=0 -tid=0 - conn=Launch-1</code>	switch current context to thread 0 of process 0 on Launch-1 connection.

system

Execute system command.

Syntax

`system [command]`

Parameter

`command`

Any system command that does not use a full screen display.

Examples

[Table 4.57](#) lists and defines examples of the `system` command.

Debugger Shell

Debugger Shell Commands

Table 4.57 system Command-Line Debugger Command — Examples

Command	Description
system del *.tmp	Delete from the current directory all files that have the .tmp filename extension.

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

[Table 4.58](#) lists and defines options of the var command.

Table 4.58 var Command-Line Debugger Command — Options

Command	Description
[none]	No option is equivalent to “var v:”.
v:	If this option appears with no <var> following it, then all variables pertinent to the current scope are printed.
<var>	Symbolic name of the variable to print. Can be a C expression as well.
-np	Don't print anything to the display, only return the data.
-s -ns	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 "config MemSwap" is used.

Table 4.58 var Command-Line Debugger Command — Options

Command	Description
%<conv>	Specifies the type of the data. Possible values for <conv> are given below. The default conversion is set by the radix command for memory and registers and by the config var command for variables.
%x	Hexadecimal.
%d	Signed decimal.
%u	Unsigned decimal.
%f	Floating point.
%[E<n>]F	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".
%s	ASCII.

Examples

[Table 4.59](#) lists and defines examples of the var command.

Table 4.59 var Command-Line Debugger Command — Examples

Command	Description
var myVar -s %d	Display the endian-swapped contents of variable myVar in decimal.
var myVar =10	Change the value of variable myVar to 16 (0x10).

wait

Tells the debugger to wait for a specified amount of time, or until you press the space bar.

Syntax

```
wait <time-ms>
```

Debugger Shell

Debugger Shell Commands

Parameter

`time-ms`

Number of milliseconds to wait.

Examples

[Table 4.60](#) lists and defines examples of the `wait` command.

Table 4.60 wait Command-Line Debugger Command — Examples

Command	Description
<code>wait</code>	Debugger waits until you press the space bar.
<code>wait 2000</code>	Wait for 2 seconds.

watchpoint

Sets, removes, disables, enables or list watchpoints. You can also set condition on watchpoint.

Syntax

```

watchpoint
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

[Table 4.61](#) lists and defines examples of the watchpoint command.

Table 4.61 watchpoint Command-Line Debugger Command — Examples

Command	Description
watchpoint	Display all watchpoints.
watchpoint gData	Set read-write (the default) watchpoint on variable gData .
watchpoint -r gData	Set read-only watchpoint on variable gData .
watchpoint all off	Remove all watchpoints.
watchpoint #4 disable	Disable watchpoint number 4.
watchpoint 10343 4	Set a watchpoint at memory address 10343 of length 4.
watchpoint #4 ignore 3	Set ignore count to 3 for watchpoint number 4.
watchpoint #4 cond x == 3	Set the condition for watchpoint number 4 to fire only if <code>x == 3</code> .
watchpoint #4 type -rw	Set the access type read/write for watchpoint number 4.
watchpoint #4 size 8	Set the size to 8 units for watchpoint number 4.



Debugger Shell

Debugger Shell Commands

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 topics of this chapter are:

- [Command-Line Syntax](#)
- [Launching a Debug Session](#)
- [Stepping](#)
- [Config Settings](#)

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
```

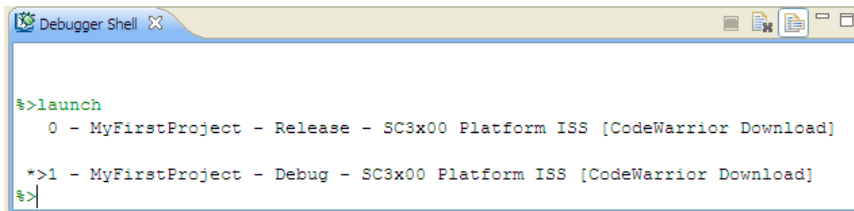
NOTE Users familiar with the `-vmargs` option in the CodeWarrior Eclipse IDE should note that CodeWarrior will not work properly if `-vmargs` is used. Please use the custom `-vmargsplus` option in place of the `-vmargs` option.

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 (See [Figure 5.1](#))
- `debug`: to start a debug session (See [Figure 5.2](#))
- `run`: to start a process (See [Figure 5.2](#))

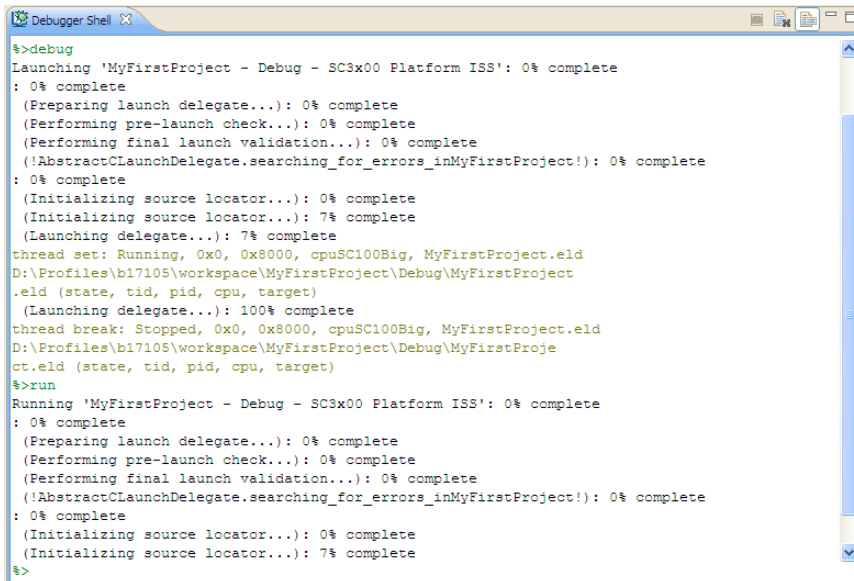
Figure 5.1 The launch Command



```

Debugger Shell X
%>launch
0 - MyFirstProject - Release - SC3x00 Platform ISS [CodeWarrior Download]
*>1 - MyFirstProject - Debug - SC3x00 Platform ISS [CodeWarrior Download]
%>
  
```

Figure 5.2 The debug and run Commands



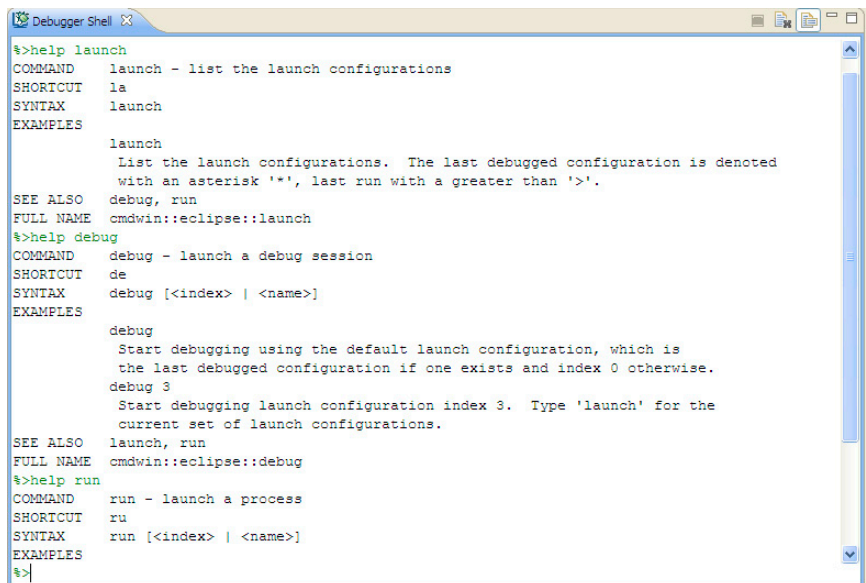
```

Debugger Shell X
%>debug
Launching 'MyFirstProject - Debug - SC3x00 Platform ISS': 0% complete
: 0% complete
(Preparing launch delegate...): 0% complete
(Performing pre-launch check...): 0% complete
(Performing final launch validation...): 0% complete
(!AbstractCLaunchDelegate.searching_for_errors_inMyFirstProject!): 0% complete
: 0% complete
(Initializing source locator...): 0% complete
(Initializing source locator...): 7% complete
(Launching delegate...): 7% complete
thread set: Running, 0x0, 0x8000, cpuSC100Big, MyFirstProject.eld
D:\Profiles\b17105\workspace\MyFirstProject\Debug\MyFirstProject
.eld (state, tid, pid, cpu, target)
(Launching delegate...): 100% complete
thread break: Stopped, 0x0, 0x8000, cpuSC100Big, MyFirstProject.eld
D:\Profiles\b17105\workspace\MyFirstProject\Debug\MyFirstProje
ct.eld (state, tid, pid, cpu, target)
%>run
Running 'MyFirstProject - Debug - SC3x00 Platform ISS': 0% complete
: 0% complete
(Preparing launch delegate...): 0% complete
(Performing pre-launch check...): 0% complete
(Performing final launch validation...): 0% complete
(!AbstractCLaunchDelegate.searching_for_errors_inMyFirstProject!): 0% complete
: 0% complete
(Initializing source locator...): 0% complete
(Initializing source locator...): 7% complete
%>
  
```

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. These commands now function through the `debug` command used with a CodeWarrior Attach launch configuration or a CodeWarrior Connect launch configuration.

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



```
Debugger Shell X
%>help launch
COMMAND    launch - list the launch configurations
SHORTCUT   la
SYNTAX     launch
EXAMPLES

    launch
        List the launch configurations. The last debugged configuration is denoted
        with an asterisk '*', last run with a greater than '>'.

SEE ALSO   debug, run
FULL NAME  cmdwin::eclipse::launch
%>help debug
COMMAND    debug - launch a debug session
SHORTCUT   de
SYNTAX     debug [<index> | <name>]
EXAMPLES

    debug
        Start debugging using the default launch configuration, which is
        the last debugged configuration if one exists and index 0 otherwise.
    debug 3
        Start debugging launch configuration index 3. Type 'launch' for the
        current set of launch configurations.

SEE ALSO   launch, run
FULL NAME  cmdwin::eclipse::debug
%>help run
COMMAND    run - launch a process
SHORTCUT   ru
SYNTAX     run [<index> | <name>]
EXAMPLES

%>
```

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

Debugger Script Migration

Config Settings

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`.

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

Command	Description
<code>config c</code>	Sets the syntax coloring
<code>config o</code>	Aborts a script
<code>config page</code>	Controls the paging behavior
<code>config s</code>	Sets the page size
<code>config project</code>	Accesses the build projects
<code>config target</code>	Accesses the build targets

The remaining `config` command settings work the same as in the CodeWarrior Classic IDE.

Index

A

- access breakpoint - refer to watchpoints 246
- actions for debugging 173
- Active Configuration 39
- active configuration 39
- Active debug context 43
- Address checkbox 71
- Address Line fault 71
- attach command 329

B

- Breakpoint Actions 158
 - add a action 160
 - attach breakpoint actions 161
 - preference page 159
- Breakpoints 148
 - actions 158
 - disable 157
 - enable 157
 - group breakpoints 156
 - limit breakpoints to active debug context 155
 - line breakpoint 150
 - method breakpoint 151
 - modify properties 153
 - regular breakpoints 150
 - remove 157
 - remove all 158
 - restrict to selected targets and threads 154
 - setting special breakpoints 152
 - skip all 158
 - special breakpoints 151
 - hardware 151
 - software 151
 - states 148
 - view 148
- breakpoints
 - conditional 148
 - disabled 148, 152
 - enabled 148, 152
 - regular 148

- temporary 148
 - working with 152
- breakpoints, disabling 157
- breakpoints, modifying properties for 153
- Bus Noise checkbox 71
- Bus Noise test
 - subtests
 - Full Range Converging 73
 - Maximum Invert Convergence 73
 - Sequential 73
- bus noise, defined 73
- Byte option button 68

C

- Cache View 20
 - opening 20
 - preserve sorting 21
 - toolbar menu 22
- Change
 - debugger settings 165
- Change Program Counter Value 176
- changing
 - register data views 225
 - register values 222
- checkboxes
 - Address 71
 - Bus Noise 71
 - Use Target CPU 71
 - Walking 1's 71
- code
 - disabling breakpoints 157
 - disabling watchpoints 251
 - enabling watchpoints 251
 - modifying breakpoint properties 153
 - setting watchpoints in 247
 - viewing watchpoint properties 249
- CodeWarrior
 - overview 15
- Codewarrior Drag & Drop support 24
- CodeWarrior Project Importer 24
 - access paths 31, 346
 - global settings 30, 345

- CodeWarrior Projects view 38
- Column Headers 39
- Command-Line syntax to launch CW Eclipse IDE 327
- commands
 - Debug 174
 - Restart 176
 - Resume 175
 - Run 175
 - Step Into 174
 - Step Out 174
 - Step Over 175
- common debugging actions 173
- conditional breakpoints 148
- Config settings in CW Eclipse IDE 330
- connect command 329
- contents
 - of register 225
- context label 43
- context menus
 - using 170
- controlling program execution 143
- conventions
 - figures 14
 - for manual 14
 - keyboard shortcuts 14
- Copy Cache 23
- core 43
- core index 43
- cores, debugging multiple 197
- Creating
 - MMU configuration 87
- Customize Column Headers 41

D

- Data Line fault 71
- Debug command 174
- debug command 328
- Debug perspective 171
- Debug view 43
- debug view 172
- debugger
 - restarting 176
 - starting 174

- debugger data
 - symbolics 233
- Debugger Settings 164
 - changing 165
 - revert 167
- Debugger settings
 - change 165
- debugger shell 169
 - code hints 255
- debugger, defined 144
- debugging
 - common actions 173
 - Ethernet TAP remote connection 240
 - multiple cores 197
 - program execution 143
 - restarting a session 176
 - starting a session 174
- definition
 - of bus noise 73
 - of debugger 144
 - of memory aliasing 72
 - of watchpoints 246
- Deselect AI 41
- details
 - viewing for registers 225
- development-process cycle for software 15
- Disable LRU 23
- disabled breakpoint 148, 152
- Disassembly View 177
- display register view 221
- documentation
 - formats 14
 - structure 13
 - types 14

E

- e500 core, 43
- ecd.exe 144
 - build 145
 - generateMakefiles 146
 - getOptions 145
 - setOptions 147
 - updateWorkspace 147
- Enable 23

-
- enabled breakpoint 148, 152
 - Erase Text 43
 - Ethernet TAP
 - remote connection 240
 - execution
 - of program, controlling 143
 - execution, resuming 175
 - execution, stopping 175
 - Export Cache 23
- F**
- figure conventions 14
 - filter 42
 - Find and Open File 46
 - finish command 330
 - Flash Programmer
 - run target task 64
 - Flash programmer 47
 - add flash device 51
 - configure target task 50
 - create a target task 48
 - flash programmer actions 53
 - target ram settings 52
 - Flash Programmer Actions 53
 - add checksum actions 57
 - add erase/blank check actions 54
 - add program/verify actions 56
 - remove an action 64
 - flat list viewing 40
 - Flush 23
 - Flush Line 23
 - formats
 - for documentation 14
 - Full Range Converging substest 73
- G**
- Global Variables window
 - 244
- H**
- hardware diagnostic panels
 - Memory Read / Write 68
 - Memory Tests 70
 - Address 72
 - Bus Noise 73
 - Bus Noise in address lines 73
 - Bus Noise in data lines 73
 - Walking Ones 71
 - Scope Loop 69
 - Hardware Diagnostics 65
 - action editor 67
 - action type 68
 - address lines 73
 - address test 72
 - data lines 73
 - loop speed 69
 - memory access 68
 - memory test use cases 74
 - memory tests 70
 - walking one tests 71
 - Hardware Diagnostics window
 - Memory Read / Write panel
 - Byte option button 68
 - Long Word option button 68
 - Read option button 68
 - Target Address text box 69
 - Word option button 68
 - Write option button 68
 - Memory Tests panel
 - Address checkbox 71
 - Bus Noise checkbox 71
 - Passes text box 71
 - Use Target CPU checkbox 71
 - Walking 1's checkbox 71
 - Scope Loop panel
 - Speed slider 70
 - help command 329
 - hierarchal tree 40
 - how to
 - add global variables for different processes 244
 - change register data views 225
 - change register values 222
 - disable a breakpoint 157
 - disable a watchpoint 251
 - enable a watchpoint 251
 - manipulate variable formats 243
-

- modify breakpoint properties 153
- open the Registers window 221, 229
- restart the debugger 176
- resume program execution 175
- run a program 175
- set a watchpoint 247
- start the debugger 174
- step into a routine 174
- step out of a routine 174
- step over a routine 175
- stop program execution 175
- use context menus 170
- view registers 222
- view watchpoint properties 249

I

- Import/Export/Fill Memory 75
 - create target task 75
 - exporting memory to a file 79
 - fill memory with data pattern 81
- Invalidate 23
- Invalidate Line 23
- Inverse LRU 23

K

- kernal awareness 43
- Kernel Awareness 43
- kernel threads 43
- Key Mappings 83
- keyboard conventions 14

L

- launch command 328
- launch configuration 164
- Launch Group 179
 - create 179
 - debug view 184
 - launching 184
- Launching a debug session in CW Eclipse IDE 328
- least significant bit 72
- Linux 43
- Load Multiple Binaries 185

- Lock 23
- Lock Line 23
- Lock Way 23
- Lock Ways 23
- Long Word option button 68
- LSB 72

M

- manual conventions 14
- Manual Path Mapping
 - Adding Path Mapping to Workspace 216
- Maximum Invert Convergence subtest 73
- memory
 - clearing watchpoint 196
 - setting watchpoint 195
- memory aliasing, defined 72
- Memory Read / Write panel 68
- memory tests
 - Address 72
 - Bus Noise 73
 - address lines 73
 - data lines 73
 - Bus Noise test
 - Full Range Converging subtest 73
 - Maximum Invert Convergence subtest 73
 - Sequential subtest 73
 - Walking Ones 71
 - Walking Ones test
 - Address Line fault 71
 - Data Line fault 71
 - Ones Retention subtest 72
 - Retention fault 72
 - Walking Ones subtest 72
 - Walking Zeros subtest 72
 - Zeros Retention subtest 72
- Memory Tests panel 70
 - Address test 72
 - Bus Noise test 73
 - address lines 73
 - data lines 73
 - Walking Ones test 71
- Memory View 188
 - adding memory monitor 189

-
- clearing watchpoint 196
 - setting watchpoint 195
 - MMU configuration
 - creating 87
 - MMU Configurator
 - pages 91
 - MMU configurator
 - create MMU configuration 87
 - open view 102
 - pages 91
 - saving generated code 101
 - saving MMU configurator settings 90
 - toolbar 90
 - MMU Configurator Generated Code
 - save 101
 - MMU configurator pages 91
 - data MATT page 97
 - general page 91
 - program MATT page 93
 - MMU Configurator Settings
 - save 90
 - MMU Configurator View
 - open 102
 - MMU configurator view 102
 - most significant bit 72
 - MSB 72
 - multi-core debugging 197
 - Multicore Group
 - creating 200
 - Multicore Groups 199
 - Multicore OS 43
 - Multiple homogeneous cores 43
- N**
- New External File 106
 - next command 330
 - nexti command 329
- O**
- Ones Retention subtest 72
 - Opening the Cache Viewer 20
 - option buttons
 - Byte 68
 - Long Word 68
 - Read 68
 - Word 68
 - Write 68
 - options
 - Memory Read / Write panel 68
 - Memory Tests panel 70
 - Scope Loop panel 69
 - overview
 - of CodeWarrior 15
- P**
- P4080 43
 - Pages
 - MMU configurator 91
 - Passes text box 71
 - Path Mapping 210
 - Automatic Path Mapping 211
 - Manual Path Mapping 213
 - perspective
 - debug 171
 - Popup 42
 - preserve sorting 21
 - process cycle
 - of software development 15
 - program
 - resuming execution 175
 - running 175
 - stopping execution 175
 - Project Importer 24
 - Project View 38
 - active configuration 39
 - column headers 40
 - list view 40
 - quick search 42
 - tree view 40
- Q**
- Quick Search 39, 42
- R**
- Read option button 68
 - Refresh 22
 - Register Details view 225
-

- register view
 - change register data format 225
- registers
 - changing data views of 225
 - changing values of 222
 - Register Details view 225
 - viewing 222
 - viewing details of 225
- registers view
 - change register values 222
 - displaying 221
 - viewing register 222
- Registers window
 - opening 221, 229
- regular breakpoints 148
- remote connection
 - for Ethernet TAP 240
- Restart command 176
- restarting
 - debugger 176
- Resume command 175
- resuming program execution 175
- Retention fault 72
- Reverting Debugger Settings 167
- routine
 - stepping into 174
 - stepping out of 174
 - stepping over 175
- Run command 175
- run command 328
- running
 - a program 175

S

- Scope Loop panel 69
- Search 23
- Search Again 23
- Select All 41
- Sequential substest 73
- setting access breakpoint 247
- shortcut conventions 14
- Show files in a flat view 40
- Show files in a hierarchal view 40
- software

- development process cycle 15
- source code
 - disabling breakpoints 157
 - disabling watchpoints 251
 - enabling watchpoints 251
 - modifying breakpoint properties 153
 - setting watchpoints in 247
 - viewing watchpoint properties 249
- Speed slider 70
- starting
 - debugger 174
- state
 - disabled, for breakpoints 148, 152
 - enabled, for breakpoints 148, 152
- Step commands in CW Eclipse IDE 329
- step in command 330
- Step Into command 174
- step li command 330
- Step Out command 174
- Step Over command 175
- stepli command 329
- stepping into a routine 174
- stepping out of a routine 174
- stepping over a routine 175
- stopping program execution 175
- structure
 - of documentation 13
- Symbolics 233
- System Browser 43
 - console view 45
 - kernal awareness 43
 - OS application 44
- system browser view 234

T

- TAP Remote Connections 240
- Target Address text box 69
- target process selection 135
- Target Task View 136
- tasks
 - adding global variables for different processes 244
 - changing register data views 225
 - changing register values 222

- disabling a breakpoint 157
- disabling a watchpoint 251
- enabling a watchpoint 251
- manipulating variable formats 243
- modifying breakpoint properties 153
- opening the Registers window 221, 229
- restarting the debugger 176
- resuming program execution 175
- running a program 175
- setting a watchpoint 247
- starting the debugger 174
- stepping into a routine 174
- stepping out of a routine 174
- stepping over a routine 175
- stopping program execution 175
- using context menus 170
- viewing registers 222
- viewing watchpoint properties 249
- temporary breakpoints 148
- text boxes
 - Passes 71
 - Target Address 69
- Tree and List View 39
- Type Ahead 42
- types
 - of documentation 14

U

- Unlock Way 23
- Use Target CPU checkbox 71

V

- variable formatting 241, 243
- variables
 - manipulating formats 243
- Variables View 241
- view
 - breakpoints 148
 - cache 20
 - debug 172
 - disassembly 177
 - memory 188
 - MMU configurator 102
 - project 38

- System Browser 43
 - system browser 234
 - target task 136
- View Memory 23
- viewing
 - register details 225
 - registers 222
- viewing access breakpoint 249
- Viewing Binaries 187
- viewing registers 222
- Views
 - Breakpoints 148
 - Cache 20
 - Disassembly 177
 - Memory 188
 - MMU configurator 102
 - project 38
 - system browser 43
 - Target Task 136
 - Variables 241

W

- Walking 1's checkbox 71
- Walking Ones subtest 72
- Walking Ones test
 - Address Line fault 71
 - Data Line fault 71
 - Retention fault 72
 - subtests
 - Ones Retention 72
 - Walking Ones 72
 - Walking Zeros 72
 - Zeros Retention 72
- Walking Zeros subtest 72
- watchpoint
 - memory 195, 196
- watchpoints
 - access breakpoint 246
 - defined 246
- watchpoints, disabling 251
- watchpoints, enabling 251
- watchpoints, setting 247
- watchpoints, viewing properties for 249
- what is

a debugger 144
Wildcard character support 42
Word option button 68
working with breakpoints 152
Write 22
Write option button 68

Z

Zeros Retention subtest 72