

CodeWarrior Development Studio for Advanced Packet Processing Targeting Manual

Document Number: CWAPPTM
Rev. 10.2, 01/2016



Contents

Section number	Title	Page
----------------	-------	------

Chapter 1 Introduction

1.1	Release notes.....	11
1.2	Accompanying documentation.....	11
1.3	CodeWarrior development tools.....	12
1.3.1	Eclipse IDE.....	12
1.3.2	C/C++ compiler.....	13
1.3.3	Standalone assembler.....	13
1.3.4	Linker.....	14
1.3.5	Debugger.....	14
1.3.6	CodeWarrior profiling and analysis tools.....	15
1.4	CodeWarrior development process.....	15
1.4.1	Project files.....	15
1.4.2	Compiling.....	16
1.4.3	Linking.....	16
1.4.4	Editing code.....	17
1.4.5	Debugging.....	17

Chapter 2 Working with Projects

2.1	CodeWarrior bareboard project wizard.....	19
2.1.1	Create a CodeWarrior Bareboard Project page.....	20
2.1.2	Processor page.....	21
2.1.3	Debug Target Settings page.....	22
2.1.4	Build Settings page.....	23
2.1.5	Configurations page.....	24
2.1.6	Software Analysis page.....	25
2.2	Creating projects.....	26
2.2.1	Creating CodeWarrior bareboard application project.....	27

Section number	Title	Page
2.2.2	Creating CodeWarrior bareboard library project.....	30
2.3	Building projects.....	31
2.3.1	Manual-build mode.....	32
2.3.2	Auto-build mode.....	32
2.4	Debugging projects.....	32
2.5	Deleting project.....	33

Chapter 3 Debug Configuration

3.1	Using CodeWarrior debug configuration tabs.....	35
3.1.1	Main.....	36
3.1.2	Arguments.....	39
3.1.3	Debugger.....	40
3.1.3.1	Debug.....	41
3.1.3.2	Download.....	43
3.1.3.3	PIC.....	45
3.1.3.4	System Call Services.....	46
3.1.3.5	Other Executables.....	47
3.1.3.6	Symbolics.....	49
3.1.3.7	OS Awareness.....	50
3.1.4	Trace and Profile.....	52
3.1.5	Source.....	53
3.1.6	Environment.....	54
3.1.7	Common.....	55
3.2	Customizing debug configurations.....	57
3.3	Reverting debug configuration settings.....	58

Chapter 4 Build Properties

4.1	Changing build properties.....	59
4.2	Restoring build properties.....	60

Section number	Title	Page
4.3	Build properties for APP.....	60
4.3.1	CPU.....	61
4.3.2	Debugging.....	62
4.3.3	Messages.....	63
4.3.4	Linker.....	63
4.3.4.1	Input.....	64
4.3.4.2	Link Order.....	65
4.3.4.3	General.....	65
4.3.4.4	Output.....	66
4.3.5	Compiler.....	68
4.3.5.1	Preprocessor.....	69
4.3.5.2	Input.....	69
4.3.5.3	Warnings.....	71
4.3.5.4	Optimization.....	72
4.3.5.5	Processor.....	73
4.3.5.6	C/C++ Language.....	75
4.3.6	Assembler.....	77
4.3.6.1	Input.....	77
4.3.6.2	General.....	78
4.3.7	Disassembler.....	79
4.3.7.1	Disassembler Settings.....	79
4.3.8	Preprocessor.....	80
4.3.8.1	Preprocessor Settings.....	80

Chapter 5

Working with AIOP Debugger

5.1	AIOP debug model.....	83
5.1.1	Overview.....	84
5.1.2	AIOP global halt.....	84
5.1.3	AIOP running.....	84

Section number	Title	Page
5.1.4	AIOP debug perspective.....	85
5.1.4.1	Task centric perspective.....	85
5.1.4.2	Core centric perspective.....	86
5.1.5	Task stepping mode.....	87
5.2	AIOP task aware debugging.....	87
5.2.1	Activating task awareness services.....	88
5.2.2	Viewing AIOP tasks.....	89
5.2.3	Viewing non-idle tasks only.....	90
5.2.4	Adding task memory location columns in System Browser.....	91
5.2.5	Viewing task entry point and OSM data in System Browser.....	93
5.2.6	Targeting AIOP tasks.....	94
5.2.7	Performing run control operations.....	95
5.3	Standard debugging features.....	97
5.3.1	Connection types.....	97
5.3.1.1	CCSSIM2 ISS.....	98
5.3.1.2	CodeWarrior TAP.....	99
5.3.2	Editing system configuration.....	101
5.3.2.1	Initialization.....	101
5.3.2.2	Memory.....	102
5.3.3	CodeWarrior command-line debugger.....	103
5.3.4	Memory configuration file.....	105
5.3.5	Displaying memory contents.....	106
5.3.6	Displaying register contents.....	108
5.3.6.1	Adding register group.....	108
5.3.6.2	Editing register group.....	110
5.3.6.3	Removing register group.....	110
5.3.6.4	Changing register bit value.....	110
5.3.7	Using register details window.....	111
5.3.7.1	Bit Fields.....	112

Section number	Title	Page
5.3.7.2	Actions.....	113
5.3.7.3	Description.....	113
5.3.7.4	Viewing register details.....	114
5.3.7.5	Changing bit field.....	116
5.3.8	Setting watchpoints.....	117
5.3.8.1	Adding watchpoints.....	118
5.3.8.2	Removing watchpoints.....	120
5.3.9	Setting breakpoints.....	120
5.3.9.1	AIOP task specific breakpoints.....	123
5.3.9.2	Setting hardware breakpoints.....	124
5.3.9.2.1	Setting hardware breakpoint using editor view.....	124
5.3.9.2.2	Setting hardware breakpoint using debugger shell.....	124
5.3.9.3	Removing breakpoints.....	125
5.3.9.3.1	Removing breakpoint using marker bar.....	125
5.3.9.3.2	Removing breakpoint using Breakpoints view.....	125
5.3.9.4	Removing hardware breakpoints.....	126
5.3.9.4.1	Removing hardware breakpoint using editor view.....	126
5.3.9.4.2	Removing hardware breakpoint using Debugger Shell.....	126
5.3.10	Setting stack depth.....	127
5.3.11	Changing program counter value.....	127
5.3.12	Hard resetting.....	127
5.3.13	Loading and saving memory.....	128
5.3.14	Filling memory.....	130
5.3.15	Controlling cross triggering.....	130
5.4	CodeWarrior Executable Importer wizard.....	131
5.4.1	Import a CodeWarrior Executable file page.....	132
5.4.2	Import C/C++/Assembler Executable Files page.....	132
5.4.3	Processor page.....	133
5.4.4	Debug Target Settings page.....	133

Section number	Title	Page
5.4.5	Configurations page.....	134
5.5	Debugging externally built executable file.....	134
5.6	Multi-core operations.....	139
5.6.1	Multi-core operations in IDE.....	139
5.6.2	Multi-core operations in Debugger Shell.....	140

Chapter 6 Working with hardware tools

6.1	Working with hardware diagnostics.....	143
6.1.1	Creating new hardware diagnostics task.....	143
6.1.2	Executing hardware diagnostics task.....	146
6.1.3	Editing hardware diagnostics task.....	146
6.1.4	Hardware Diagnostics Action editor	148
6.2	Manipulating target memory.....	148
6.2.1	Creating target task to manipulate memory.....	149
6.2.2	Editing import/export/fill memory task.....	150
6.2.3	Import/Export/Fill Memory Action editor	151

Chapter 7 JTAG configuration files

7.1	JTAG configuration file syntax.....	153
7.2	Using JTAG configuration file to override RCW.....	154
7.3	Using JTAG configuration file to specify multiple linked devices on a JTAG chain.....	155
7.4	Setting remote system to use JTAG configuration file.....	157

Chapter 8 Target initialization files

8.1	Using target initialization files.....	159
8.2	Target initialization file commands.....	161
8.2.1	Access to named registers from within scripts.....	162
8.2.2	Cfg target initialization commands.....	162
8.2.2.1	alternatePC.....	163
8.2.2.2	ANDmem.l.....	163

Section number	Title	Page
8.2.2.3	ANDmmr.....	164
8.2.2.4	IncorMMR.....	165
8.2.2.5	ORmem.l.....	166
8.2.2.6	reset.....	167
8.2.2.7	run.....	167
8.2.2.8	setCoreID.....	168
8.2.2.9	resetCoreID.....	168
8.2.2.10	sleep.....	169
8.2.2.11	stop.....	169
8.2.2.12	writemem.b.....	169
8.2.2.13	writemem.w.....	170
8.2.2.14	writemem.l.....	171
8.2.2.15	writemmr.....	172
8.2.2.16	writereg.....	172
8.2.2.17	writereg64.....	173
8.2.2.18	writereg128.....	174
8.2.2.19	writereg192.....	175
8.2.2.20	writespr.....	176
8.3	Target initialization using Tcl script.....	177

Chapter 9

Memory configuration files

9.1	Using memory configuration files.....	179
9.2	Memory configuration file commands.....	180
9.2.1	autoEnableTranslations.....	181
9.2.2	range.....	182
9.2.3	reserved.....	183
9.2.4	reservedchar.....	183
9.2.5	translate.....	184

Chapter 10



Section number	Title	Page
	Debugger limitations and workarounds	
10.1	MC/AIOP cores.....	187

Chapter 1

Introduction

This manual explains how to use the CodeWarrior Development Studio tool set to develop software for bareboard applications running on Freescale processors. This chapter presents an overview of this manual and introduces you to the CodeWarrior development tools and development process.

1.1 Release notes

It is recommended that you should read the release notes before using the CodeWarrior IDE. The release notes include information about new features, last-minute changes, bug fixes, incompatible elements, or other sections that may not be included in this manual.

NOTE

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

1.2 Accompanying documentation

The **Documentation** page describes the documentation included in this version of CodeWarrior Development Studio for Advanced Packet Processing. You can access the **Documentation** page by:

- Using a shortcut link on the Desktop created by your CodeWarrior installer.
- Opening the `START_HERE.html` file available in the `<CWInstallDir>\CW_APP\LS\Help` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

1.3 CodeWarrior development tools

The programming for advanced packet processing (APP) processors is similar to the programming for any other CodeWarrior platform target. If you have not used the CodeWarrior tools before, you can start by understanding the Eclipse IDE, that is used to host the tools. More information on the Eclipse IDE is available in the next section.

If you are an experienced CodeWarrior user, note that the CodeWarrior Development Studio for Advanced Packet Processing v10.x environment uses the Eclipse IDE, whose user interface is substantially different from the *classic* CodeWarrior IDE. For more details on these interface differences, see *CodeWarrior Common Features Guide* available in the `<CWInstallDir>\CW_APP\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

If you have not used the CodeWarrior IDE before, then you have to become familiar with following tools:

- [Eclipse IDE](#)
- [C/C++ compiler](#)
- [Standalone assembler](#)
- [Linker](#)
- [Debugger](#)
- [CodeWarrior profiling and analysis tools](#)

1.3.1 Eclipse IDE

The Eclipse integrated development environment (IDE) is an open-source development environment that lets you develop and debug your software. It controls the project manager, the source code editor, the class browser, the compilers and linkers, and the debugger.

If you are more familiar with command-line development tools, you may find the concept of a CodeWarrior project new. The Eclipse Workspace organizes all files related to your project. This allows you to see your project at a glance and eases the organization and navigation between the source code files.

The Eclipse IDE has an extensible architecture that uses the plug-in compilers and linkers to target the various operating systems and microprocessors. The IDE is hosted on the Microsoft Windows, x86 Linux, and other platforms. There are many development tools available for the IDE, including C, C++, and Java compilers for the Desktop and embedded processors.

For more information about the Eclipse IDE, read the Eclipse documentation at:

<http://www.eclipse.org/documentation/>

1.3.2 C/C++ compiler

The CodeWarrior C/C++ compiler is an ANSI-compliant compiler. It compiles the C and C++ statements and assembles the inline assembly language statements. You can generate the applications and libraries by using the CodeWarrior compiler in conjunction with the CodeWarrior linker for APP processors.

The IDE manages the execution of the compiler. It runs the compiler in the following conditions:

- When you change a source file and issue the `make` command.
- When you select a source file in your project and issue the compile, preprocess, or precompile command.

For more information about the CodeWarrior C/C++ compiler and its inline assembler, see *CodeWarrior Development Studio for Advanced Packet Processing Build Tools Reference Manual* available in the `<CWInstallDir>\CW_APP\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

1.3.3 Standalone assembler

The CodeWarrior assembler is a standalone assembler that translates the assembly-language source code to the machine-language object files or executable programs.

For more information about the CodeWarrior assembler, see *CodeWarrior Development Studio for Advanced Packet Processing Build Tools Reference Manual* available in the `<CWInstallDir>\CW_APP\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

1.3.4 Linker

The CodeWarrior linker generates the binaries that conform to the embedded application binary interface (EABI). The linker combines the object modules created by the compiler and assembler with the modules in the static libraries to produce a binary file in the executable and linkable (ELF) format.

The CodeWarrior linker provides the following key features, that enables you to:

- use the absolute addressing
- create the multiple user-defined sections
- generate the S-Record files
- generate the PIC/PID binary files

The IDE runs the linker each time you build your project.

For more information about the CodeWarrior linker, see *CodeWarrior Development Studio for Advanced Packet Processing Build Tools Reference Manual* available in the `<CWInstallDir>\CW_APP\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

1.3.5 Debugger

The CodeWarrior debugger controls the execution of your program and allows you to see what is happening internally as the program runs. You can use the debugger to identify the problems in your program.

You can use the debugger to execute your program one statement at a time and suspend the execution when control reaches a specified point. When the debugger stops a program, you can view the chain of the function calls, examine and change the values of the variables, and inspect the content of the registers.

The debugger communicates with the board through a hardware probe (such as the CodeWarrior TAP).

For general information about the debugger, including all of its common features and its visual interface, see *Working with the Debugger* chapter of the *CodeWarrior Common Features Guide* available in the `<CWInstallDir>\CW_APP\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

1.3.6 CodeWarrior profiling and analysis tools

The CodeWarrior profiling and analysis tools provide the visibility to an application that runs on the simulator and hardware. This visibility can help you understand how your application runs, as well as to identify the operational problems.

1.4 CodeWarrior development process

While working with the CodeWarrior IDE, you will proceed through the development stages familiar to all programmers, such as writing code, compiling and linking, and debugging. For complete information on the tasks such as editing, compiling, and linking and basic information on debugging, see *CodeWarrior Common Features Guide*.

In comparison to the traditional command-line environments, the CodeWarrior IDE helps you manage your work more effectively.

If you are unfamiliar with an integrated environment in general, or with the Eclipse IDE in particular, you may find the below listed topics in this section helpful. Each topic explains how one component of the CodeWarrior tools relates to a traditional command-line environment.

- [Project files](#)
- [Compiling](#)
- [Linking](#)
- [Editing code](#)
- [Debugging](#)

1.4.1 Project files

A CodeWarrior *project* is analogous to a set of make files, as a project can have multiple settings that are applied when building the program. For example, you can have one project that has both a debug version and a release version of your program. You can build one or the other, or both as per your requirement. The different settings that are used to launch your program within a single project are called *launch configurations*.

The IDE uses the **CodeWarrior Project** view to list all the files in a project that includes source code files and libraries.

You can easily add or remove the files. You can also assign the files to one or more different build configurations within the project, to manage the files common within the multiple build configurations.

The IDE manages all the interdependencies between the files automatically and tracks the files that are changed since the last build.

The IDE also stores the settings for the compiler and linker options for each build configuration. You can modify these settings using the IDE, or with `#pragma` statements in your code.

1.4.2 Compiling

To compile a source code file, it must be among the files that are the part of the current launch configuration. If the file is in the configuration, select it in the **CodeWarrior Projects** view and select **Project > Build Project** from the CodeWarrior IDE menu bar.

To automatically compile all the files in the current launch configuration after you modify them, select **Project > Build Automatically** from the CodeWarrior IDE menu bar.

1.4.3 Linking

Select **Project > Build Project** from the CodeWarrior IDE menu bar to link an object code to a final binary file. The **Build Project** command updates the selected project, then links the resulting object code into a final output file.

You can control the linker through the IDE. There is no need to specify a list of object files. The Workspace tracks all the object files automatically.

You can also modify the build configuration settings to specify the name of the final output file.

1.4.4 Editing code

The CodeWarrior IDE has an integral text editor designed for programmers. It supports the text files in ASCII, Microsoft® Windows®, and UNIX® formats.

To edit a file in a project, double-click the file name in the **CodeWarrior Projects** view. The CodeWarrior IDE opens the file in the editor associated with the file type.

The editor view has excellent navigational features that allow you to switch between related files, locate any particular function, mark any location within a file, or go to a specific line of code.

1.4.5 Debugging

Select **Run > Debug** from the CodeWarrior IDE menu bar to debug your project. This command downloads the current project's executable to the target board and starts a debug session.

NOTE

Before you start debugging your project, you must configure the debugger settings for the launch configuration by selecting **Run > Debug Configurations** from the CodeWarrior IDE menu bar. The CodeWarrior IDE uses the settings in the launch configuration to generate the debugging information and initiates the communication with the target board.

You can now use the debugger to step through the code of the program, view and change the value of variables, set breakpoints, and much more. For more information, see *Working with the Debugger* chapter of the *CodeWarrior Common Features Guide* available in the `<CWInstallDir>\CW_APP\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.



Chapter 2

Working with Projects

This chapter explains how to create, build and debug projects for the boards that are supported by the current release of CodeWarrior Development Studio for Advanced Packet Processing.

- [CodeWarrior bareboard project wizard](#)
- [Creating projects](#)
- [Building projects](#)
- [Debugging projects](#)
- [Deleting project](#)

2.1 CodeWarrior bareboard project wizard

The **CodeWarrior Bareboard Project Wizard** presents a series of pages that prompt you for the features and settings to be used for creating your program. This wizard also helps you specify other settings, such as whether the program executes on a simulator rather than actual hardware.

This topic describes the various pages that the **CodeWarrior Bareboard Project Wizard** displays as it assists you in creating a bareboard project.

NOTE

The pages that the wizard presents can differ, based on the project type or execution target you selected.

The pages of the **CodeWarrior Bareboard Project Wizard** are as follows:

- [Create a CodeWarrior Bareboard Project page](#)
- [Processor page](#)
- [Debug Target Settings page](#)
- [Build Settings page](#)

- [Configurations page](#)
- [Software Analysis page](#)

2.1.1 Create a CodeWarrior Bareboard Project page

Use this page to specify the project name and the directory where the project files are located. This figure shows the **Create a CodeWarrior Bareboard Project** page.

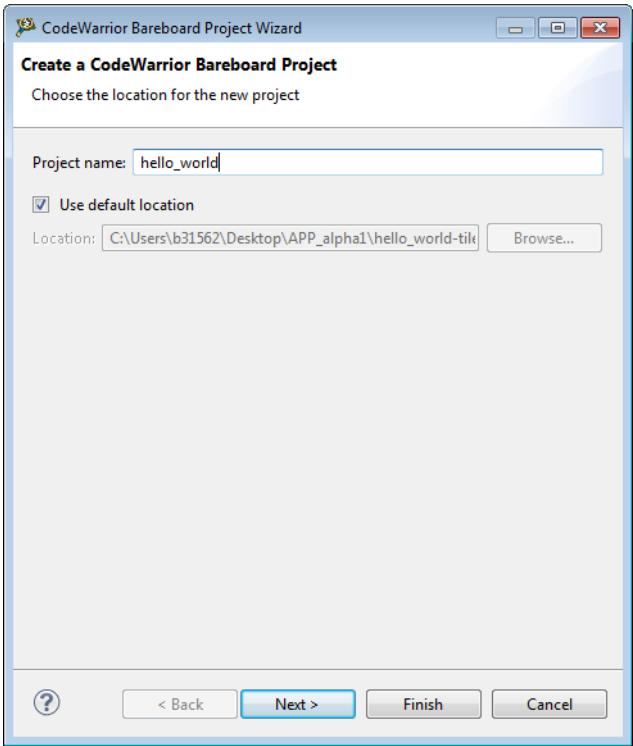


Figure 2-1. Create a CodeWarrior Bareboard Project page

The table below lists and describes the various options available on the **Create a CodeWarrior Bareboard Project** page.

Table 2-1. Create a CodeWarrior Bareboard Project page settings

Option	Description
Project name	Enter the name for the project in this text box.
Use default location	Select to choose the directory to store the files required to build the program. Use the Location option to select the desired directory.
Location	Specifies the directory that contains the project files. Use the Browse button to navigate to the desired directory. This option is only available when Use default location is cleared. Ensure that you append the path with the name of the project to create a new location for your project.

2.1.2 Processor page

This page displays the processors supported by the current installation. Use this page to specify the type of processor and the output for the new project. The figure listed below shows the **Processor** page.

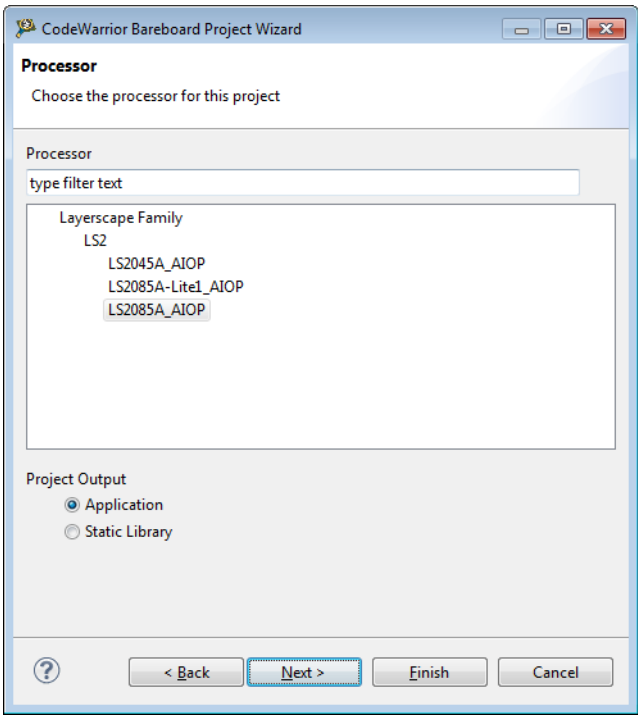


Figure 2-2. Processor page

The table below describes the various options available on the **Processor** page.

Table 2-2. Processor Page Settings

Option	Description
Processor	Expand the processor family tree and select a supported target. The toolchain uses this choice to generate code that makes use of processor-specific features, such as multiple cores.
Project Output	Select any one of the following supported project output: <ul style="list-style-type: none"> • Application-Select to create an application with ".elf" extension, that includes information related to the debug over a board. • Static Library-Select to create a library with ".a" extension, that can be included in other projects. Library files created using this option do not include board specific details.

2.1.3 Debug Target Settings page

This page displays the debugger connection types supported by the current installation. Use this page to specify the connection type and the launch configurations created for the new project.

NOTE

This wizard page may prompt you to either create a new remote system configuration or select an existing one.

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. For more information, see *CodeWarrior Common Features Guide* available in the `<CWInstallDir>\CW_APP\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

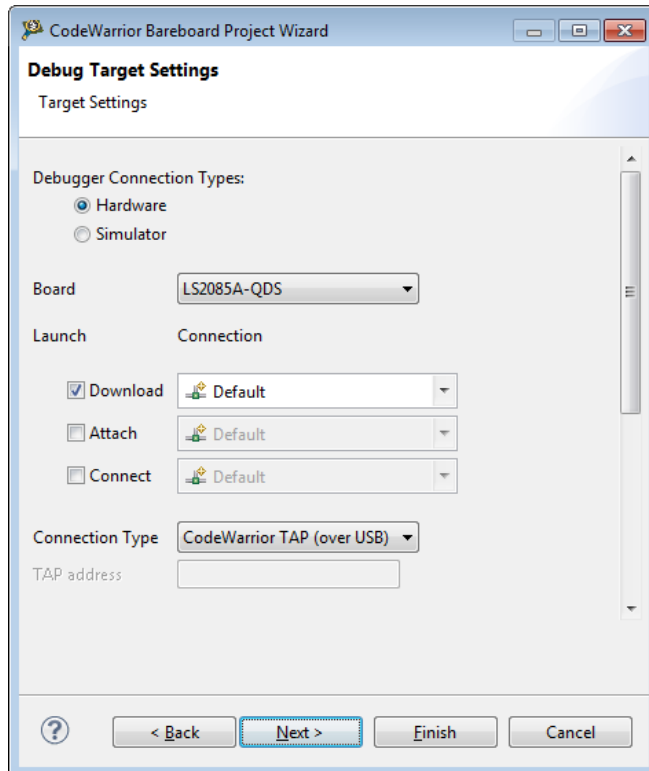


Figure 2-3. Debug Target Settings page

The table below describes the various options available on the **Debug Target Settings** page.

Table 2-3. Debug Target Settings page settings

Option	Description
Debugger Connection Types	Specifies the target on which the program executes on. <ul style="list-style-type: none"> • Hardware - Select to execute the program on a hardware board. • Simulator - Select to execute the program on a software simulator.
Board	Specifies the hardware supported by the selected processor.
Launch	Specifies the launch configurations and the corresponding connection, supported by the selected processor.
Connection Type	Specifies the interface to communicate with the target.
TAP address	Specify the IP address of the selected TAP device. This option is disabled and not supported in the current release.

2.1.4 Build Settings page

This page displays the toolchains supported by the current installation. Use this page to specify the toolchain and the output project type for the new project. The figure listed below shows the **Build Settings** page.

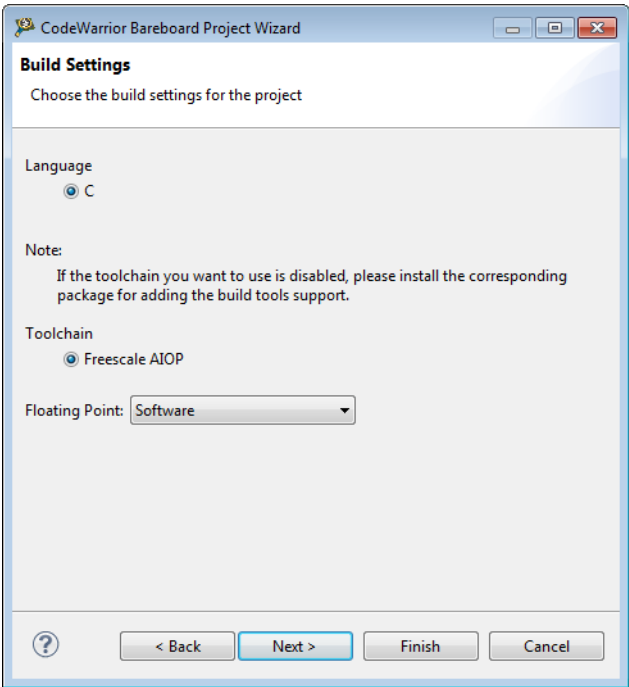


Figure 2-4. Build Settings page

The table below describes the various options available on the **Build Settings** page.

Table 2-4. Build Settings page setting

Option	Description
Language	Specifies the programming language used by the new project. The current installation supports the following languages: <ul style="list-style-type: none"> • C - Select to generate ANSI C-compliant startup code, and initializes global variables.
Toolchain	Specifies the toolchains supported by the current installation. Selected toolchain sets up the default compiler, linker, and libraries used to build the new project. Each toolchain generates code targeted for a specific platform.
Floating Point	Specifies how the compiler handles floating-point operations, encountered in the source code.

2.1.5 Configurations page

Use this page to specify the processing model and the processor core that executes the project. The figure listed below shows the **Configurations** page.

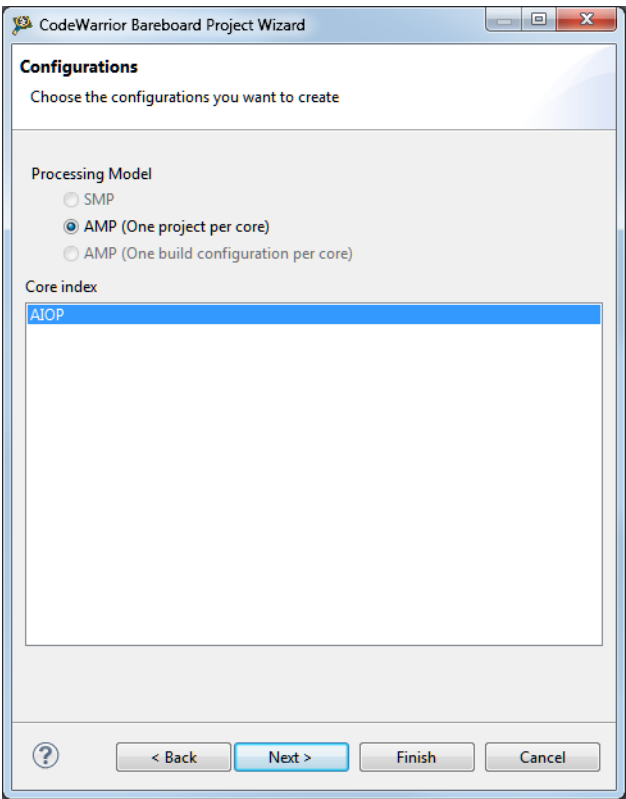


Figure 2-5. Configurations page

The table below describes the various options available on the **Configurations** page.

Table 2-5. Configurations page setting

Option	Description
Processing Model	<p>The current installation supports the following processing models:</p> <ul style="list-style-type: none"> • SMP - Select this option to generate a single project for the selected cores. The cores share the same interrupt vector, text, data sections and heap memory. Each core has its own, dedicated stack. A single initialization file should be executed for each core. <p>NOTE: The SMP option is disabled and not available for selection in the current release.</p> <ul style="list-style-type: none"> • AMP (one project per core) - Select this option to generate a separate project for each selected core. The option will also set the core index for each project based on the core selection. • AMP (one build configuration per core) - Select this option to generate one project with multiple targets, each containing an lcf file for the specified core.
Core index	Select the processor core that executes the project.

2.1.6 Software Analysis page

Use this page to enable the GNU coverage support to analyze the source code that runs on the AIOP or MC cores, or to enable the stack usage estimation at the linker level. The figure listed below shows the **Software Analysis** page.

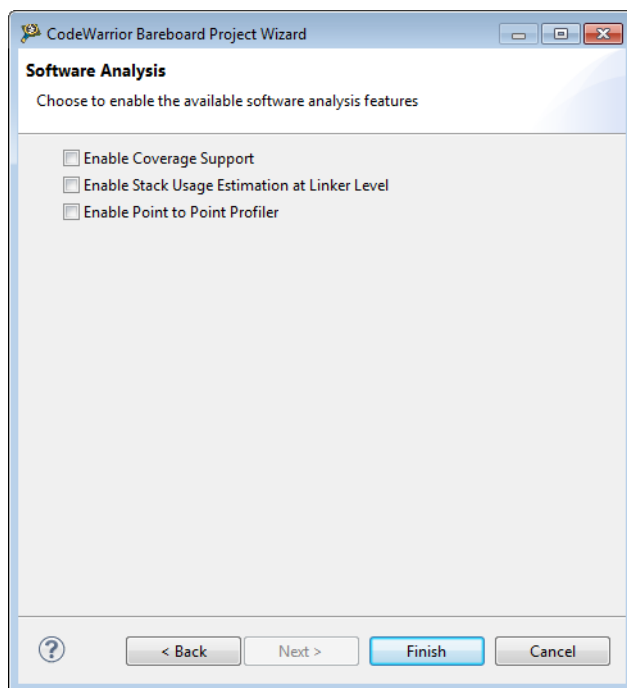


Figure 2-6. Software Analysis page

The table below describes the various options available on the **Software Analysis** page.

Table 2-6. Software Analysis page setting

Option	Description
Enable Coverage Support	Controls the Trace and profile analysis tool. For more details, refer Tracing and Profile Analysis .
Enable Stack Usage Estimation at Linker Level	Controls the generation of stack usage information in the linker map file.
Enable Point to Point Profiler	Controls performance gathering. For more details, refer Point to Point Profiler .

2.2 Creating projects

This section explains how to use the **CodeWarrior Bareboard Project Wizard** to quickly create new projects with default settings (build and launch configurations).

This chapter explains the following topics:

- [Creating CodeWarrior bareboard application project](#)
- [Creating CodeWarrior bareboard library project](#)

2.2.1 Creating CodeWarrior bareboard application project

To create an application project using the **CodeWarrior Bareboard Project Wizard**, perform these steps:

1. Select **Start > All Programs > Freescale CodeWarrior > CW4NET *vnumber* > CodeWarrior for APP**, where *number* is the version number of your product.

The **Workspace Launcher** dialog appears, prompting you to select a workspace to use.

NOTE

Click **Browse** to change the default location for workspace folder. You can also select the **Use this as the default and do not ask again** checkbox to set default or selected path as the default location for storing all your projects.

2. Click **OK**.

The default workspace is accepted. The CodeWarrior IDE launches and the **Welcome** page appears.

NOTE

The **Welcome** page appears only if the CodeWarrior IDE or the selected Workspace is opened for the first time. Otherwise, the Workbench window appears.

3. Click **Go to Workbench**, on the **Welcome** page.

The Workbench window appears.

4. Select **File > New > CodeWarrior Bareboard Project Wizard**, from the CodeWarrior IDE menu bar.

The **CodeWarrior Bareboard Project Wizard** launches and the [Create a CodeWarrior Bareboard Project](#) page appears.

5. Specify a name for the new project in the **Project name** text box.

For example, enter the project name as `application_project`.

6. If you do not want to create your project in the default workspace:
 - a. Clear the **Use default location** checkbox.
 - b. Click **Browse** and select the desired location from the **Browse For Folder** dialog box.
 - c. In the **Location** text box, append the location with the name of the directory in which you want to create your project.

NOTE

An existing directory cannot be specified for the project location.

7. Click **Next**.

The **Processor** page appears.

8. Select the target processor for the new project, from the **Processor** list.
9. Select **Application** from the **Project Output** group, to create an application with `.elf` extension, that includes information required to debug the project.
10. Click **Next**.

The **Debug Target Settings** page appears.

11. Select a supported connection type, from the **Debugger Connection Types** group. Your selection determines the launch configurations that you can include in your project.
12. Select the hardware or simulator, you plan to use, from the **Board** drop-down list.

NOTE

Hardware or simulators that supports the target processor you selected on the **Processor** page, are only available for selection.

13. Select the launch configurations, that you want to include in your project and the corresponding connection.
14. Select the interface to communicate with the hardware, from the **Connection Type** dropdown.
15. Specify the IP address of the TAP device in the **TAP address** text box.

NOTE

This option is disabled and cannot be edited for **CodeWarrior TAP (over USB)** option.

16. Click **Next**.

The **Build Settings** page appears.

17. Select the programming language, you want to use, from the **Language** group.

The language you select determines the libraries that are linked with your program and the contents of the main source file that the wizard generates.

18. Select a toolchain from the **Toolchain** group.

Selected toolchain sets up the default compiler, linker, and libraries used to build the new project. Each toolchain generates code targeted for a specific platform.

19. Select an option from the **Floating Point** drop-down list, to prompt the compiler to handle the floating-point operations by generating instructions for the selected floating-point unit.
20. Click **Next**.

The **Configurations** page appears.

21. Select a processing model option from the **Processing Model** group.

NOTE

The **SMP** option is disabled and cannot be selected in the current release.

- Select **AMP (One project per core)** to generate a separate project for each selected core. The option will also set the core index for each project based on the core selection.
 - Select **AMP (One build configuration per core)** to generate one project with multiple targets, each containing an `.lcf` file for the specified core.
22. Select the processor core that executes the project, from the **Core index** list.
 23. Click **Next**.

The **Software Analysis** page appears.

24. If you plan to analyze the source code, select **Enable Coverage Support**.
25. If you plan to generate stack usage information in the linker map file, select **Enable Stack Usage Estimation at Linker Level**.
26. Select **Enable Point to Point Profiler**, if you plan to enable performance gathering.
27. Click **Finish**.

The wizard creates an application project according to your specifications. You can access the project from the **CodeWarrior Projects** view.

The new project is ready for use. You can now customize the project by adding your own source code files, changing debugger settings and adding libraries.

2.2.2 Creating CodeWarrior bareboard library project

To create a library project using the **CodeWarrior Bareboard Project Wizard**, perform these steps:

1. Select **Start > All Programs > Freescale CodeWarrior > CW4NET *vnumber* > CodeWarrior for APP**, where *number* is the version number of your product.

The **Workspace Launcher** dialog appears, prompting you to select a workspace to use.

NOTE

Click **Browse** to change the default location for workspace folder. You can also select the **Use this as the default and do not ask again** checkbox to set default or selected path as the default location for storing all your projects.

2. Click **OK**.

The default workspace is accepted. The CodeWarrior IDE launches and the **Welcome** page appears.

NOTE

The **Welcome** page appears only if the CodeWarrior IDE or the selected Workspace is opened for the first time. Otherwise, the Workbench window appears.

3. Click **Go to Workbench**, on the **Welcome** page.

The Workbench window appears.

4. Select **File > New > CodeWarrior Bareboard Project Wizard**, from the CodeWarrior IDE menu bar.

The IDE launches the wizard and the **Create a CodeWarrior Bareboard Project** page appears.

5. Specify a name for the new project in the **Project name** text box.

For example, enter the project name as `application_project`.

6. If you do not want to create your project in the default workspace:
 - a. Clear the **Use default location** checkbox.
 - b. Click **Browse** and select the desired location from the **Browse For Folder** dialog box.

- c. In the **Location** text box, append the location with the name of the directory in which you want to create your project.

NOTE

An existing directory cannot be specified for the project location.

7. Click **Next**.

The **Processor** page appears.

8. Select the target processor for the new project, from the **Processor** list.
9. Select **Static Library** from the **Project Output** group, to create a library with .a extension, that can be included in other projects. Library files created using this option do not include board specific details.
10. Click **Next**.

The **Build Settings** page appears.

11. Select the programming language, you want to use, from the **Language** group.

The language you select determines the libraries that are linked with your program and the contents of the main source file that the wizard generates.

12. Select the architecture type used by the new project, from the **Build Tools Architecture** group. This option may not be available for some target processors selected on the **Processor** page.
13. Select a toolchain from the **Toolchain** group.

Selected toolchain sets up the default compiler, linker, and libraries used to build the new project. Each toolchain generates code targeted for a specific platform.

14. Select an option from the **Floating Point** drop-down list, to prompt the compiler to handle the floating-point operations by generating instructions for the selected floating-point unit.
15. Click **Finish**.

The wizard creates a library project according to your specifications. You can access the project from the **CodeWarrior Projects** view.

The new library project is ready for use. You can now customize the project to match your requirements.

2.3 Building projects

The CodeWarrior IDE supports two modes of building projects:

- [Manual-build mode](#)
- [Auto-build mode](#)

2.3.1 Manual-build mode

In the large workspaces, building the entire workspace can take a long time if users make changes with a significant impact on dependent projects. Often there are only a few projects that really matter to a user at a given time.

To build only the selected projects, and any prerequisite projects that need to be built in, select **Project > Build Project** from the CodeWarrior IDE menu bar. Alternatively, right-click on the selected project in the **CodeWarrior Projects** view and select **Build Project** from the context menu.

To build the entire workspace, select **Project > Build All** from the IDE menu bar.

2.3.2 Auto-build mode

You can configure the CodeWarrior IDE to compile the source files automatically. When auto-build is enabled, builds will occur automatically in the background every time you make changes to the files. To automatically build all the projects in a workspace, select **Project > Build Automatically** from the CodeWarrior IDE menu bar.

If auto-build needs to be disabled then select **Project > Build Automatically** from the CodeWarrior IDE menu bar.

NOTE

It is advised that you do not use the **Build Automatically** option for C/C++ development. Using this option results in building the entire project whenever you save a change to the makefile or source files. This can take a significant amount of time for very large projects.

2.4 Debugging projects

The **CodeWarrior Bareboard Project Wizard** sets the debugger settings of the project's launch configurations to default values. You can change these default values based on your requirements.

To modify the debugger settings and start debugging a CodeWarrior project, perform these steps:

1. Launch the CodeWarrior IDE.
2. From the CodeWarrior IDE menu bar, select **Run > Debug Configurations**.
CodeWarrior IDE uses the settings in the launch configuration to generate debugging information and initiate communications with the target board.

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

3. Expand the **CodeWarrior** configuration.
4. From the expanded list, select the debug configuration that you want to modify.
5. Select a predefined debug session type or custom type for maximum flexibility.
6. From the **Target settings** group, select **AIOP-0** for targeting AIOP.
7. Click Apply to save the new settings.

Tip

You can click **Revert** to undo any of the unsaved changes. CodeWarrior IDE restores the last set of saved settings to all pages of the **Debug Configurations** dialog. Also, the IDE disables revert until you make new pending changes.

8. Click **Debug** to start the debugging session.

The **Debug** perspective appears.

You just modified the debugger settings and initialized a debugging session.

2.5 Deleting project

To delete a project, follow these steps:

1. Select the project you want to delete in the **CodeWarrior Projects** view.
2. Select **Edit > Delete**.

The **Delete Resources** dialog appears.

NOTE

Alternatively, you can also select **Delete** from the context menu that appears when you right-click the project.

3. Check **Delete project contents on disk (cannot be undone)** option to delete the project contents permanently.
4. Click **OK**.

You just finished deleting a project using the CodeWarrior IDE.

Chapter 3

Debug Configuration

A CodeWarrior project can have multiple associated debug configurations. A debug configuration is a named collection of settings that the CodeWarrior tools use.

Debug configurations let you specify settings, such as:

- the files that belong to the debug configuration
- behavior of the debugger and the related debugging tools

This chapter explains:

- [Using CodeWarrior debug configuration tabs](#)
- [Customizing debug configurations](#)
- [Reverting debug configuration settings](#)

3.1 Using CodeWarrior debug configuration tabs

This section lists the debugger settings specific to developing software using CodeWarrior Development Studio for Advanced Packet Processing.

NOTE

As you modify a debug configuration's debugger settings, you create pending, or unsaved, changes to that debug configuration. To save the pending changes, you must click the **Apply** button of the **Debug Configurations** dialog box, or click **Close** button and then the **Yes** button.

The following table lists the various debugger setting panels.

Table 3-1. Debug configuration tabs

Main	
Arguments	
Debugger	Debug
	Download
	PIC
	System Call Services
	Other Executables
	Symbolics
	OS Awareness
Trace and Profile	
Source	
Environment	
Common	

3.1.1 Main

Use this tab to specify the project and the application you want to run or debug.

You also specify a remote system configuration on this tab. The remote system configuration is separated into connection and system configurations allowing you to define a single system configuration that can be referred to by multiple connection configurations. The launch configurations refer to a connection configuration, which in turn refers to a system configuration.

NOTE

The options displayed on the Main tab varies depending on the selected debug session type.

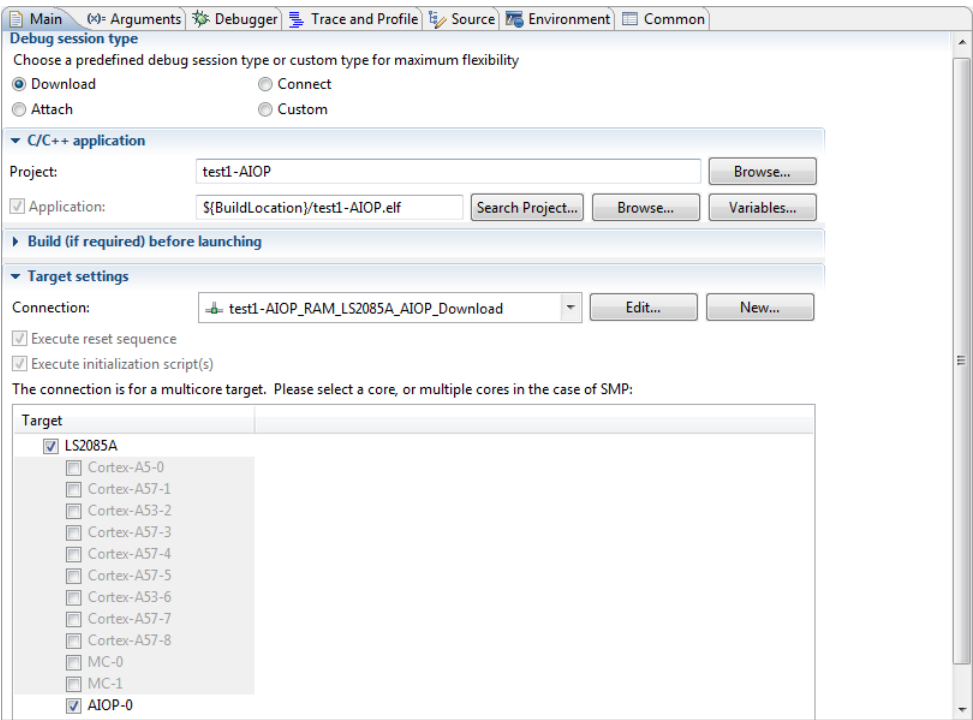


Figure 3-1. Debug Configurations - Main tab

The table below lists the various options available on the **Main** tab page.

Table 3-2. Main tab options

Option	Description
Debug session type	<p>Specifies the options to initiate a debug session using pre-configured debug configurations. The options include:</p> <ul style="list-style-type: none"> • Download - Resets the target if the debug configuration specifies the action. Further, the command stops the target, (optionally) runs an initialization script, downloads the specified ELF file, and modifies the program counter(PC). • Attach - Assumes that code is already running on the board and therefore does not run a target initialization file. The state of the running program is undisturbed. The debugger loads symbolic debugging information for the current build target's executable. The result is that you have the same source-level debugging facilities you have in a normal debug session (the ability to view source code and variables, and so on). The function does not reset the target, even if the launch configuration specifies this action. Further, the command loads symbolics, does not stop the target, run an initialization script, download an ELF file, or modify the program counter (PC). <p>NOTE: The debugger does not support restarting debugging sessions that you start by attaching the debugger to a process.</p> <ul style="list-style-type: none"> • Connect - Runs the target initialization file specified in the RSE configuration to set up the board before connecting to it. The Connect debug session type does

Table continues on the next page...

Table 3-2. Main tab options (continued)

Option	Description
	<p>not load any symbolic debugging information for the current build target's executable thereby, denying access to source-level debugging and variable display. The Connect command resets the target if the launch configuration specifies this action. Further, the command stops the target, (optionally) runs an initialization script, does not load symbolics, download an ELF file, or modify the program counter(PC).</p> <p>NOTE: The default debugger configuration causes the debugger to cache symbolics between sessions. However, selecting the Connect option invalidates this cache. If you must preserve the contents of the symbolics cache, and you plan to use the Connect option, clear the Cache Symbolics Between Sessions check box in the Symbolics tab page.</p> <ul style="list-style-type: none"> • Custom - Provides user an advantage to create a custom debug configuration.
C/C++ application	<p>Specifies the settings for the C/C++ application. The options include:</p> <ul style="list-style-type: none"> • Project - Specifies the name of the project associated with the selected debug launch configuration. Click Browse to select a different project. • Application - Specifies the name of the C or C++ application executable. <p>NOTE: This option is disabled when Connect debug session type is selected.</p> <ul style="list-style-type: none"> • Search Project - Click to open the Program Selection dialog box and select a binary. <p>NOTE: This option is disabled when Connect debug session type is selected.</p> <ul style="list-style-type: none"> • Variables - Click to open the Select build variable dialog box and select the build variables to be associated with the program. <p>NOTE: The dialog box displays an aggregation of multiple variable databases and not all these variables are suitable to be used from a build environment. Given below are the variables that should be used:</p> <p>ProjDirPath - returns the absolute path of the current project location in the file system</p> <p><code>\${ProjDirPath}/Source/main.c"</code></p> <p>workspace_loc - returns the absolute path of a workspace resource in the file system, or the location of the workspace if no argument is specified</p> <p><code>\${workspace_loc:/ProjectName/Source main.c"\$ {workspace_loc}</code></p> <p>Gnu_Make_Install_Dir - returns the absolute path of the GNU make.exe tool</p> <p><code>\${Gnu_Make_Install_Dir}\make.exe</code></p>

Table continues on the next page...

Table 3-2. Main tab options (continued)

Option	Description
	<p>NOTE: This option is disabled when Connect debug session type is selected.</p>
Build (if required) before launching	<p>Controls how auto build is configured for the launch configuration. Changing this setting overrides the global workspace setting and can provide some speed improvements. NOTE: These options are set to default and collapsed when Connect debug session type is selected. The options include:</p> <ul style="list-style-type: none"> • Build configuration - Specifies the build configuration either explicitly or use the current active configuration. • Select configuration using `C/C++ Application` - Select/clear to enable/disable automatic selection of the configuration to be built, based on the path to the program. • Enable auto build - Enables auto build for the debug configuration which can slow down launch performance. • Disable auto build - Disables auto build for the debug configuration which may improve launch performance. No build action will be performed before starting the debug session. You have to rebuild the project manually. • Use workspace settings - Uses the global auto build settings. • Configure Workspace Settings - Opens the Launching preference panel where you can change the workspace settings. It will affect all projects that do not have project specific settings.
Target settings	<p>Specifies the connection and other settings for the target. The options include:</p> <ul style="list-style-type: none"> • Connection - Specifies the applicable Remote System configuration. • Edit - Click to edit the selected Remote System configuration. • New - Click to create a new Remote System configuration for the selected project and application. • Execute reset sequence - Select to apply reset settings, specified in the target configuration, when attaching to a target. Alternatively, clear the option to ignore reset settings. <p>NOTE: This option is not available when Connect debug session type is selected.</p> <ul style="list-style-type: none"> • Execute initialization script(s) - Select to execute the initialization script(s), specified in the target configuration, when attaching to a target. Alternatively, clear the option to ignore the initialization script(s). <p>NOTE: This option is not available when Connect debug session type is selected.</p> <ul style="list-style-type: none"> • Target - Select the target core.

3.1.2 Arguments

Use this tab to specify the program arguments that an application uses and the working directory for a run or debug configuration.

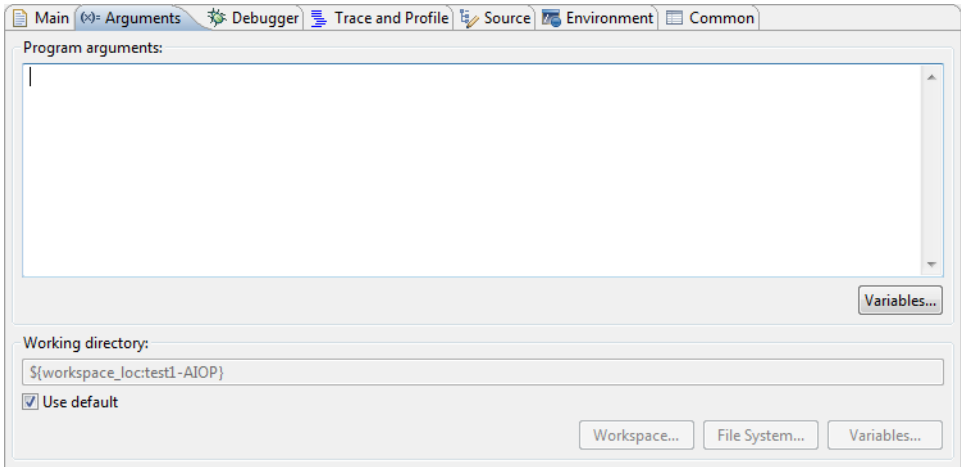


Figure 3-2. Debug Configurations - Arguments tab

The table below lists the various options available on the Arguments tab page.

Table 3-3. Arguments tab options

Option	Description
Program arguments	Specifies the arguments passed on the command line.
Variables	Click to select variables by name to include in the program arguments list.
Working directory	Specifies the run/debug configuration working directory.
Use default	Select to specify the local directory or clear to specify a different workspace, a file system location, or variable.
Workspace	Click to specify the path of, or browse to, a workspace relative working directory.
File System	Click to specify the path of, or browse to, a file system directory.
Variables	Click to specify variables by name to include in the working directory.

3.1.3 Debugger

Use this tab to configure debugger settings. The **Debugger** tab presents various debugger configuration options.

NOTE

The content in the Debugger tab page changes, depending on the **Debug session type** selected on the **Main** tab page.

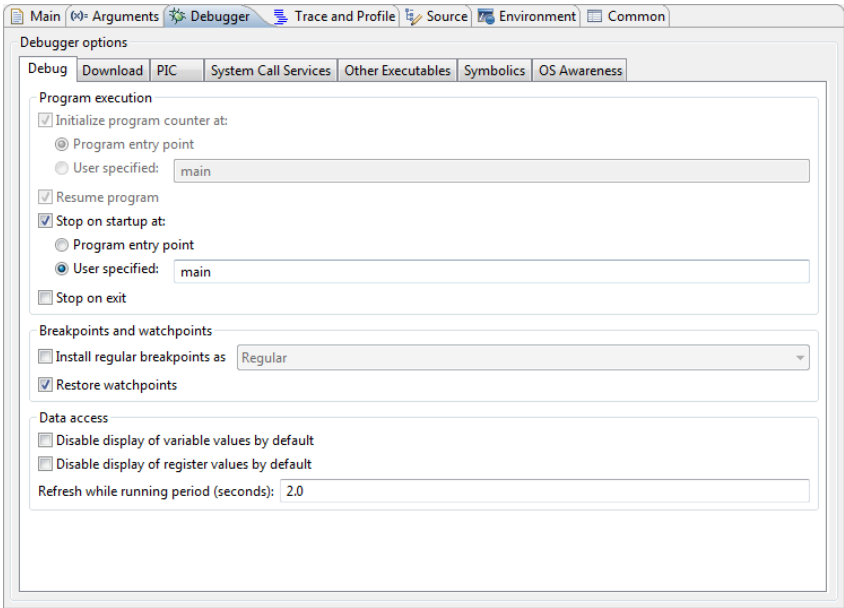


Figure 3-3. Debug Configurations - Debugger tab

The table below lists the various options available on the **Debugger** tab page.

Table 3-4. Debugger tab options

Option	Description
Debugger Options	<p>Displays configuration options specific to the selected debugger type. Refer to the following sections for more details:</p> <ul style="list-style-type: none"> • Debug • Download • PIC • System Call Services • Other Executables • Symbolics • OS Awareness

3.1.3.1 Debug

Use this page to specify the program execution options, breakpoint and watchpoint options, and target access behavior.

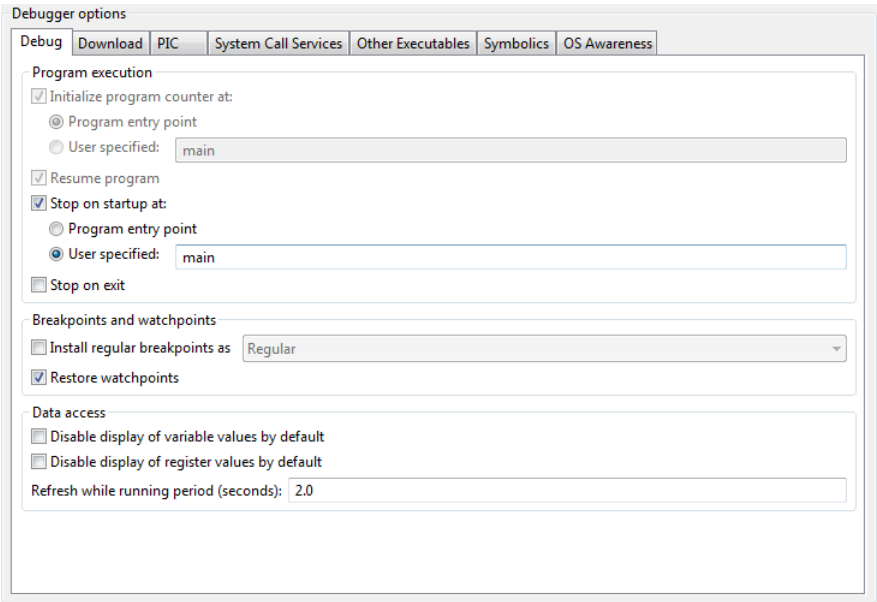


Figure 3-4. Debugger options - Debug page

The table below lists the various options available on the **Debug** page.

NOTE

The options displayed on the **Debug** tab varies depending on the **Debug session type** selection on the **Main** tab page.

Table 3-5. Debug page options

Option	Description
Initialize program counter at	<p>Controls the initialization of program counter.</p> <ul style="list-style-type: none"> • Program entry point - Select to initialize the program counter at a specified program entry pont. • User specified - Select to initialize the program counter at a user-specified function. The default location is <code>main</code>. <p>NOTE: Disabling this option will also disable the Resume program and Stop on startup at options.</p>
Resume program	<p>Select to resume the execution after the program counter is initialized. NOTE: Disabling this option will also disable the Stop on startup at option.</p>
Stop on startup at	<p>Stops program at specified location. When cleared, the program runs until you interrupt it manually, or until it hits a breakpoint.</p> <ul style="list-style-type: none"> • Program entry point - Select to stop the debugger at a specified program entry pont. • User specified - Select to stop the debugger at a user-specified function. The default location is <code>main</code>.

Table continues on the next page...

Table 3-5. Debug page options (continued)

Option	Description
Stop on exit	Check this option to have the debugger set a breakpoint at the code's exit point. For multicore projects, when you set this option for one project on one core, it is set for projects on the other cores. Clear this option to prevent the debugger from setting a breakpoint at the code's exit point.
Install regular breakpoints as	Check this option to install breakpoints as either: <ul style="list-style-type: none"> Regular Hardware Software Clear this option to install breakpoints as Regular breakpoints.
Restore watchpoints	Check this option to restore previous watchpoints.
Disable display of variable values by default	Check this option to disable the display of variable values Clear this option to enable the display of variable values
Disable display of register values by default	Check this option to disable the display of register values Clear this option to enable the display of register values
Refresh while running period (seconds)	Specifies the refresh period used when a view is configured to refresh, while the application is running.

3.1.3.2 Download

Use this page to specify which executable code sections the debugger downloads to the target, and whether the debugger should read back those sections and verify them.

NOTE

Selecting all options in the **Perform standard download** group significantly increases the download time.

The **First** options apply to the first debugging session. The **Subsequent** runs options apply to subsequent debugging sessions. The **Download** options control whether the debugger downloads the specified **Program Section** data type to the target hardware. The **Verify** options control whether the debugger reads the specified program section data type from the target hardware and compares the read data against the data written to the device.

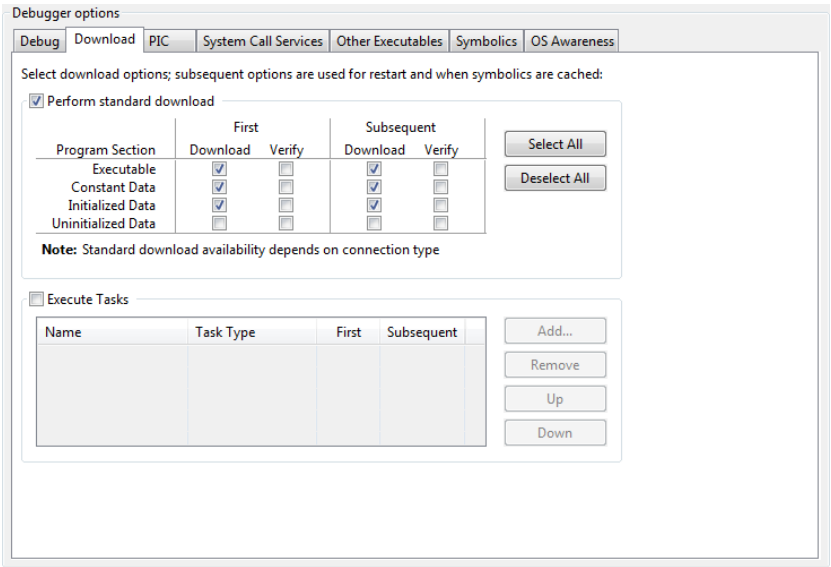


Figure 3-5. Debugger options - Download page

The table below lists the various options available on the **Download** page.

Table 3-6. Download page options

Option	Description
Perform standard download	Controls download of the target application using memory write command
Executable	Controls downloading and verification for executable sections. Check appropriate check boxes to specify downloading and verifications, for initial launch and for subsequent runs
Constant Data	Controls downloading and verification for constant-data sections. Check appropriate check boxes to specify downloading and verifications, for initial launch and for subsequent runs
Initialized Data	Controls downloading and verification for initialized-data sections. Check appropriate check boxes to specify downloading and verifications, for initial launch and for subsequent runs
Uninitialized Data	Controls downloading and verification for uninitialized-data sections. Check appropriate check boxes to specify downloading and verifications, for initial launch and for subsequent runs
Execute Tasks	Enables the execution of target tasks
Name	For target tasks, this is the name of the target task as seen in the Target Task view. For Debugger Shell scripts, this is the path to the CLDE script
Task Type	Contains either Debugger Shell scripts or target tasks (such as Flash Programmer)
Add	Adds a download task that can be either a target task or Debugger shell script
Remove	Removes the selected target task or debugger shell script
Up	Moves the selected task up the list

Table continues on the next page...

Table 3-6. Download page options (continued)

Option	Description
Down	Moves the selected task down the list

3.1.3.3 PIC

Use this page to specify an alternate address at which the debugger loads the position independent code (PIC) module onto target memory. Usually, PIC is linked in such a way that the entire image starts at address 0x00000000.

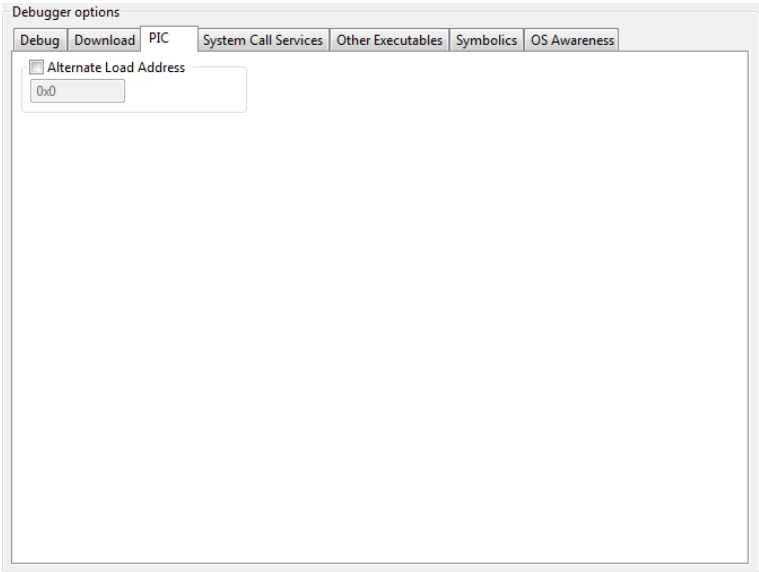


Figure 3-6. Debugger options - PIC page

The table below lists the various options available on the PIC page.

Table 3-7. PIC page options

Option	Description
Alternate Load Address	Specify the starting address at which the debugger loads your program. You can also use this setting when you have an application which is built with ROM addresses and then relocates itself to RAM (such as U-Boot). Specifying a relocation address lets the debugger map the symbolic debugging information contained in the original ELF file (built for ROM addresses) to the relocated application image in RAM. Clear the check box to have the debugger load your program at a default starting address.

NOTE

The debugger does not verify whether your code can execute at the specified address. As a result, the PIC generation settings of the compiler, linker and your program's startup routines must correctly set any base registers and perform any required relocations.

3.1.3.4 System Call Services

Use this page to activate the debugger's support for system calls and to select options that define how the debugger handles system calls. The CodeWarrior debugger provides system call support over JTAG. System call support lets bareboard applications use the functions of host OS service routines. This feature is useful if you do not have a board support package (BSP) for your target board.

The host debugger implements these services. Therefore, the host OS service routines are available only when you are debugging a program on a target board or simulator.

NOTE

The OS service routines provided must comply with an industry-accepted standard. The definitions of the system service functions provided are a subset of Single UNIX Specification (SUS).

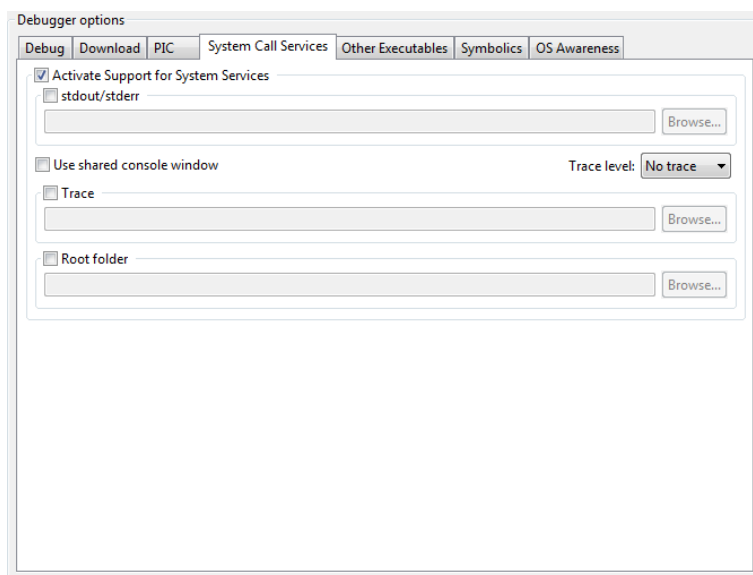


Figure 3-7. Debugger options -System Call Services page

The table below lists the various options available on the **System Call Services** page.

Table 3-8. System Call Services page options

Option	Description
Activate Support for System Services	Check this option to enable support for system services. All the other options on the System Call Services panel are enabled only if you check this check box.
stdout/stderr	By default, the output written to stdout and stderr appears in a CodeWarrior IDE "console" window. To redirect console output to a file, check the stdout/stderr check box. Click Browse to display a dialog box and specify the path and name of this file.
Use shared console window	Check this option if you wish to share the same console window between different debug targets. This setting is useful in multi-core or multi-target debugging.
Trace level	Use this drop-down list to specify the system call trace level. The place where the debugger displays the traced system service requests is determined by the Trace check box. The system call trace level options available are: <ul style="list-style-type: none"> No Trace - system calls are not traced Summary - the requests for system services are displayed Detailed - the requests for system services are displayed along with the arguments/parameters of the request
Trace	By default, traced system service requests appear in a CodeWarrior IDE "console" window. To log traced system service requests to a file, check the Trace check box. Click Browse to display a dialog box and define the path and name of this file. In a project created through the CodeWarrior Bareboard Project wizard, use the library syscall.a rather than a UART library for handling the output.
Root folder	The directory on the host system which contains the OS routines that the bareboard program uses for system calls.

3.1.3.5 Other Executables

Use this page to specify additional ELF files to download or debug in addition to the main executable file associated with the launch configuration.

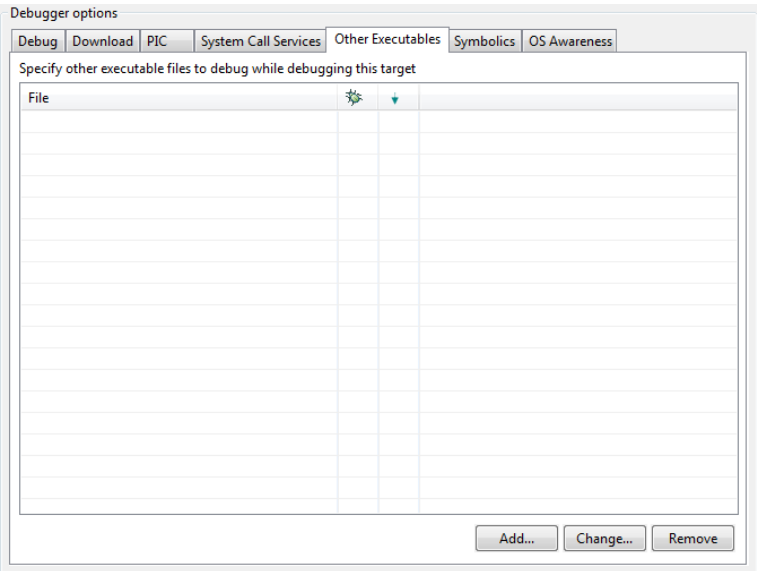


Figure 3-8. Debugger options - Other Executables page

The table below lists the various options available on the **Other Executables** page.

Table 3-9. Other Executables page options



Option	Description
File	Shows files and projects that the debugger uses during each debug session. The Debug column () - If this option is checked the debugger loads symbolics for the file. If you clear this option, the IDE does not load symbolics for the file. The Download column () - If this option is checked the debugger downloads the file to the target device. If you clear this option, the debugger does not download the file to the target device.
Add	Click to open the Debug Other Executable dialog box. Use the dialog box to specify the following settings: <ul style="list-style-type: none"> Specify the location of the additional executable - Enter the path to the executable file that the debugger controls in addition to the current project's executable file. Alternatively, click the Workspace, File System or Variables button to specify the file path. Load Symbols - Check this 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. Download to Device - Check this option to have the debugger download the specified file to the target device. Specify the path of the file in the Specify the remote download path text box. 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. OK - Click to add the information that you specify in the Debug Other Executable dialog box to the File list.

Table continues on the next page...

Table 3-9. Other Executables page options (continued)

Option	Description
Change	Click to open the Debug Other Executable dialog box. The dialog box shows the current settings for selected executable file in the File list column. Change this information as required and click the OK button to update the entry in the File list.
Remove	Click to remove the entry currently selected in the File list.

3.1.3.6 Symbolics

Use this 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, is beneficial when you debug a large-size application.

Consider a situation in which the debugger loads symbolics for a large application, but does not download content to a hardware device and the project uses custom makefiles with several build steps to generate this application. In such a situation, caching symbolics helps speed up the debugging process. The debugger uses the readily available 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 significant benefit for large applications, where doing so speeds up application-launch time. If you debug a small application, caching symbolics does not significantly improve the launch time.

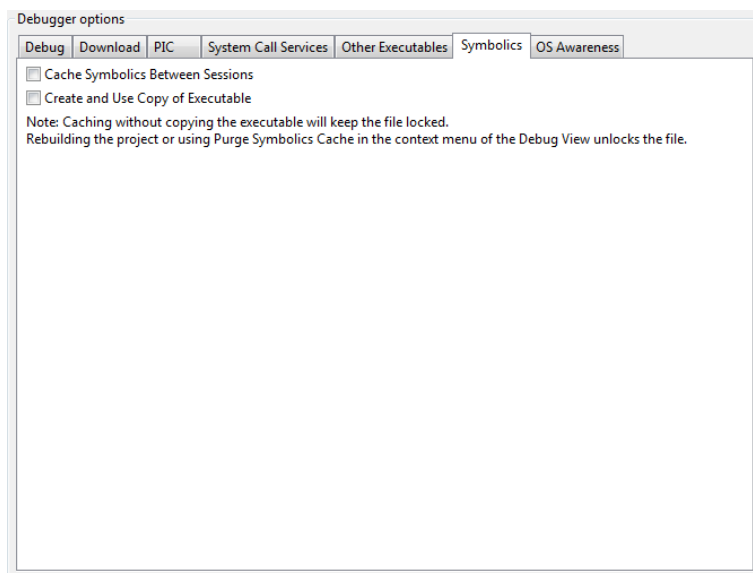


Figure 3-9. Debugger options - Symbolics page

The table below lists the various options available on the **Symbolics** page.

Table 3-10. Symbolics page options

Option	Description
Cache Symbolics Between Sessions	Check this option to have the debugger cache symbolics between debugging sessions. If you check this check box and clear the Create and Use Copy of Executable check box, the executable file remains locked after the debugging session ends. In the Debug view, right-click the locked file and select Un-target Executables to have the debugger delete its symbolics cache and release the file lock. The IDE enables this menu command when there are currently unused cached symbolics that it can purge. Clear this option so that the debugger does not cache symbolics between debugging sessions.
Create and Use Copy of Executable	Check this option to have the debugger create and use a copy of the executable file. Using the copy helps avoid file-locking issues with the build system. If you check this check box, the IDE can build the executable file in the background during a debugging session. Clear this option so that the debugger does not create and use a copy of the executable file.

3.1.3.7 OS Awareness

Use this page to specify the operating system (OS) that resides on the target device.

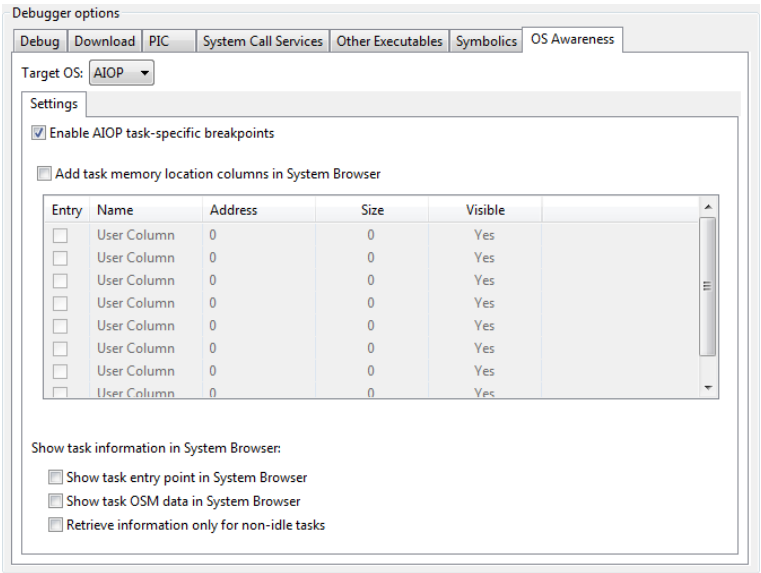


Figure 3-10. Debugger options - OS Awareness page

The table below lists the options available on the **OS Awareness** page.

Table 3-11. OS Awareness page options

Option		Description
Target OS	Use the Target OS list box to specify the operating system that runs on the target device. For details on activating the AIOP task awareness services, refer to the AIOP task aware debugging section.	
Enable AIOP task-specific breakpoints	Controls support for AIOP task-specific breakpoint types such as per TaskX Global Halt or task level.	
Add task memory location columns in System Browser	Entry	Controls if the column should be enabled and displayed in the System Browser view.
	Name	Specifies the column name. The text specified should be unique and can be alphanumeric.
	Address	Specifies the start memory location address. The specified address can be of hex (FF00 or 0xFF0 or decimal formats.
	Size	Specifies the size, in bytes, to read. The specified size can be of hex (FF00 or 0xFF) or decimal formats in the range of 1 - 1024.
	Visible	Controls the visibility of the column in the System Browser view.
Show task information in System Browser	Controls display of AIOP task information in System Browser view. For more details, refer Viewing task entry point and OSM data in System Browser .	
	Show task entry point in System Browser	Controls creation of custom column in the System Browser view to display task entry point.
	Show task OSM data in System Browser	Controls display of Order scope manager (OSM) data of each task in the System Browser view.

Table continues on the next page...

Table 3-11. OS Awareness page options (continued)

Option		Description
	Retrieve information only for non-idle tasks	Controls retrieval of information about the active tasks only. Select this option to improve System Browser performance.

3.1.4 Trace and Profile

Use this tab to configure the hardware trace collection.

NOTE

Hardware trace collection is currently supported only by projects configured to work with hardware emulators. The page may appear different for projects created to work on simulator targets.

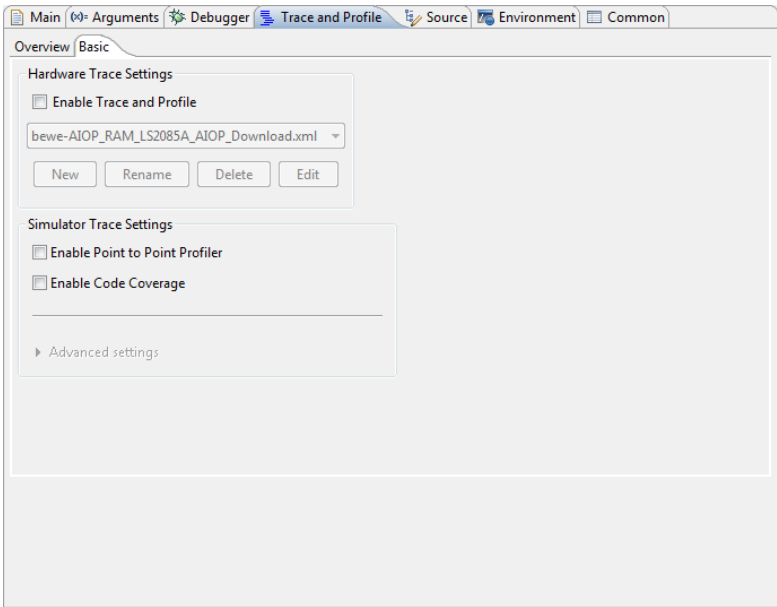


Figure 3-11. Debug Configurations - Trace and Profile tab

The table below lists the various options available on the **Trace and Profile** tab page.

Table 3-12. Trace and Profile tab options

Option	Description
Overview	Provides an overview of the Hardware trace support in CodeWarrior.

Table continues on the next page...

Table 3-12. Trace and Profile tab options (continued)

Option	Description
Basic	
Enable Trace and Profile	Check to enable collection of hardware trace.
New	Click to create a new platform configuration. Upon clicking, the New Platform Configuration dialog appears. Enter a name for the new platform configuration, select the hardware and click OK to create a new platform configuration.
Rename	Click to rename the selected platform configuration. Upon clicking, the Rename Platform Configuration dialog appears. Enter a new name for the selected platform configuration and click OK to rename an existing platform configuration.
Edit	Click to edit the trace platform configuration. Upon clicking, the Software Analysis Configuration dialog appears. Enable trace on individual AIOP and MC cores, select a trace scenario and configure trace location and debug the target to collect hardware trace.
Enable Point to Point Profiler	Check to enable performance gathering.
Enable Code Coverage	Check to enable trace collection.
Advanced settings	Specifies the advanced settings to configure the communication settings, communication port number, and processing scenario for the simulator, CodeWarrior, or compatibility mode.

3.1.5 Source

Use this tab to specify the location of source files used when debugging a C or C++ application. By default, this information is taken from the build path of your project.

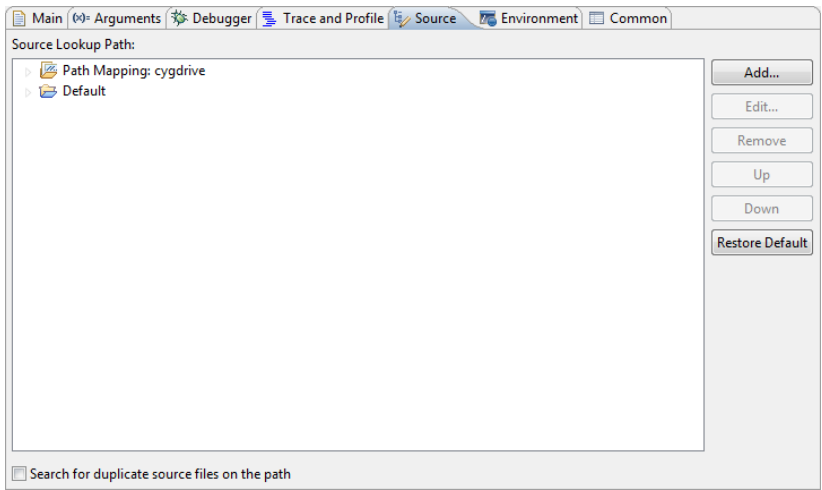


Figure 3-12. Debug Configurations - Source tab

The table below lists the various options available on the **Source** tab page.

Table 3-13. Source tab options

Option	Description
Source Lookup Path	Lists the source paths used to load an image after connecting the debugger to the target.
Add	Click to add new source containers to the Source Lookup Path search list.
Edit	Click to modify the content of the selected source container.
Remove	Click to remove selected items from the Source Lookup Path list.
Up	Click to move selected items up the Source Lookup Path list.
Down	Click to move selected items down the Source Lookup Path list.
Restore Default	Click to restore the default source search list.
Search for duplicate source files on the path	Select to search for files with the same name on a selected path.

3.1.6 Environment

Use this tab to specify the environment variables and values to use when an application runs.

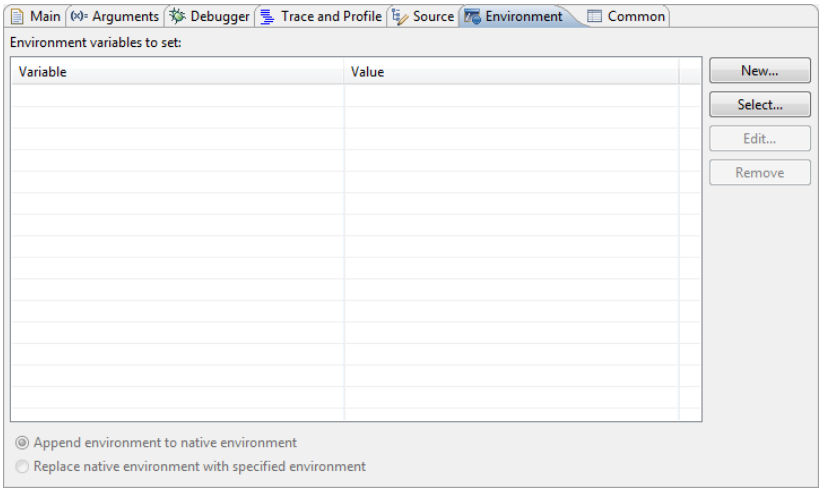


Figure 3-13. Debug Configurations - Environment tab

The table below lists the various options available on the **Environment** tab page.

Table 3-14. Environment tab options

Option	Description
Environment variables to set	Lists the environment variable name and its value.
New	Click to create a new environment variable.
Select	Click to select an existing environment variable.
Edit	Click to modify the name and value of a selected environment variable.
Remove	Click to remove selected environment variables from the list.
Append environment to native environment	Select to append the listed environment variables to the current native environment.
Replace native environment with specified environment	Select to replace the current native environment with the specified environment set.

3.1.7 Common

Use this tab to specify the location to store your run configuration, standard input and output, and background launch options.

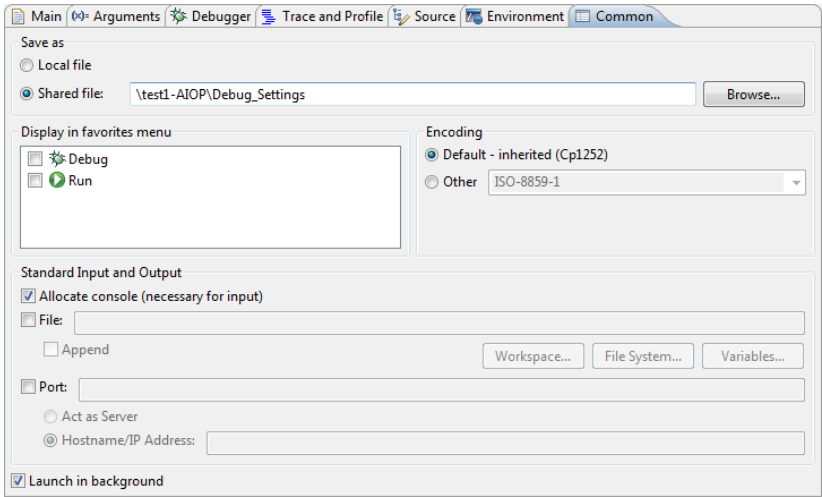


Figure 3-14. Debug Configurations - Common tab

The table below lists the various options available on the **Common** tab page.

Table 3-15. Common tab options

Option	Description
Local file	Select to save the launch configuration locally.
Shared file	Select to specify the path of, or browse to, a workspace to store the launch configuration file, and be able to commit it to a repository.
Display in favorites menu	Select to add the configuration name to Run or Debug menus for easy selection.
Encoding	Select an encoding scheme to use for console output.
Allocate console (necessary for input)	Select to assign a console view to receive the output.
File	Specify the file name to save output.
Workspace	Specifies the path of, or browse to, a workspace to store the output file.
File System	Specifies the path of, or browse to, a file system directory to store the output file.
Variables	Select variables by name to include in the output file.
Append	Select to append output. Clear to recreate file each time.
Port	Select to redirect standard output (<code>stdout</code> , <code>stderr</code>) of a process being debugged to a user specified socket. Note: You can also use the <code>redirect</code> command in debugger shell to redirect standard output streams to a socket.
Act as Server	Select to redirect the output from the current process to a local server socket bound to the specified port.
Hostname/IP Address	Select to redirect the output from the current 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
Launch in background	Select to launch configuration in background mode.

3.2 Customizing debug configurations

When you use the CodeWarrior wizard to create a new project, the wizard sets the project's launch configurations to default values. You can change these default values based on your program's requirements.

To modify the debug configurations:

1. Start the CodeWarrior IDE.
2. From the IDE menu bar, select **Run > Debug Configurations**.

The **Debug Configurations** dialog appears.

3. Expand the **CodeWarrior** debug configuration.
4. Select an option from the **Debug session type** group to specify the debug configuration that you want to modify.

The figure below shows the **CodeWarrior Debug Configuration** dialog with the settings for the debug session type you selected.

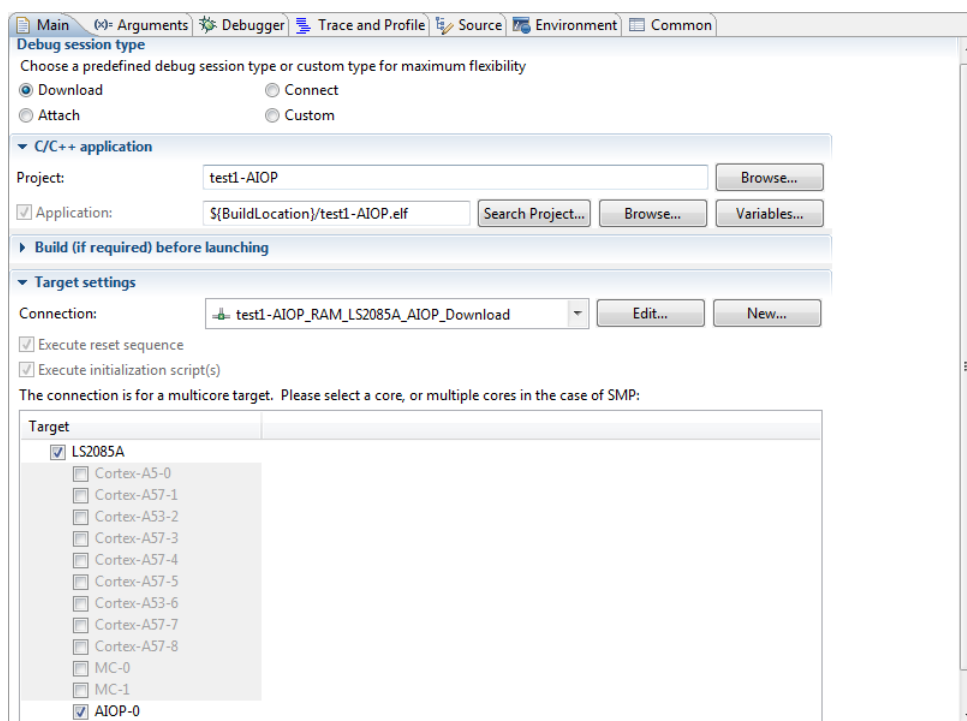


Figure 3-15. CodeWarrior Debug Configurations - Main tab

5. Change the settings on the debug configuration page as per your requirements. See [Using CodeWarrior debug configuration tabs](#) section for details on the various settings of this page.

6. Click **Apply** to save the new settings.

NOTE

Click **Debug** to start a new debugging session, or click **Close** to save your changes and close the **Debug Configurations** dialog.

3.3 Reverting debug configuration settings

As you modify a debug configuration's settings, you create pending, or unsaved, changes to that launch configuration. To save the pending changes, you must click the **Apply** button of the **Debug Configurations** dialog, or click the **Close** button and then the **Yes** button.

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

Chapter 4

Build Properties

A *build configuration* is a named collection of build tools options. The set of options in a given build configuration causes the build tools to generate a final binary with specific characteristics. For example, the binary produced by a "Debug" build configuration might contain symbolic debugging information and have no optimizations, while the binary product by a "Release" build configuration might contain no symbolics and be highly optimized.

NOTE

The settings of the CodeWarrior IDE's build and launch configuration correspond to an object called a target made by the classic CodeWarrior IDE.

This chapter explains:

- [Changing build properties](#)
- [Restoring build properties](#)
- [Build properties for APP](#)

4.1 Changing build properties

You can modify the build properties of a project to better suit your needs. To change build properties:

1. Start the CodeWarrior IDE.
2. In the **CodeWarrior Projects** view, select the project for which you want to modify the build properties.
3. Select **Project > Properties** from the IDE menu bar.

The **Properties for <project>** dialog appears. The left panel of this dialog shows the build properties that apply to the current project.

4. From the left panel, select **C/C++ Build > Settings**.
5. Use the **Configuration** drop-down list in the right panel to specify the launch configuration for which you want to modify the build properties.
6. Click the **Tool Settings** tab. The corresponding page appears.
7. From the list of tools on the **Tool Settings** page, select the tool for which you want to modify the properties.
8. Change the settings that appear in the page.
9. Click **Apply**.

The IDE saves your new settings. You can select other tool pages and modify their settings.

10. Click **OK**.

The IDE saves your changes and closes the **Properties for <project>** dialog.

4.2 Restoring build properties

If you modify a build configuration of a project and might choose to restore the build properties in order to have a factory-default configuration, or to revert to a last-known working build configuration. To undo your modifications to build properties, click the **Restore Defaults** button at the bottom of the **Properties** window.

This changes the values of the options to the absolute default of the toolchain. By default, the toolchain options are blank.

For example, when a bareboard project is created the **Linker** panel has some values set, which are specific to the project. By selecting the **Restore Defaults** button the default values of settings will return to blank state of the toolchain.

4.3 Build properties for APP

The build tools used in a project depend upon the processor and the build toolchain that is selected while creating a project. The table below lists the CodeWarrior build tool settings specific to developing software for APP.

NOTE

For more details on CodeWarrior build tools, see *CodeWarrior Development Studio for Advanced Packet Processing Build*

Tools Reference Manual available in the `<CWInstallDir>\LS\Help`
`\PDF\` folder, where `<CWInstallDir>` is the path where you have
 installed your CodeWarrior software.

Table 4-1. CodeWarrior build tool settings for QorIQ LS Series processor family

Build Tool	Build Properties Panels
CPU	
Debugging	
Messages	
Linker	Input
	Link Order
	General
	Output
Compiler	Preprocessor
	Input
	Warnings
	Optimization
	Processor
	C/C++ Language
Assembler	Input
	General
Disassembler	Disassembler Settings
Preprocessor	Preprocessor Settings

4.3.1 CPU

Use the CPU panel to specify the APP processor family for the project. The properties specified on this page are also used by the build tools (compiler, linker, and assembler).

The table below lists and describes the various options available on the **CPU** panel.

Table 4-2. CodeWarrior build tool settings - CPU options

Option	Explanation
Processor	Generates and links object code for a specific processor. This setting is equivalent to specifying the <code>-proc [essor]</code> keyword command-line option.
Floating Point	Controls floating-point code generation. This setting is equivalent to specifying the <code>-fp</code> keyword command-line option.

Table continues on the next page...

Table 4-2. CodeWarrior build tool settings - CPU options (continued)

Option	Explanation
Byte Ordering	Generates object code and links an executable image to use the specified data format. This setting is equivalent to specifying the <code>-big</code> or <code>-little</code> command-line options.
Code Model	Specifies the addressing mode that the linker uses when resolving references. This setting is equivalent to specifying the <code>-model</code> <i>keyword</i> command-line option.
ABI	Chooses which ABI (Application Binary Interface) to conform to. This setting is equivalent to specifying the <code>-abi</code> <i>keyword</i> command-line option.
Tune Relocations	Ensures that references made by the linker conform to the EABI (Embedded Application Binary Interface) or position-independent ABI (Application Binary Interface). Use this option only when you select EABI or SDA PIC/PID from the ABI drop-down list, to ensure that references in the executable image conform to these ABIs. To conform to both of these ABIs, the linker will modify relocations that do not reach the desired executable code. The linker first converts near branch instructions to far branch instructions. Then it will convert absolute branches to PC-relative branches. For branches that cannot be converted to far or PC-relative addressing, the linker will generate branch islands. To conform to the SDA PIC/PID ABI , the linker will generate the appropriate style of addressing. This setting is equivalent to specifying the <code>-tune_relocations</code> command-line option.
Compress for AIOP VLE	Specifies compression of the VLE (Variable Length Encoding) code by shortening the gaps between the functions. NOTE: For processors that do not have the VLE capability, this option is disabled and cannot be selected.
Small Data	Limits the size of the largest objects in the small data section. This setting is equivalent to specifying the <code>-sdata[threshold] size</code> command-line option. The <i>size</i> value specifies the maximum size, in bytes, of all objects in the small data section (<code>.sdata</code>). The default value for <i>size</i> is 8. The linker places objects that are greater than this size in the data section (<code>.data</code>) instead.
Small Data2	Limits the size of the largest objects in the small constant data section. This setting is equivalent to specifying the <code>-sdata2[threshold] size</code> command-line option. The <i>size</i> value specifies the maximum size, in bytes, of all objects in the small constant data section (<code>.sdata2</code>). The default value for <i>size</i> is 8. The linker places constant objects that are greater than this size in the constant data section (<code>.rodata</code>) instead.

4.3.2 Debugging

Use the Debugging panel to specify the global debugging options for the project.

The table below lists and describes the various options available on the **Debugging** panel.

Table 4-3. CodeWarrior build tool settings - Debugging options

Option	Explanation
Generate DWARF Information	Generates DWARF 2.x conforming debugging information. This setting is equivalent to specifying the <code>-sym dwarf-2</code> command-line option.
Store Full Paths To Source Files	Stores absolute paths of the source files instead of relative paths. This setting is equivalent to specifying the <code>-sym full[path]</code> command-line option.

4.3.3 Messages

Use the Messages panel to specify the error and warning message options for the project.

[Table 4-4](#) lists and describes the various options available on the **Messages** panel.

Table 4-4. CodeWarrior build tool settings - Messages options

Option	Explanation
Message Style	Controls the style used to show error and warning messages. This setting is equivalent to specifying the <code>-msgstyle keyword</code> command-line option.
Maximum Number of Errors	Specifies the maximum number of errors messages to show. This setting is equivalent to specifying the <code>-maxerrors number</code> command-line option.
Maximum Number of Warnings	Specifies the maximum number of warning messages to show. This setting is equivalent to specifying the <code>-maxwarnings number</code> command-line option.

4.3.4 Linker

Use the Linker panel to specify the CodeWarrior linker options that are specific to APP software development.

NOTE

The list of tools presented on the **Tool Settings** page can differ, based upon the toolchain used by the project.

The table below lists and describes the various options available on the **Linker** panel.

Table 4-5. CodeWarrior build tool settings - Linker options

Option	Explanation
Command	Specifies the location of the linker executable file.
All options	Specifies the actual command line, the linker will be called with.
Command line pattern	Specifies the expert settings command line parameters.

4.3.4.1 Input

Use the Input panel to specify the path to the linker command file and the libraries.

The table below lists and describes the various options available on the **Input** panel.

Table 4-6. CodeWarrior build tool settings - Input options

Option	Explanation
No Standard Library	Uses standard system library access paths as specified by the environment variable %MWLibraries% to add system libraries as specified by the environment variable %MWLibraryFiles% at the end of link order. This setting is equivalent to specifying the <code>-nostdlib</code> command-line option.
Link Command File (.lcf)	Specifies the path of the linker-command file that the linker reads to determine how to build the output file. Alternatively, click Browse, then use the resulting dialog box to specify the linker command file. This setting is equivalent to specifying the <code>-lcf filename</code> command-line option.
Code Address	Sets the run-time address of the executable code. This setting is equivalent to specifying the <code>-codeaddr addr</code> command-line option. The <code>addr</code> value is an address, in decimal or hexadecimal format. Hexadecimal values must begin with <code>0x</code> . The default is <code>65536</code> . This option is disabled and cannot be selected if you have specified the <code>.lcf</code> file in the Link Command File (.lcf) text box.
Data Address	Sets the loading address of the data. This setting is equivalent to specifying the <code>-dataaddr addr</code> command-line option. The <code>addr</code> value is an address, in decimal or hexadecimal format. Hexadecimal values must begin with <code>0x</code> . The default is the address after the code and large constant sections. This option is disabled and cannot be selected if you have specified the <code>.lcf</code> file in the Link Command File (.lcf) text box.
Small Data Address	Sets the loading address of small data. This setting is equivalent to specifying the <code>-sdataaddr addr</code> command-line option. The <code>addr</code> value is an address, in decimal or hexadecimal format. Hexadecimal values must begin with <code>0x</code> . The default is the address after the large data section. This option is disabled and cannot be selected if you have specified the <code>.lcf</code> file in the Link Command File (.lcf) text box.
Small Data 2 Address	Sets the loading address of small constant data. This setting is equivalent to specifying the <code>-sdata2addr addr</code> command-line option. The <code>addr</code> value is an address, in decimal or hexadecimal format. Hexadecimal values must begin with <code>0x</code> . The default is the address after the small data section. This option is disabled and cannot be selected if you have specified the <code>.lcf</code> file in the Link Command File (.lcf) text box.
Entry Point	Specifies the main entry point for the executable image. This setting is equivalent to specifying the <code>-m[ain] symbol</code> command-line option. The maximum length of <code>symbol</code> is 63 characters. The default is <code>__start</code> .






Table continues on the next page...

Table 4-6. CodeWarrior build tool settings - Input options (continued)

Option	Explanation
Library Search Paths	Use this panel to specify multiple paths that the linker searches for libraries. The linker searches the paths in the order shown in this list. Table 4-7 lists and describes the toolbar buttons that help work with the library search paths.
Library Files	Lists paths to libraries that the linker uses. The linker uses the libraries in the order shown in this list. Table 4-7 lists and describes the toolbar buttons that help work with the library file search paths.

The table below lists and describes the toolbar buttons that help work with the library search paths.

Table 4-7. CodeWarrior build tool settings - Input toolbar buttons

Button	Tooltip	Description
	Add	Click to open the Add file path or the Add directory path dialog box and create a file or directory path.
	Delete	Click to delete the selected file or directory. To confirm deletion, click Yes in the Confirm Delete dialog box.
	Edit	Click to open the Edit file path or Edit directory path dialog box and update the selected file or directory.
	Move up	Click to move the selected file search path one position higher in the list.
	Move down	Click to move the selected file search path one position lower in the list.

4.3.4.2 Link Order

Use the Link Order panel to control the link input order.

The table below lists and describes the various options available on the **Link Order** panel.

Table 4-8. CodeWarrior build tool settings - Link Order options

Option	Explanation
Customize linker input order	Allows to change the default link input order. Selecting this option enables the Link Order panel, allowing you to change the default link input order by using the Move Up and Move Down buttons on the Link Order panel toolbar.
Link Order	Shows the default link input order that you can change by selecting a link input and clicking the Move Up or Move Down button on the Link Order panel toolbar.

4.3.4.3 General

Use the General panel to specify the linker performance and optimization parameters.

The table below lists and describes the various options available on the **General** panel.

Table 4-9. CodeWarrior build tool settings - General options

Option	Explanation
Link Mode	Controls the performance of the linker. The default options are: <ul style="list-style-type: none"> • Normal - Uses little memory but may take more processing time. • Use Less RAM - Uses medium amount of memory for medium processing time. • Use More RAM - Uses lots of memory to improve processing time. This setting is equivalent to specifying the <code>-linkmode keyword</code> command-line option.
Code Merging	Code merging reduces the size of object code by removing identical functions. The default options are: <ul style="list-style-type: none"> • All Functions - Controls code merging for all identical functions. • Safe Functions - Controls code merging for weak functions. This setting is equivalent to specifying the <code>-code_merging all safe</code> command-line option.
Aggressive Merging	The code merging optimization will not remove an identical copy of a function if your program refers to its address. In this case, the compiler keeps this copied function but replaces its executable code with a branch instruction to the original function. To ignore references to function addresses, use aggressive code merging. This setting is equivalent to specifying the <code>-code_merging all,aggressive</code> or <code>-code_merging safe,aggressive</code> command-line options.
Merges FP Constants	Compiler pools strings of a file, when the option is checked. Clear this option to keep individual the strings of each file. (This permits deadstripping of unused strings.) This setting is equivalent to specifying the <code>#pragma fp_constants merge pragma</code> .
Other Flags	Specify linker flags.

4.3.4.4 Output

Use the Output panel to specify the configuration of your final output file.

The table below lists and describes the various options available on the **Output** panel.

Table 4-10. CodeWarrior build tool settings - Output options

Option	Explanation
Output Type	Specifies the generated output type. The default options are: <ul style="list-style-type: none"> • Application • Static Library • Partial Link This setting is equivalent to specifying the <code>-application, -library, -partial</code> command-line options.

Table continues on the next page...

Table 4-10. CodeWarrior build tool settings - Output options (continued)

Option	Explanation
Optimize Partial Link	Specifies the use of a linker command file, create tables for C++ static constructors, C++ static destructors, and C++ exceptions. This option also configures the linker to build an executable image, even if some symbols cannot be resolved. NOTE: Select Partial Link from the Output Type list box, to enable this option. This setting is equivalent to specifying the <code>-opt_partial</code> command-line option.
Deadstrip Unused Symbols	Removes unreferenced objects on a partially linked image. NOTE: Select Partial Link from the Output Type list box, to enable this option. This setting is equivalent to specifying the <code>-strip_partial</code> command-line option.
Require Resolved Symbols	Finishes a partial link operation and issues error messages for unresolved symbols. NOTE: Select Partial Link from the Output Type list box, to enable this option. This setting is equivalent to specifying the <code>-resolved_partial</code> command-line option.
Heap Size (k)	Sets the run-time size of the heap, in kilobytes. This setting is equivalent to specifying the <code>-heapsize size</code> command-line option.
Stack Size (k)	Sets the run-time size of the stack, in kilobytes. This setting is equivalent to specifying the <code>-stacksize size</code> command-line option.
Interpreter	Specifies the interpreter file used by the linker.
Generate Link Map	Generates a text file that describes the contents of the linker's output file. This setting is equivalent to specifying the <code>-map [filename]</code> command-line option.
List Closure	Controls the appearance of symbol closures in the linker map file. This setting is equivalent to specifying the <code>-listclosure</code> command-line option.
List Unused Objects	Controls the appearance of a list of unused symbols in the linker map file. This setting is equivalent to specifying the <code>-mapunused</code> command-line option.
List DWARF Objects	Controls the appearance of DWARF debugging information in the linker map file. This setting is equivalent to specifying the <code>-listdwarf</code> command-line option.
List Estimated Stack Usage	Controls the generation of stack usage information in the linker map file.
Root Function	This is the function at which the analysis is performed. Normally this would be <code>main()</code> , but in the case of AIOP, it could be any entry point function. By default, the <code>root_function</code> is set to <code>all</code> for iterating through all the entry point functions (identified by <code>__declspec(entry_point)</code>) and performing the analysis for each entry point. This setting is equivalent to specifying the <code>-estimate_stack_usage</code> command-line option.
Workspace Size	Allows the user to specify the workspace size in bytes. The default is 2048 (2Kb). This linker option is optional. Using this option affects the stack usage calculations.
Recursion Depth	Controls level of recursion reporting in the map file report. This option is only activated if List Estimated Stack Usage option is selected. By default, the value is set to 1 if not specified. This setting is equivalent to specifying the <code>-recursion_depth</code> command-line option.
Sort Root Function	Sorts root function reports and outputs the results of the Stack Estimator in descending sorted order according to the cumulative stack usage of the root function. This setting is equivalent to specifying <code>-root_sort</code> command-line option.
Generate Binary File	Controls generation of the binary files. The default options are: <ul style="list-style-type: none"> • None - Generates no binary file even if S-record generation is on. This is the default option. • One - Generates a single binary file with all the loadable code and data, even if S-record generation is off. • Multiple - Generates separate binary files for each MEMORY directive, even if S-record generation is off. This setting is equivalent to specifying the <code>-genbinary keyword</code> command-line option.

Table continues on the next page...

Table 4-10. CodeWarrior build tool settings - Output options (continued)

Option	Explanation
Generate S-Record File	Generates an S-record file. This setting is equivalent to specifying the <code>-srec</code> command-line option.
Sort S-Record	Sorts the records, in ascending order, in an S-record file. NOTE: Select Generate S-Record File , to enable this option. This setting is equivalent to specifying the <code>-sortsrec</code> command-line option.
Max S-Record Length	Specifies the length of S-records. You can select a value from 8 to 255. The default is 26. NOTE: Select Generate S-Record File , to enable this option. This setting is equivalent to specifying the <code>-sreclength</code> command-line option.
EOL Character	Specifies the end-of-line style to use in an S-record file. The default options are: <ul style="list-style-type: none"> • Mac - Use Mac OS®-style end-of-line format. • DOS - Use Microsoft® Windows®-style end-of-line format. This is the default choice. • UNIX - Use a UNIX-style end-of-line format. NOTE: Select Generate S-Record File , to enable this option. This setting is equivalent to specifying the <code>-sreceol keyword</code> command-line option.
Generate Warning Messages	Turns on most warning messages issued by the build tools. This setting is equivalent to specifying the <code>-w on</code> command-line option.
Heap Address	Sets the run-time address of the heap. The specified address must be in decimal or hexadecimal format. Hexadecimal values must begin with <code>0x</code> . The default is <code>stack_address - (heap_size + stack_size)</code> where <code>stack_address</code> is the address of the stack, <code>heap_size</code> is the size of the heap, and <code>stack_size</code> is the size of the stack. This setting is equivalent to specifying the <code>-heapaddr address</code> command-line option.
Stack Address	Sets the run-time address of the stack. The specified address must be in decimal or hexadecimal format. Hexadecimal values must begin with <code>0x</code> . This setting is equivalent to specifying the <code>-stackaddr address</code> command-line option.
Generate ROM Image	Enables generation of a program image that may be stored in and started from ROM.
ROM Image Address	Generates a ROM image and specifies the image's starting address at run time. NOTE: Select Generate ROM Image , to enable this option. This setting is equivalent to specifying the <code>-romaddr address</code> command-line option.
RAM Buffer Address of ROM Image	Specifies a run-time address in which to store the executable image in RAM so that it may be transferred to flash memory. NOTE: Select Generate ROM Image , to enable this option. This option specifies information for a legacy flashing tool. This tool required that the executable image must first be loaded to an area in RAM before being transferred to ROM. NOTE: Do not use this option if your flash memory tool does not follow this behavior. This setting is equivalent to specifying the <code>-rombuffer address</code> command-line option.

4.3.5 Compiler

Use the Compiler panel to specify the compiler options that are specific to APP software development.

The table below lists and describes the various options available on the **Compiler** panel.

Table 4-11. CodeWarrior build tool settings - Compiler options

Option	Explanation
Command	Specifies the location of the compiler executable file that will be used to build the project.
All Options	the actual command line the compiler will be called with.
Command line pattern	Shows the expert settings command line parameters.

4.3.5.1 Preprocessor

Use the Preprocessor panel to specify the preprocessor behavior by providing details of the file, whose contents can be used as prefix to all source files.

The table below lists and describes the various options available on the **Preprocessor** panel.

Table 4-12. CodeWarrior build tool settings - Preprocessor options

Option	Explanation
Source encoding	Specifies the default source encoding used by the compiler. The compiler automatically detects UTF-8 (Unicode Transformation Format) header or UCS-2/UCS-4 (Uniform Communications Standard) encodings regardless of setting. The default setting is ascii. This setting is equivalent to specifying the <code>-enc[oding] keyword</code> command-line option.
Prefix Files	Adds contents of a text file or precompiled header as a prefix to all source files. This setting is equivalent to specifying the <code>-prefix file</code> command-line option.
Defined Macros (-D)	Defines a specified symbol name. This setting is equivalent to specifying the <code>-Dname</code> command-line option, where <i>name</i> is the symbol name to define.
Undefined Macros (-U)	Undefines the specified symbol name. This setting is equivalent to specifying the <code>-U name</code> command-line option, where <i>name</i> is the symbol name to undefine.

4.3.5.2 Input

Use the Input panel to specify the path and search order of the `#include` files.






The table below lists and describes the various options available on the **Input** panel.

Table 4-13. CodeWarrior build tool settings - Input options

Option	Explanation
Compile Only, Do Not Link	Instructs the compiler to compile but not invoke the linker to link the object code. This setting is equivalent to specifying the <code>-c</code> command-line option.
Do not use MWCIncludes variable	Restricts usage of standard system include paths as specified by the environment variable <code>%MWCIncludes%</code> . This setting is equivalent to specifying the <code>-nostdinc</code> command-line option.
Always Search User Paths	Performs a search of both the user and system paths, treating <code>#include</code> statements of the form <code>#include <xyz></code> the same as the form <code>#include "xyz"</code> . This setting is equivalent to specifying the <code>-nosyspath</code> command-line option.
User Path (-i)	Use this panel to specify multiple user paths and the order in which to search those paths. Table 4-14 lists and describes the toolbar buttons that help work with the file search paths. This setting is equivalent to specifying the <code>-i</code> command-line option.
User Recursive Path (-ir)	Appends a recursive access path to the current User Path list. Table 4-14 lists and describes the toolbar buttons that help work with the file search paths. This setting is equivalent to specifying the <code>-ir path</code> command-line option.
System Path (-I -l)	<p>Changes the build target's search order of access paths to start with the system paths list. Table 4-14 lists and describes the toolbar buttons that help work with the file search paths.</p> <ul style="list-style-type: none"> The compiler can search <code>#include</code> files in several different ways. Use this panel to set the search order as follows: For include statements of the form <code>#include "xyz"</code>, the compiler first searches user paths, then the system paths For include statements of the form <code>#include <xyz></code>, the compiler searches only system paths <p>This setting is equivalent to specifying the <code>-I- -I path</code> command-line option.</p>
System Recursive Path (-I- -ir)	Appends a recursive access path to the current System Path list. Table 4-14 lists and describes the toolbar buttons that help work with the file search paths. This setting is equivalent to specifying the <code>-I- -ir</code> command-line option.
Disable CW Extensions	Controls deadstripping files. Not all third-party linkers require checking this option.

The table below lists and describes the toolbar buttons that help work with the **Input** panel.

Table 4-14. CodeWarrior build tool settings - Input toolbar buttons

Button	Tooltip	Description
	Add	Click to open the Add file path or the Add directory path dialog box and create a file or directory path.
	Delete	Click to delete the selected file or directory. To confirm deletion, click Yes in the Confirm Delete dialog box.
	Edit	Click to open the Edit file path or Edit directory path dialog box and update the selected file or directory.
	Move up	Click to move the selected file search path one position higher in the list.
	Move down	Click to move the selected file search path one position lower in the list.

4.3.5.3 Warnings

Use the Warnings panel to control how the compiler reports the error and warning messages.

The table below lists and describes the various options available on the **Warnings** panel.

Table 4-15. CodeWarrior build tool settings - Warnings options

Option	Explanation
Treat All Warnings As Errors	Select to make all warnings into hard errors. Source code which triggers warnings will be rejected.
Illegal Pragmas	Select to issue a warning message if the compiler encounters an unrecognized pragma. This setting is equivalent to specifying the <code>pragma warn_illpragma</code> pragma and the <code>-warnings illpragmas</code> command-line option.
Possible Errors	Select to issue warning messages for common, usually-unintended logical errors: in conditional statements, using the assignment (<code>=</code>) operator instead of the equality comparison (<code>==</code>) operator, in expression statements, using the <code>==</code> operator instead of the <code>=</code> operator, placing a semicolon (<code>;</code>) immediately after a <code>do</code> , <code>while</code> , <code>if</code> , or <code>for</code> statement. This setting is equivalent to specifying the <code>warn_possunwant</code> pragma and the <code>-warnings possible</code> command-line option.
Extended Error Checking	Select to issue warning messages for common programming errors: mis-matched return type in a function's definition and the return statement in the function's body, mismatched assignments to variables of enumerated types. This setting is equivalent to specifying the <code>extended_errorcheck</code> pragma and the <code>-warnings extended</code> command-line option.
Hidden virtual functions	Select to issue warning messages if you declare a non-virtual member function that prevents a virtual function, that was defined in a superclass, from being called. This setting is equivalent to specifying the <code>warn_hidevirtual</code> pragma and the <code>-warnings hidevirtual</code> command-line option.
Implicit Arithmetic Conversions	Select to issue warning messages when the compiler applies implicit conversions that may not give results you intend: assignments where the destination is not large enough to hold the result of the conversion, a signed value converted to an unsigned value, an integer or floating-point value is converted to a floating-point or integer value, respectively. This setting is equivalent to specifying the <code>warn_implicitconv</code> pragma and the <code>-warnings implicitconv</code> command-line option.
Implicit Integer To Float Conversions	Select to issue warning messages for implicit conversions from integer to floating-point values. This setting is equivalent to specifying the <code>warn_impl_i2f_conv</code> pragma and the <code>-warnings impl_int2float</code> command-line option.
Implicit Float To Integer Conversions	Select to issue warning messages for implicit conversions from floating point values to integer values. This setting is equivalent to specifying the <code>warn_impl_f2i_conv</code> pragma and the <code>-warnings impl_float2int</code> command-line option.
Implicit Signed/Unsigned Conversions	Select to issue warning messages for implicit conversions from a signed or unsigned integer value to an unsigned or signed value, respectively. This setting is equivalent to specifying the <code>warn_impl_s2u_conv</code> pragma and the <code>-warnings signedunsigned</code> command-line option.
Pointer/Integral Conversions	Select to issue warning messages for implicit conversions from pointer values to integer values and from integer values to pointer values. This setting is equivalent to specifying the <code>warn_any_ptr_int_conv</code> and <code>warn_ptr_int_conv</code> pragmas and the <code>-warnings ptrintconv, anyptrintconv</code> command-line option.

Table continues on the next page...

Table 4-15. CodeWarrior build tool settings - Warnings options (continued)

Option	Explanation
Unused Arguments	Select to issue warning messages for function arguments that are not referred to in a function. This setting is equivalent to specifying the <code>warn_unusedarg</code> pragma and the <code>-warnings unusedarg</code> command-line option.
Unused Variables	Select to issue warning messages for local variables that are not referred to in a function. This setting is equivalent to specifying the <code>warn_unusedvar</code> pragma and the <code>-warnings unusedvar</code> command-line option.
Missing 'return' Statement	Select to issue warning messages, if a function that is defined to return a value has no return statement. This setting is equivalent to specifying the <code>warn_missingreturn</code> pragma and the <code>-warnings missingreturn</code> command-line option.
Expression Has No Side Effect	Select to issue warning messages if a statement does not change the program's state. This setting is equivalent to specifying the <code>warn_no_side_effect</code> pragma and the <code>-warnings unusedexpr</code> command-line option.
Extra Commas	Select to issue a warning messages if a list in an enumeration terminates with a comma. The compiler ignores terminating commas in enumerations when compiling source code that conforms to the ISO/IEC 9899-1999 ("C99") standard. This setting is equivalent to specifying the <code>warn_extracomma</code> pragma and the <code>-warnings extracomma</code> command-line option.
Empty Declarations	Select to issue warning messages if a declaration has no variable name. This setting is equivalent to specifying the <code>warn_emptydecl</code> pragma and the <code>-warnings emptydecl</code> command-line option.
Inconsistent 'class' / 'struct' Usage	Select to issue warning messages if the class and struct keywords are used interchangeably in the definition and declaration of the same identifier in C++ source code. This setting is equivalent to specifying the <code>warn_structclass</code> pragma and the <code>-warnings structclass</code> command-line option.
Include File Capitalization	Select to issue warning messages if the name of the file specified in a <code>#include</code> file directive uses different letter case from a file on disk. This setting is equivalent to specifying the <code>warn_filenameecaps</code> pragma and the <code>-warnings fileecaps</code> command-line option.
Check System Includes	Select to issue warning messages if the name of the file specified in a <code>#include</code> file directive uses different letter case from a file on disk. This setting is equivalent to specifying the <code>warn_filenameecaps_system</code> pragma and the <code>-warnings sysfilecaps</code> command-line option.
Pad Bytes Added	Select to issue warning messages when the compiler adjusts the alignment of components in a data structure. This setting is equivalent to specifying the <code>warn_padding</code> pragma and the <code>-warnings padding</code> command-line option.
Undefined Macro in #if	Select to issue warning messages if an undefined macro appears in <code>#if</code> and <code>#elif</code> directives. This setting is equivalent to specifying the <code>warn_undefmacro</code> pragma and the <code>-warnings undefmacro</code> command-line option.
Non-Inlined Functions	Select to issue warning messages if a call to a function defined with the <code>inline</code> , <code>__inline__</code> , or <code>__inline</code> keywords could not be replaced with the function body. This setting is equivalent to specifying the <code>warn_notinlined</code> pragma and the <code>-warnings notinlined</code> command-line option.

4.3.5.4 Optimization

Use the Optimization panel to control the code optimization settings.

The table below lists and describes the various options available on the **Optimization** panel.

Table 4-16. CodeWarrior build tool settings - Optimization options

Option	Explanation
Optimization Level	Specifies code optimization options to apply to object code. This setting is equivalent to specifying the <code>-opt keyword</code> command-line option.
Speed vs. Size	Specifies code optimization for speed or size. This setting is equivalent to specifying the <code>optimize_for_size on</code> or <code>optimize_for_size off</code> pragmas and <code>-opt speed</code> or <code>-opt size</code> command-line option.
Inlining	Specifies inline options. Default settings are: <ul style="list-style-type: none"> Smart - The compiler considers the functions declared with <code>inline</code>. Auto Inline - The compiler also inlines C++ functions declared <code>inline</code> and member functions defined within a class declaration. This setting is equivalent to specifying the <code>-inline, -inline auto</code> command-line option.
Bottom-up Inlining	Select to instruct the compiler to inline functions from the last function called to the first function in a chain of function calls. This setting is equivalent to specifying the <code>inline_bottom_up</code> pragma and <code>-inline bottomup</code> command-line option.

4.3.5.5 Processor

Use the Processor panel to control the processor-dependent code-generation settings.

The table below lists and describes the various options available on the **Processor** panel.

Table 4-17. CodeWarrior build tool settings - Processor options

Option	Explanation
Struct Alignment	Specifies structure and array alignment. The default options are: <ul style="list-style-type: none"> AIOP - Use conventional Power Architecture alignment. This choice is the default. 68K - Use conventional Mac OS® 68K alignment. 68K 4-Byte - Use Mac OS® 68K 4-byte alignment. This setting is equivalent to specifying the <code>-align keyword</code> command-line option.
Function Alignment	Specifies alignment of functions in executable code. The default alignment is 4. However, at an optimization level 4, the alignment changes to 16. If you are using <code>-func_align 4</code> (or none) and if you are compiling for VLE, then the linker will compress gaps between VLE functions: <ul style="list-style-type: none"> if those functions are not called by a Classic AIOP function the function has an alignment greater than 4. NOTE: Compression of the gaps will only happen on files compiled by the CodeWarrior compiler. This setting is equivalent to specifying the <code>-func_align</code> command-line option.
Relax HW IEEE	Controls the use of relaxed IEEE floating point operations. This setting is equivalent to specifying the <code>-relax_ieee</code> command-line option.
Use Fused Multi-Add/Sub	Controls the use of fused multiply-addition instructions. This setting is equivalent to specifying the <code>-maf on off</code> command-line option.

Table continues on the next page...

Table 4-17. CodeWarrior build tool settings - Processor options (continued)

Option	Explanation
Generate FSEL Instructions	Controls the use of FSEL instructions. NOTE: Do not turn on this option, if the processor of your target platform does not have hardware floating-point capabilities, that includes fsel. This option only has an effect if Relax HW IEEE option or <code>-relax_ieee</code> command-line option is also specified. The default is <code>off</code> . This setting is equivalent to specifying the <code>-gen_fsel</code> command-line option.
Assume Ordered Compares	Controls the assumption of no unordered values in comparisons. This setting is equivalent to specifying the <code>-ordered-fp-compares</code> , <code>-no-ordered-fp-compares</code> command-line options.
Vector Support	Specifies supported vector options. Default settings are: <ul style="list-style-type: none"> • SPE - Enables the SPE vector support. This option needs to be enabled when the floating point is set to SPFP or DPFP as both SPFP and DPFP require support from the SPE vector unit. If the option is not turned on, the compiler generates a warning and automatically enables the SPE vector generation. • AltiVec - Enables the AltiVec vector support and generate AltiVec vectors and related instructions. This setting is equivalent to specifying the <code>-spe_vector</code> and <code>-vector keyword</code> command-line options.
Make Strings ReadOnly	Places string constants in a read-only section. This setting is equivalent to specifying the <code>-readonlystrings</code> command-line option.
Merges Strings Constant	Merges the string constants. This setting is equivalent to specifying the <code>-flag no-pool_strings</code> command-line option.
Pool Data	Controls the grouping of similar-sized data objects. Use this option to reduce the size of executable object code in functions that refer to many object of the same size. These similar-sized objects do not need to be of the same type. The compiler only applies this option to a function if the function refers to at least 3 similar-sized objects. The objects must be global or static. At the beginning of the function, the compiler generates instructions to load the address of the first similar-sized object. The compiler then uses this address to generate 1 instruction for each subsequent reference to other similar-sized objects instead of the usual 2 instructions for loading an object using absolute addressing. This setting is equivalent to specifying the <code>pool_data</code> pragma and <code>-pool[data]</code> command-line option.
Use Common Section	Moves uninitialized data into a common section. The default is off. This setting is equivalent to specifying the <code>-common</code> command-line option.
Use LMW STMW	Controls the use of multiple load and store instructions for function prologues and epilogues. The default is off. NOTE: This option is only available for big-endian processors. This option is not available for big-endian e500v1 and e500v2 architectures when vector and double-precision floating-point instructions are used. This setting is equivalent to specifying the <code>-use_lmw_stmw</code> command-line option.
Inlined Assembler is Volatile	Controls whether or not inline assembly statements will be optimized. This setting is equivalent to specifying the <code>-volatileasm</code> , <code>-novolatileasm</code> command-line options.
Instruction Scheduling	Controls the rearrangement of instructions to reduce the effects of instruction latency. The default is off. This setting is equivalent to specifying the <code>-schedule</code> command-line option.
Peephole Optimization	Specifies peephole optimization. This setting is equivalent to specifying the <code>peephole</code> pragma and the <code>-opt peep[hole]</code> command-line option.
Profiler Information	Controls the appearance of calls to a profiler library at the entry and exit points of each function. The default is off. This setting is equivalent to specifying the <code>-profile</code> command-line option.
Generate ISEL Instructions	Controls the use of isel instructions. The default is off. NOTE: If the processor of your target platform does not implement the Freescale ISEL APU, this option appears disabled and cannot be selected. This setting is equivalent to specifying the <code>-use-isel</code> command-line option.

Table continues on the next page...

Table 4-17. CodeWarrior build tool settings - Processor options (continued)

Option	Explanation
Generate Code Coverage Files	Enables coverage to verify if the code meets a specified code coverage acceptance criteria. This setting is equivalent to specifying the <code>-fcoverage</code> command line option. Compiling a program file with this option generates a <code>.gcno</code> file. The file contains information about basic block, control flow and line of source that was compiled. The basic format of the file is: File: <uint32>MAGIC:<uint32>VERSION:<uint32>STAMP RECORD*
Translate Asm to VLE Asm	Controls VLE code generation for inline assembly statements. NOTE: If the processor of your target platform does not have the VLE capability, this option appears disabled and cannot be selected. This setting is equivalent to specifying the <code>-ppc_asm_to_vle</code> command-line option.
Generate AIOP code extensions	Controls code generation of AIOP extension instructions. The default is off. This setting is equivalent to specifying the <code>-aiop</code> command-line option.
Disable AIOP e_ldw/e_stdw code generation	Controls code generation of <code>e_ldw</code> and <code>e_stdw</code> instructions for AIOP. This setting is equivalent to specifying the <code>-nogen_ld_std</code> command-line option.
Enable user-defined performance markers	Controls performance marker functionality. Enable this option to generate performance marker symbols used by the Point to Point Profiler. Performance markers are specially names symbols in an ELF which the Point to Point Profiler uses to identify regions of code to profile.

4.3.5.6 C/C++ Language

Use the C/C++ Language panel to control compiler language features and some object code storage features for the current build target.

The table below lists and describes the various options available on the **C/C++ Language** panel.

Table 4-18. CodeWarrior build tool settings - C/C++ Language options

Option	Explanation
Force C++ Compilation	Translates all C source files as C++ source code. This setting is equivalent to specifying the <code>cplusplus</code> pragma and <code>-lang c++</code> command-line option.
ISO C++ Template Parser	Enforces the use of ISO/IEC 14882-1998 standard for C++ to translate templates, and more careful use of the <code>typename</code> and <code>template</code> keywords. The compiler also follows stricter rules for resolving names during declaration and instantiation. This setting is equivalent to specifying the <code>parse_func_tmpl</code> pragma and <code>-iso_templates</code> command-line option.
Use Instance Manager	Reduces compile time by generating any instance of a C++ template (or non-inlined inline) function only once. This setting is equivalent to specifying the <code>-instmgr</code> command-line option.
Enable C++ Exceptions	Generates executable code for C++ exceptions. Enable this option, if you use the <code>try</code> , <code>throw</code> , and <code>catch</code> statements specified in the ISO/IEC 14882-1998 C++ standard. Otherwise, disable this setting to generate smaller and faster code. This setting is equivalent to specifying the <code>-cpp_exceptions</code> command-line option.
Enable RTTI	Allows the use of the C++ run-time type information (RTTI) capabilities, including the <code>dynamic_cast</code> and <code>typeid</code> operators. This setting is equivalent to specifying the <code>-RTTI</code> command-line option.

Table continues on the next page...

Table 4-18. CodeWarrior build tool settings - C/C++ Language options (continued)

Option	Explanation
Enable C++ 'bool' type, 'true' and 'false' Constants	Instructs the C++ compiler to recognize the bool type and its true and false values specified in the ISO/IEC 14882-1998 C++ standard. This setting is equivalent to specifying the <code>-bool</code> command-line option.
Enable wchar_t Support	Instructs the C++ compiler to recognize the wchar_t data type specified in the ISO/IEC 14882-1998 C++ standard. This setting is equivalent to specifying the <code>-wchar_t</code> command-line option.
EC++ Compatibility Mode	Verifies C++ source code files for Embedded C++ source code. This setting is equivalent to specifying the <code>-dialect ec++</code> command-line option.
ANSI Strict	Recognizes source code that conforms to the ISO/IEC 9899-1990 standard for C. This setting is equivalent to specifying the <code>-ansi strict</code> command-line option.
ANSI Keywords Only	Generates an error message for all non-standard keywords. NOTE: Enable this setting only if the source code strictly adheres to the ISO standard. This setting is equivalent to specifying the <code>-stdkeywords</code> command-line option.
Expand Trigraphs	Specifies compiler to recognize trigraph sequences. clear this option to use many common characters, that look like trigraph sequences, without including escape characters. This setting is equivalent to specifying the <code>-trigraphs</code> command-line option.
Legacy for-scoping	Generates an error message when the compiler encounters a variable scope usage that the ISO/IEC 14882-1998 C++ standard disallows, but is allowed in the C++ language specified in <i>The Annotated C++ Reference Manual ("ARM")</i> . This setting is equivalent to specifying the <code>-for_scoping</code> command-line option.
Require Prototypes	Specifies compiler to enforce the requirement of function prototypes. NOTE: The compiler generates an error message, if you define a previously referenced function that does not have a prototype. The compiler generates a warning message, if you define the function before it is referenced but do not give it a prototype. This setting is equivalent to specifying the <code>-requireprotos</code> command-line option.
Enable C99 Extensions	Specifies compiler to recognize ISO/IEC 9899-1999 ("C99") language features. This setting is equivalent to specifying the <code>-dialect c99</code> command-line option.
Enable GCC Extensions	Specifies compiler to recognize language features of the GNU Compiler Collection (GCC) C compiler that are supported by CodeWarrior compilers. This setting is equivalent to specifying the <code>-gcc_extensions</code> command-line option.
Enum Always Int	Specifies compiler to use signed integers to represent enumerated constants. This setting is equivalent to specifying the <code>-enum</code> command-line option.
Use Unsigned Chars	Specifies compiler to treat char declarations as unsigned char declarations. This setting is equivalent to specifying the <code>-char unsigned</code> command-line option.
Pool Strings	Specifies compiler to collect all string constants into a single data section in the object code, it generates. This setting is equivalent to specifying the <code>-strings pool</code> command-line option.
Reuse	Specifies compiler to store only one copy of identical string literals. This setting is equivalent to specifying the <code>-string reuse</code> command-line option.
IPA	<p>Specifies the Interprocedural Analysis (IPA) policy. The default values are:</p> <ul style="list-style-type: none"> • Off - No interprocedural analysis, but still performs function-level optimization. Equivalent to the "no deferred inlining" compilation policy of older compilers. • File - Completely parse each translation unit before generating any code or data. Equivalent to the "deferred inlining" option of older compilers. Also performs an early dead code and dead data analysis in this mode. Objects with unreferenced internal linkages will be dead-stripped in the compiler rather than in the linker. <p>This setting is equivalent to specifying the <code>-ipa</code> command-line option.</p>
Other flags	Specify compiler flags.

4.3.6 Assembler

Use the Assembler panel to determine the format used for the assembly source files and the code generated by the assembler.

The table below lists and describes the various options available on the **Assembler** panel.

Table 4-19. CodeWarrior build tool settings - Assembler options

Option	Explanation
Command	Shows the location of the assembler executable file.
All Options	Shows the actual command line the assembler will be called with.
Command line pattern	Shows the expert settings command line parameters.

4.3.6.1 Input

Use the Input panel to specify the path and search order of the `#include` files.






The table below lists and describes the various options available on the **Input** panel.

Table 4-20. CodeWarrior build tool settings - Input options

Option	Description
Always Search user Paths	Performs a search of both the user and system paths, treating <code>#include</code> statements of the form <code>#include <xyz></code> , the same as the form <code>#include "xyz"</code> . This setting is equivalent to specifying the <code>-nosyspath</code> command-line option.
User Path (-i)	Use this panel to specify multiple user paths and the order in which to search those paths. Table 4-21 lists and describes the toolbar buttons that help work with the file search paths. This setting is equivalent to specifying the <code>-i</code> command-line option.
User Recursive Path (-ir)	Appends a recursive access path to the current User Path list. Table 4-21 lists and describes the toolbar buttons that help work with the file search paths. This setting is equivalent to specifying the <code>-ir path</code> command-line option.
System Path (-I -I)	Changes the build target's search order of access paths to start with the system paths list. Table 4-21 lists and describes the toolbar buttons that help work with the file search paths. This setting is equivalent to specifying the <code>-I -I path</code> command-line option.
System Recursive Path (-I- -ir)	Appends a recursive access path to the current System Path list. Table 4-21 lists and describes the toolbar buttons that help work with the file search paths. This setting is equivalent to specifying the <code>-I- -ir</code> command-line option.

The table below lists and describes the toolbar buttons that help work with the **Input** panel.

Table 4-21. CodeWarrior build tool settings - Input toolbar buttons

Button	Tooltip	Description
	Add	Click to open the Add file path or the Add directory path dialog box and create a file or directory path.
	Delete	Click to delete the selected file or directory. To confirm deletion, click Yes in the Confirm Delete dialog box.
	Edit	Click to open the Edit file path or Edit directory path dialog box and update the selected file or directory.
	Move up	Click to move the selected file search path one position higher in the list.
	Move down	Click to move the selected file search path one position lower in the list.

4.3.6.2 General

Use the General panel to specify the assembler options that are specific to APP software development.

The table below lists and describes the various options available on the **General** panel.

Table 4-22. CodeWarrior build tool settings - General options

Option	Explanation
Labels Must End With ':'	Specifies whether labels must end with a colon (:). Clear this option to omit the ending colon from label names that start in the first column. This setting is equivalent to specifying the <code>.option colon off on reset</code> assembler control option.
Directives Begin With '.'	Controls period usage for directives. Check this option to ensure that each directive must start with a period. This setting is equivalent to specifying the <code>.option period off on reset</code> assembler control option.
Case Sensitive Identifier	Specifies case sensitivity for identifiers. This setting is equivalent to specifying the <code>.option case off on reset</code> assembler control option.
Allow Space In Operand Field	Controls spaces in operand fields. Clear this option, if a space in an operand field starts with a comment. This setting is equivalent to specifying the <code>.option space off on reset</code> assembler control option.
GNU Compatible Syntax	CodeWarrior Assembler supports several GNU-format assembly language extensions. Check this option to control GNU's assembler format conflicts with that of the CodeWarrior assembler.
Generate Listing File	Controls generation of a listing file that includes files source, line numbers, relocation information, and macro expansions. Clear this option, if no listing file is specified.
Other Flags	Specify assembler flags.

4.3.7 Disassembler

Use the Disassembler panel to specify the command, options, and expert settings related to the disassembler.

The table below lists and describes the various options available on the **Disassembler** panel.

Table 4-23. CodeWarrior build tool settings - Disassembler options

Option	Explanation
Command	Shows the location of the disassembler executable file.
All options	Shows the actual command line the disassembler will be called with.
Command line pattern	Shows the expert settings command line parameters .

4.3.7.1 Disassembler Settings

Use the Disassembler Settings panel to specify the disassembler options that are specific to APP software development.

The table below lists and describes the various options available on the **Disassembler** panel.

Table 4-24. CodeWarrior build tool settings - Disassembler options

Option	Explanation
Show Headers	Controls display of object header information This setting is equivalent to specifying the <code>-show headers noheaders</code> command-line option.
Show Symbol and String Tables	Controls display of character string and symbol tables. This setting is equivalent to specifying the <code>-show tables notables</code> command-line option.
Show Core Modules	Controls display of executable code sections. This setting is equivalent to specifying the <code>-show code nocode</code> command-line option.
Show Extended Mnemonics	Controls display of extended mnemonics. This setting is equivalent to specifying the <code>-show extended noextended</code> command-line option.
Show Source Code	Interleaves the code disassembly with C or C++ source code. This setting is equivalent to specifying the <code>-show source nosource</code> command-line option.
Only Show Operands and mnemonics	Controls display of address and op-code values. This setting is equivalent to specifying the <code>-show binary nobinary</code> command-line option.
Show Data Modules	Controls display of data sections. This setting is equivalent to specifying the <code>-show data nodata</code> command-line option.

Table continues on the next page...

Table 4-24. CodeWarrior build tool settings - Disassembler options (continued)

Option	Explanation
Disassemble Exception Tables	Controls display of C++ exception tables. This setting is equivalent to specifying the <code>-show xtab[les] noxtab[les]</code> or <code>-show exceptions noexceptions</code> command-line option.
Show DWARF Info	Controls display of debugging information. This setting is equivalent to specifying the <code>-show debug nodebug</code> or <code>-show dwarf nodwarf</code> command-line option.
Verbose	Controls display of extra information. This setting is equivalent to specifying the <code>-show detail nodetail</code> command-line option.

4.3.8 Preprocessor

Use the Preprocessor panel to specify the command, options, and expert settings related to the preprocessor.

The table below lists and describes the various options available on the **Preprocessor** panel.

Table 4-25. CodeWarrior build tool settings - Preprocessor options

Option	Explanation
Command	Shows the location of the preprocessor executable file.
All options	Shows the actual command line the preprocessor will be called with.
Command line pattern	Shows the expert settings command line parameters.

4.3.8.1 Preprocessor Settings

Use the Preprocessor Settings panel to specify the preprocessor options that are specific to APP software development.

The table below lists and describes the various options available on the **Preprocessor** panel.

Table 4-26. CodeWarrior build tool settings - Preprocessor options

Option	Explanation
Mode	Specifies the tool to preprocess source files. This setting is equivalent to specifying the <code>-E</code> command-line option.

Table continues on the next page...

Table 4-26. CodeWarrior build tool settings - Preprocessor options (continued)

Option	Explanation
Emit file change	Controls generation of file and line breaks. This setting is equivalent to specifying the <code>-ppopt [no]break</code> command-line option.
Emit #pragmas	Controls generation of <code>#pragma</code> directives. This setting is equivalent to specifying the <code>-ppopt [no]pragmacommand-line</code> option.
Show full path	Controls generation of full paths or just the base file name. This setting is equivalent to specifying the <code>-ppopt [no]full[path]</code> command-line option.
Keep comment	Controls generation of comments. This setting is equivalent to specifying the <code>-ppopt [no]comment</code> command-line option.
Use #line	Controls generation of <code>#line</code> directives. This setting is equivalent to specifying the <code>-ppopt [no]line</code> command-line option.
Keep whitespace	Controls generation of white spaces. This setting is equivalent to specifying the <code>-ppopt [no]space</code> command-line option.



Chapter 5

Working with AIOP Debugger

This chapter explains debugger features that are specific to the CodeWarrior Developer Studio for Advanced Packet Processing.

NOTE

For more information on debugger features that are in all CodeWarrior products, see *CodeWarrior Common Features Guide* available in the `<CWInstallDir>\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

This chapter explains:

- [AIOP debug model](#)
- [AIOP task aware debugging](#)
- [Standard debugging features](#)
- [CodeWarrior Executable Importer wizard](#)
- [Debugging externally built executable file](#)
- [Multi-core operations](#)

5.1 AIOP debug model

This chapter describes how the various AIOP specific notions and components are represented and handled in CodeWarrior debugger.

- [Overview](#)
- [AIOP task specific breakpoints](#)
- [AIOP global halt](#)
- [AIOP running](#)
- [AIOP debug perspective](#)
- [Task stepping mode](#)

5.1.1 Overview

AIOP is a C programmable processor comprised of a variable number of cores which are further grouped into “core clusters” as an implementation artifact. The table below lists and describes various AIOP specific components.

Table 5-1. AIOP specific components

Component	Description
AIOP Core	An AIOP core is an instance of an e200 core comprising the AIOP.
AIOP Cluster	A core cluster consists of four cores, each with their own I-cache, workspace RAM, and a Shared IRAM (Instruction RAM). The AIOP clusters are not represented in any way in the debugger GUI but the user should be aware that the debugger does handle the AIOP clusters internally by downloading code or setting software breakpoints in all AIOP clusters comprising the AIOP.
AIOP	The totality of AIOP cores comprising an AIOP instance is also referred to as the AIOP. The debugger offers run control commands at the AIOP level.
AIOP SMP Architecture	CodeWarrior debugger for AIOP supports Symmetric Multi_Processing (SMP) and is capable of debugging the same executable (or set of executables) on the whole number of cores comprising the AIOP.
AIOP Tasks	The fundamental unit of operation in an AIOP is the task. Tasks are created and terminated by the hardware. A finite number of tasks (maximum of 256) can exist and execute simultaneously inside the AIOP.

5.1.2 AIOP global halt

Global halt represents a state in which all the AIOP cores comprising the AIOP are suspended for debug purposes (not to be confused with AIOP tasks not being scheduled for execution by the task scheduler).

The AIOP can be put in global halt state by user command or by one of its cores hitting a breakpoint type that induces the global halt state. By default, AIOP breakpoints induce the global halt state when hit, the cores being linked together using the hardware cross-triggering mechanism.

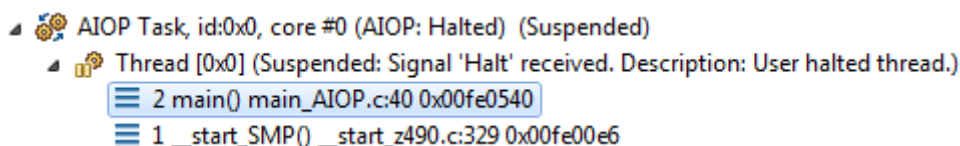


Figure 5-1. AIOP global halt

5.1.3 AIOP running

AIOP running represents a state in which all the AIOP cores comprising the AIOP are in running mode (that is, not suspended for debug purposes). AIOP running state describes only the state of the AIOP cores and not of the tasks. During AIOP running state, a various number of tasks could be suspended for debug purposes but the cores on which they are scheduled are still running other tasks.

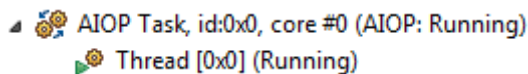


Figure 5-2. AIOP running

5.1.4 AIOP debug perspective

CodeWarrior debugger offers two different perspectives of the AIOP:

- [Task centric perspective](#)
- [Core centric perspective](#)

5.1.4.1 Task centric perspective

This is the default and recommended debug perspective. Most of AIOP specific debug features are available only in this perspective. CodeWarrior debugger operates in task centric perspective only when task awareness services are activated. For more details on how to activate task awareness, refer [Activating task awareness services](#) section.

In the task centric perspective, the debugger offers user introspection and control only for the AIOP and for individual AIOP tasks. The AIOP cores are completely abstracted to the user in this perspective.

User can inspect the scheduling state of the tasks in the AIOP from the **System Browser** view.

Task Aware	Task Id	Core	PC	Status	Accel Id	OSM [State, XPOS, TPOS]:SCOPE_ID	Entry Point
AIOP Tasks	0x0	0	0xfe1418	Ready to execute	NA	[XC, 0x0*, 0x0] : 0x55550000 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x1	0	0xfe0a0e	Executing	NA	[XX, 0x0*, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x2	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x3	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x4	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x5	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x6	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x7	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x8	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0x9	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0xa	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
	0xb	0	0x0	Idle	NA	n/a	n/a
	0xc	0	0x0	Idle	NA	n/a	n/a

Figure 5-3. System Browser-AIOP Tasks

Double-click a task in the **System Browser** view to present it as a process with one thread in the **Debug** view. The task and core IDs are described in the label of the process. Workspace memory, variables and registers associated with the task can also be inspected by double-clicking the task in the **System Browser** view.

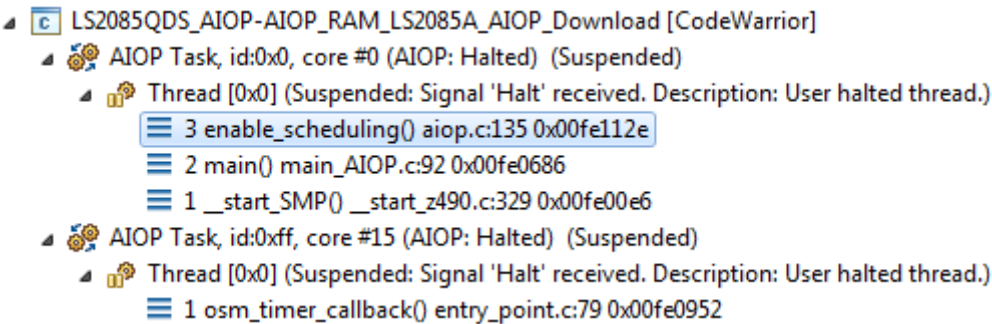


Figure 5-4. Stack crawls for two AIOP tasks

The figure above presents the stack crawls for two AIOP tasks: task 0x0 on core 0 and task 0xff on core 15.

You can perform run control commands both on the AIOP as a whole or on individual tasks (but not on individual cores). Typically, single core run control commands operate at task level and multicore run control commands operate at AIOP level.

5.1.4.2 Core centric perspective

When core centric perspective is used, the CodeWarrior debugger is agnostic of the AIOP Core Task Scheduler which is responsible of AIOP task management. In this case, the debugger becomes a typical core level baremetal tool. Most of AIOP specific debug features are not available in this perspective.

You will get a multicore AIOP debugger that can be successfully used for debugging the active task on each of the cores of the AIOP. You can also perform run control operations at the AIOP level and also at an individual core level.

The figure below displays the stack crawls of active tasks on the first three AIOP cores. Typically, in core centric perspective, all AIOP cores are presented in the **Debug** view at all time during debugging.

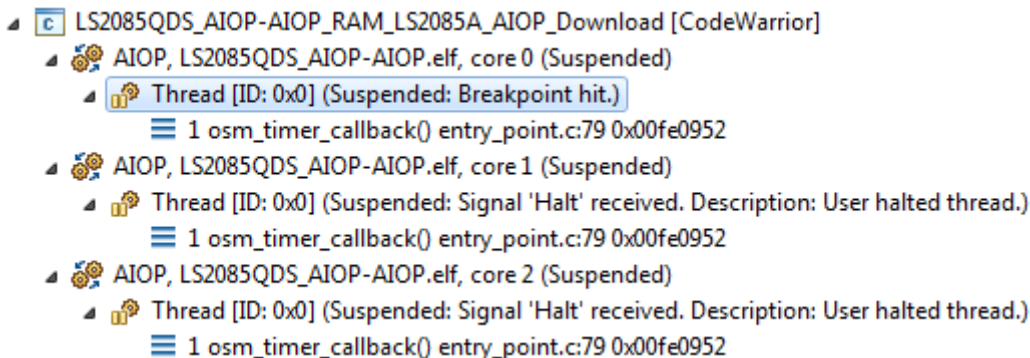


Figure 5-5. Stack crawls of active tasks on the first three AIOP cores

5.1.5 Task stepping mode

Stepping over a line of code from a task will always return in the same task. During stepping the whole AIOP will run and the step will return when the task from where the stepping was originated is an active task again and it has reach at the end of the stepping scope.

Stepping on the AIOP can be done at either task level or at AIOP level.

- Task level stepping leaving AIOP running: At the end of the stepping scope the AIOP is in running. In other words, only the stepped task is suspended after stepping completes.
- Task level stepping with global halt: At the end of the stepping scope the AIOP is in global halt. In other words, the stepped task and the whole AIOP is suspended after stepping completes.

NOTE

You can click the **Task Stepping Mode** button in the **Debug** view to enable the AIOP task stepping mode while the AIOP is either running or halted.

5.2 AIOP task aware debugging

Activating AIOP task awareness services help in tracking and debugging tasks, running inside an AIOP core. The fundamental unit of operation in an AIOP instance is the task. Tasks are created and terminated by the hardware. A finite number of tasks (maximum of 256) can exist and execute simultaneously inside of an AIOP core.

While debugging AIOP core with task awareness services activated:

- cross-triggering is enabled by default.
- stopping an AIOP core stops all AIOP cores.
- resuming a task will put the AIOP, it is allocated to, in a running state.
- terminating the first process, ends the entire session.
- performing any run operations, would terminate all processes except the first process and the one on which the action was performed. The first process that hits the breakpoint will be targeted.

This chapter explains:

- [Activating task awareness services](#)
- [Viewing AIOP tasks](#)
- [Adding task memory location columns in System Browser](#)
- [Viewing task entry point and OSM data in System Browser](#)
- [Targeting AIOP tasks](#)
- [Performing run control operations](#)

5.2.1 Activating task awareness services

To activate the task awareness services:

1. Select an existing AIOP project from the **CodeWarrior Projects** view.
2. From the main menu bar of the IDE, select Run > Debug Configurations.

The Debug Configurations dialog box appears.

3. Expand the CodeWarrior debug configuration.
4. Select the debug configuration that you want to modify.
5. Click **Main** and select **AIOP** from the list of cores.
6. Click **Debugger** and select the **OS Awareness** tab.
7. From the **Target OS** dropdown list, select **AIOP**.
8. Click Apply to save the new settings.

The task awareness services are activated.

5.2.2 Viewing AIOP tasks

The **System Browser** view provides details of all the tasks running inside the AIOP cores. To open the **System Browser** view:

1. Start a debugging session.
2. Choose **Window > Show View > Other** from the IDE menu bar.

The **Show View** dialog appears.

3. Expand the **Debug** group and select **System Browser**.
4. Click **OK**.

The **System Browser** view appears.

5. Select the **AIOP Tasks** tab to view the details for each task running inside the AIOP cores. The table below describes the columns that appear in the view.

Table 5-2. AIOP Tasks column details

Column	
Task ID	Displays the AIOP task id number. This column cannot be hidden from the view.
Core	Displays the core number that the task belongs to.
PC	Displays the current program counter of the task.
Status	Displays the status of the task. Following is the list of supported values: <ul style="list-style-type: none"> • Idle • Allocated • Ready to execute • Executing • Accelerator job requested • Executing on accelerator • Ready to execute, inhibited • Accelerator job requested, inhibited • Executing on accelerator, inhibited
Accel ID	Displays the accelerator id number (if available) of a task called
OSM	Displays the order scope information (if available) for a task. Ordering is about keeping a single task in executing a part of code and the other waiting in line to get exclusive execution. Task can enter in one or more (up to 4) scopes. Following is a list of information displayed: <ul style="list-style-type: none"> • STATE - Specifies the order scope state. The supported values are: <ul style="list-style-type: none"> • Execution concurrent (XC) • Execution exclusive (XX) • Waiting for transition (WT) • Waiting for exclusivity (WX)

Table continues on the next page...

**Table 5-2. AIOP Tasks column details
(continued)**

Column	
	<ul style="list-style-type: none"> • XPOS - Specifies the order scope exclusivity position. The information is valid only when STATE is WX and scope is valid. The supported values are: <ul style="list-style-type: none"> • 0 - Next in line to be granted exclusivity • >0 - Waiting for exclusivity • TPOS - Specifies the order scope transition position. The information is valid only when scope is valid and is used when the STATE is WT, to determine the scope exit order. <ul style="list-style-type: none"> • 0 - First in line to exit scope • >0 - Not first in line to exit • SCOPE_ID - Specifies the outermost order scope identifier. <p>For more details on how to add and view OSM details in the System Browser view, refer Viewing task entry point and OSM data in System Browser.</p>
Entry Point	<p>Displays the entry point address and optionally the name that a task currently executes.</p> <p>For more details on how to add and view Entry point details in the System Browser view, refer Viewing task entry point and OSM data in System Browser.</p>
Custom memory location columns	<p>Displays the user-defined memory locations. A maximum of eight memory locations can be displayed.</p> <p>For more details on how to add and view custom memory locations in the System Browser view, refer Adding task memory location columns in System Browser.</p>

5.2.3 Viewing non-idle tasks only

The **System Browser** view provides an option to filter out the idle tasks. To view the non-idle tasks only, perform the following steps:

1. Start a debugging session.
2. Select **Window > Show View > Other** from the IDE menu bar.

The **Show View** dialog appears.

3. Expand the **Debug** group and select **System Browser**.
4. Click **OK**.

The **System Browser** view appears.

5. Click the **Show Non-Idle Tasks Only** button from the **System Browser** view toolbar menu, as the [Figure 5-6](#) shows.

The **System Browser** view filters out the idle tasks and displays only the non-idle tasks in the view.

[Figure 5-6](#) shows the **System Browser** view with filtered non-idle tasks.

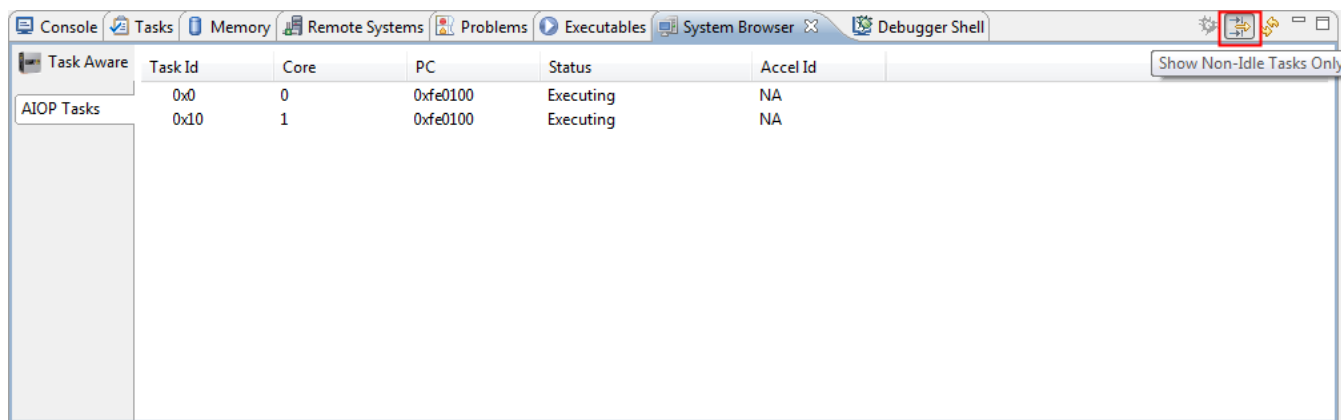


Figure 5-6. System Browser view - Viewing non-idle tasks

5.2.4 Adding task memory location columns in System Browser

The debugger supports defining and selecting a maximum of eight memory locations as columns in the **System Browser** view. The new user defined columns will be placed in the System Browser after the existing pre-defined columns.

To define custom memory locations and add them to the **System Browser** view:

1. Select an existing AIOP project from the **CodeWarrior Projects** view.
2. Select **Run > Debug Configurations**.

The Debug Configurations dialog box appears.

3. Expand the CodeWarrior debug configuration.
4. Select the debug configuration that you want to modify.
5. Click **Main** and select **AIOP** from the list of cores.
6. Click **Debugger** and select the **OS Awareness** tab.
7. From the **Target OS** dropdown list, select **AIOP**.
8. Select **Add task memory location column in System Browser**.
9. Check the **Entry** checkbox. The Entry column controls the enablement of the column in the **System Browser** view.
10. Specify an unique alphanumeric column name.
11. Specify the start memory location address. The specified value can be of hex or decimal formats.
12. Specify the size, in bytes, to read. The specified value can be of hex or decimal formats. If specified in decimal formats, the number should be within the range of 1 - 1024.

NOTE

Input errors are highlighted in red on the **OS Awareness** tab page.

13. Specify the visibility of the column on the **System Browser** view by selecting an appropriate value from the **Visible** column.
14. Click **Apply** to save the new settings.

The custom memory locations are defined.

15. Click **Debug**.
16. Select **Window > Show View > System Browser**.

The **System Browser** view appears. The custom task memory location columns are displayed in the **System Browser** view. The new column header contains details like the name, start address and read size (in bytes).

Task Id	Core	PC	Status	Accel Id	myMemInfo (0xFF00 : 100)
0x0	0	0x1002000	Idle	NA	00000000000000000...
0x10	1	0x10220b4	Idle	NA	00000000000000000...
0x20	2	0x10220b4	Idle	NA	00000000000000000...
0x30	3	0x10220b4	Idle	NA	00000000000000000...
0x40	4	0x10220b4	Idle	NA	00000000000000000...
0x50	5	0x10220b4	Idle	NA	00000000000000000...
0x60	6	0x10220b4	Idle	NA	00000000000000000...
0x70	7	0x10220b4	Idle	NA	00000000000000000...
0x80	8	0x10220b4	Idle	NA	00000000000000000...
0x90	9	0x10220b4	Idle	NA	00000000000000000...
0xa0	10	0x10220b4	Idle	NA	00000000000000000...
0xb0	11	0x10220b4	Idle	NA	00000000000000000...
0xc0	12	0x10220b4	Idle	NA	00000000000000000...
0xd0	13	0x10220b4	Idle	NA	00000000000000000...
0xe0	14	0x10220b4	Idle	NA	00000000000000000...
0xf0	15	0x10220b4	Idle	NA	00000000000000000...

Figure 5-7. System Browser-AIOP Tasks

17. To control the appearance of the columns, right-click on the **System Browser** view and select **Add/Remove columns**.
A list of pre-defined and custom memory location columns, even the ones that are marked hidden on the **OS Awareness** tab, appears.

NOTE

All columns except the **Task ID** column can be selected from the context menu that appears. Selecting a hidden column on the list also updates its **Entry** value on the **OS Awareness** tab.

18. Uncheck a column to hide its details. Alternatively, check a column to view its details in the **System Browser** view.

NOTE

Click the **Refresh** button on the **System Browser** view to retrieve the data from memory if a ? character appears in the column.

19. Terminate the debug session.

5.2.5 Viewing task entry point and OSM data in System Browser

The debugger supports viewing the task entry point and order scope manager (OSM) data details as columns in the **System Browser** view. These columns will be placed in the **System Browser** after the existing pre-defined columns.

To view the task information in the **System Browser** view:

1. Select an existing AIOP project from the **CodeWarrior Projects** view.
2. Select **Run > Debug Configurations**.

The Debug Configurations dialog box appears.

3. Expand the CodeWarrior debug configuration.
4. Select the debug configuration that you want to modify.
5. Click **Main** and select **AIOP** from the list of cores.
6. Click **Debugger** and select the **OS Awareness** tab.
7. From the **Target OS** dropdown list, select **AIOP**.
8. Select **Show task entry point in System Browser**.
9. Select **Show task OSM data in System Browser**.
10. Select **Retrieve information only for non-idle tasks**. Selecting this option will improve performance by retrieving information about the active tasks only.
11. Click **Apply** to save the new settings.
12. Click **Debug**.
13. Select **Window > Show View > System Browser**.

The **System Browser** view appears. The **OSM** and **Entry Point** columns are displayed in the **System Browser** view.

Task Id	Core	PC	Status	Accel Id	OSM [State, XPOS, TPOS]:SCOPE_ID	Entry Point
0x0	0	0xfe1418	Ready to execute	NA	[XC, 0x0*, 0x0] : 0x55550000 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x1	0	0xfe0a0e	Executing	NA	[XX, 0x0*, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x2	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x3	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x4	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x5	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x6	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x7	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x8	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0x9	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0xa	0	0xfe0992	Ready to execut...	NA	[WX, 0x1, 0x0] : 0x12345600 \ [XC, 0x0*, 0x0*]	timer_callback(0xfe0992)
0xb	0	0x0	Idle	NA	n/a	n/a
0xc	0	0x0	Idle	NA	n/a	n/a
...

Figure 5-8. System Browser-AIOP Tasks

5.2.6 Targeting AIOP tasks

While debugging AIOP with task awareness services activated, following targeting actions can be performed on the tasks:

Read/Write Registers

1. In the **Debug** view, click **Resume**.
2. Click **Terminate**.
3. Open System Browser.
4. Target another process from the **System Browser**.
5. Perform some read/write operations on the private task registers (including the entire GRP set) from the **Register view**.

Observe different GPR set for each process.

Read/Write Memory

1. In the **Debug** view, click **Resume**.
2. Click **Terminate**.
3. Open System Browser.
4. Target another process from the **System Browser**.
5. Open **Memory View**.
6. Perform some read/write operations on the private memory zones and on shared memory zones.

Observe that a write in private zones would not affect the same addresses for other processes.

Per-Task Global Variables

1. In the **Debug** view, click **Resume**.
2. Click **Terminate**.
3. Open System Browser.
4. Target another process from the **System Browser**.
5. For both the processes, open the **Variables** view and add a global variable that has been declared as per-task variable, using `__declspec(section ".tdata")`.

Observe that the variables have the same virtual address but they point to different physical addresses and in consequence have different values.

5.2.7 Performing run control operations

While debugging AIOP with task awareness services activated, following run control operations can be performed on the tasks:

AIOP multicore run/stop

1. Debug AIOP with task awareness services activated.
2. In the **Debug** view, click **Multicore Resume** or **Resume**.

All AIOP cores start executing. If any, the tasks are removed from debug inhibited for scheduling state.

3. In the **Debug** view, click **Multicore Suspend**.

All AIOP cores enter the debug mode.

AIOP cross-triggering

1. Debug AIOP with task awareness services activated.
2. In the **Debug** view, select the first process.
3. In the **Debug** view, click **Multicore Resume**.

All AIOP cores start executing.

4. In the **Debug** view, click **Multicore Suspend**. to suspend the first process.

Observe that all AIOP cores enter the debug mode.

NOTE

Cross-Trigering is activated by default.

AIOP run task X

1. Debug AIOP with task awareness services activated.

2. Target a task, that is allocated on core X, from the **System Browser** view.
3. Select the targeted process.
4. In the **Debug** view, select the targeted process and click **Resume**.

Observe that the task is resumed if the AIOP is already running. Otherwise if the system is in global halt mode the entire AIOP is resumed. If any, the tasks are removed from debug inhibited for scheduling state.

AIOP run task using breakpoints

1. Debug AIOP with task awareness services activated.
2. In the **Debug** view, select the first process.
3. Set a breakpoint.
4. In the **Debug** view, click **Resume**.

The task that hit the breakpoint, will be targeted, if it is not already targeted. At this instance, all cores are stopped.

5. In the **Debug** view, click **Multicore Resume**.

The processes that hit the breakpoint is targeted in the **Debug** View.

Advanced run/control

1. Debug AIOP with task awareness services activated.
2. In the **Debug** view, select a process.
3. Set a breakpoint.
4. In the **Debug** view, click **Resume**.

The task that hit the breakpoint, will be targeted, if it is not already targeted. At this instance, all cores are stopped.

5. In the **Debug** view, click **Step Over**.

The current task will be killed (not if it is the first process) and will be targeted again, upon the execution of the next statement.





6. In the **Debug** view, click **Step Into**.

The task will be killed and targeted again, upon a successful step into operation.

AIOP run control commands

[Table 5-3](#) lists and describes the run control commands in the CodeWarrior IDE.

Table 5-3. AIOP run control commands

Command	Icon	Description	
		AIOP running	AIOP halted (Suspended)
Single-core commands			
Resume		Resumes the suspended task	Resumes the whole AIOP
Suspend		Suspends the AIOP	Not applicable
Multi-core commands			
Multicore Resume		Resumes all debug inhibited tasks	Resumes the whole AIOP
Multicore Suspend		Suspends the AIOP	Not applicable

5.3 Standard debugging features

This section describes debugging features that apply to bareboard debugging:

- [Connection types](#)
- [Editing system configuration](#)
- [CodeWarrior command-line debugger](#)
- [Memory configuration file](#)
- [Displaying memory contents](#)
- [Displaying register contents](#)
- [Using register details window](#)
- [Setting watchpoints](#)
- [Setting breakpoints](#)
- [Setting stack depth](#)
- [Changing program counter value](#)
- [Hard resetting](#)
- [Loading and saving memory](#)
- [Filling memory](#)

5.3.1 Connection types

The debugger supports configuring the following connection types for connecting the target board with a computer:

- [CCSSIM2 ISS](#)
- [CodeWarrior TAP](#)

5.3.1.1 CCSSIM2 ISS

Select this connection type to connect to simulators based on ccssim2 interface. To configure the settings of a **CCSSIM2 ISS** connection type:

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

The **Debug Configurations** window appears.

2. In the **Connection** group, click **Edit**.

The **Properties for <connection launch configuration>** window appears.

3. Select **CCSSIM2 ISS** from the **Connection type** dropdown.

The **Connection** tab and the **Advanced** tab appears with the respective settings of a connection type.

The table below describes various options available on the **Connection** tab page.

Table 5-4. CCSSIM2 ISS - Connection Tab Options

Option		Description
CCS server	Automatic launch	Select to automatically launch the specified CCS server on the specified port.
	Server port number	Specifies the port number to launch the CCS server on.
	CCS executable	Click to specify the path of, or browse to, the executable file of the CCS server.
	Manual launch	Select to manually launch the specified CCS server on the specified port.
	Server hostname/IP	Specifies hostname or the IP address of the CCS server.
	Server port number	Specifies the port number to launch the CCS server on.
	Connect server to TAP	Check to have the CCS server connect to the TAP.

The table below describes the various options available on the **Advanced** tab page.

Table 5-5. CCSSIM2 ISS - Advanced Tab Options

Option		Description
Target connection lost settings	Try to reconnect	If this option is selected, the lost CCS connection between the target and host is reset. Check the Timeout check box to specify the time interval (in seconds) after which the connection will be lost.

Table continues on the next page...

Table 5-5. CCSSIM2 ISS - Advanced Tab Options (continued)

Option		Description
	Terminate the debug session	If this option is selected, the debug session is terminated and the lost connection between JTAG and CCS server is not reset.
	Ask me	This is the default setting. If the CCS connection is lost between the target and host, the user is asked if the connection needs to be reset or terminated.
Advanced CCS settings	CCS timeout	Specifies the CCS timeout period. If the target does not respond in the provided time-interval, you receive a CCS timeout error.
	Enable logging	Check to display protocol logging in console.

5.3.1.2 CodeWarrior TAP

Select this connection type when either the CodeWarrior TAP is used as interface to communicate with the hardware device. To configure the settings of a **CodeWarrior TAP** connection type:

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

The **Debug Configurations** window appears.

2. In the **Connection** group, click **Edit**.

The Properties for <connection launch configuration> window appears.

3. Select the **CodeWarrior TAP** from the Connection type drop-down list.

The **Connection** tab and the **Advanced** tab appears with the respective settings of a connection type.

The table below describes various options available on the Connection tab page.

Table 5-6. CodeWarrior TAP - Connection Tab Options

Option		Description
CodeWarrior TAP	Hardware Connection	Specifies CodeWarrior TAP interface to communicate with the hardware device. CodeWarrior TAP supports both USB and Ethernet network interfaces.
	Hostname/IP	Specifies hostname or the IP address of the TAP.
	Serial Number	Check and specify the USB serial number of the USB TAP; required only if using multiple CodeWarrior USB TAPs.
JTAG settings	JTAG clock speed (kHz)	Specifies the JTAG clock speed. By default, set to 10230 kHz.
CCS server	Automatic launch	Select to automatically launch the specified CCS server on the specified port.

Table continues on the next page...

Table 5-6. CodeWarrior TAP - Connection Tab Options (continued)

Option		Description
	Server port number	Specifies the port number to launch the CCS server on.
	CCS executable	Click to specify the path of, or browse to, the executable file of the CCS server.
	Manual launch	Select to manually launch the specified CCS server on the specified port.
	Server hostname/IP	Specifies hostname or the IP address of the CCS server.
	Server port number	Specifies the port number to launch the CCS server on.
	Connect server to TAP	Check to have the CCS server connect to the CodeWarrior TAP.

The table below describes the various options available on the Advanced tab page.

Table 5-7. CodeWarrior TAP - Advanced Tab Options

Option		Description
Target connection lost settings	Try to reconnect	If this option is selected, the lost CCS connection between the target and host is reset. Check the Timeout check box to specify the time interval (in seconds) after which the connection will be lost.
	Terminate the debug session	If this option is selected, the debug session is terminated and the lost connection between JTAG and CCS server is not reset.
	Ask me	This is the default setting. If the CCS connection is lost between the target and host, the user is asked if the connection needs to be reset or terminated.
Advanced CCS settings	CCS timeout	Specifies the CCS timeout period. If the target does not respond in the provided time-interval, you receive a CCS timeout error.
	Enable logging	Check to display protocol logging in console.
JTAG config file		This panel displays the JTAG configuration file being used. This panel is populated only if you select a JTAG configuration file from the System type drop-down list. If a JTAG configuration file is not selected, this panel displays a None value. For more details on JTAG configuration files, refer to the chapter.
Advanced TAP settings	Force shell download	Check to force a reload of the TAP shell software.
	Disable fast download	Check to disable fast download. NOTE: This option is not available for <i>e500mc</i> , <i>e5500</i> , <i>e6500</i> core based targets.
Advanced TAP settings	Enable JTAG diagnostics	When checked, the option enables performing advanced diagnostics of the JTAG connection to be used during custom board bring-up. After the connection to the probe has been established the debugger performs the JTAG diagnostics tests (Power at probe, IR scan check, Bypass (DR) scan check, Arbitrary TAP state move, IDCODE scan check) and the result of the tests are printed to the console log and in case of an error, a CodeWarrior Alert box appears. When this option is not checked, the CodeWarrior debugger only performs a limited test while configuring the JTAG chain. It checks if the PWR pin is correctly connected and displays a Cable disconnected error if not connected properly. The connection details are provided in the CCS protocol log along with the JTAG ID and in case of an error, a CodeWarrior Alert box appears.

Table continues on the next page...

Table 5-7. CodeWarrior TAP - Advanced Tab Options (continued)

Option		Description
	Secure debug key	Check to enable the debugger to unlock the secured board with the secure debug key provided in the associated text box. If this option is not checked, you will receive a secure debug violation error when you try to debug on the locked board. NOTE: If you provide a wrong key and an unlock sequence is run by the debugger with the erroneous key, the associated part will be locked until a rest occurs and you will need to reset the target to connect again.
	Reset Delay (ms)	Configures the time in milliseconds to delay after PoR is deasserted but before PoD is deasserted; it defaults to 200ms. The delay should be increased for supporting SPI/SD boot scenarios in which the PBL is used to perform boot image manipulation (for example, copying u-boot from SPI flash to internal cache/SRAM during reset) that does not complete in the default reset timeout window. Reset Delay is supported for processors based on GPP cores.

5.3.2 Editing system configuration

The Remote System Configuration model defines the connection and system configurations where you can define a single system configuration that can be referred to by multiple connection configurations. To edit the system configuration:

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

The **Debug Configurations** window appears.

2. In the **Connection** panel, click **Edit**.

The **Properties for <connection launch configuration>** window appears.

3. Click the **Edit** button next to the **Target** dropdown.

The **Properties for <system launch configuration>** window appears.

4. Select the appropriate system type from the **Target type** drop-down list.
5. Make the respective settings in the **Initialization** and **Memory** tabs.
6. Click **OK** to save the settings.
7. Click **OK** to close the **Properties** window.

5.3.2.1 Initialization

Use this tab to specify target initialization file for various cores.

The figure below shows the **Initialization** tab page.

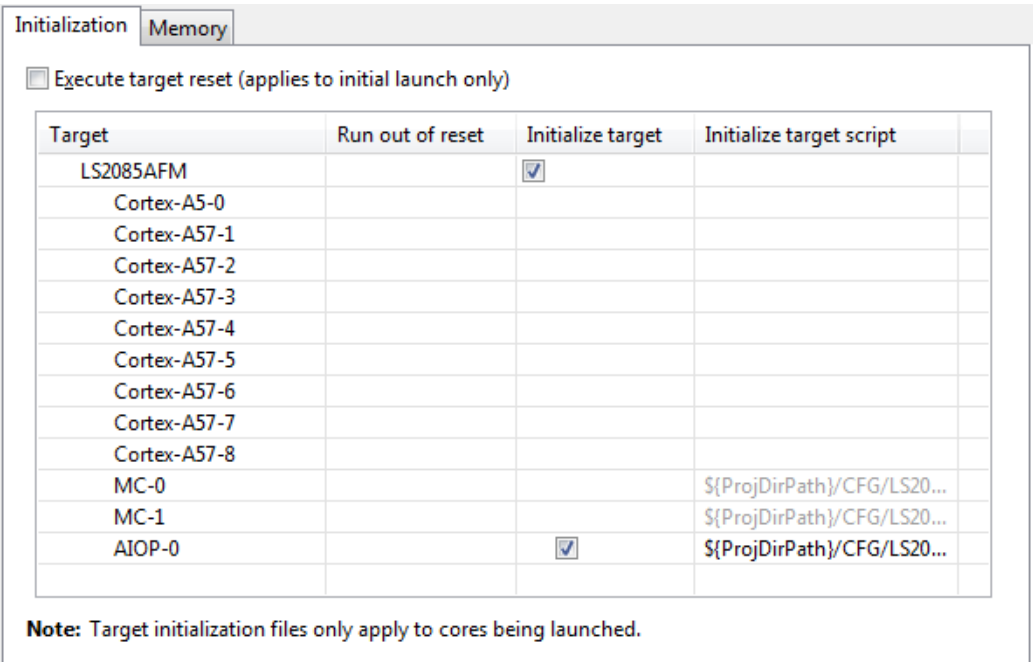


Figure 5-9. Hardware or Simulator Target connection type-Initialization tab

The table below lists the various options available on the **Initialization** tab page.

Table 5-8. Initialization Tab Options

Option	Description
Execute target reset	Check to execute target system reset.
Run out of reset	Check to include the respective core for run out of reset operation.
Initialize target	Click to specify a target initialization file for the respective core.
Initialize target script	Lists the path to a Debugger Shell Tcl script that runs when launching a debug session for the respective core. To edit, select a cell, then click Click the Ellipsis button to open the Target InitializationFile dialog box. The settings for a group of cores can be changed all at once by editing the cell of a common ancestor node in the Target hierarchy.

5.3.2.2 Memory

Use this tab to specify memory configuration file for various cores.

The figure below shows the **Memory** tab page.

Initialization		Memory	
Target	Memory configuration	Memory configuration file	
LS2085AFM	<input checked="" type="checkbox"/>		
Cortex-A5-0			
Cortex-A57-1			
Cortex-A57-2			
Cortex-A57-3			
Cortex-A57-4			
Cortex-A57-5			
Cortex-A57-6			
Cortex-A57-7			
Cortex-A57-8			
MC-0		\${ProjDirPath}/CFG/LS2085A-A...	
MC-1		\${ProjDirPath}/CFG/LS2085A-A...	
AIOP-0	<input checked="" type="checkbox"/>	\${ProjDirPath}/CFG/LS2085A-A...	

Figure 5-10. Hardware or simulator target connection type - Memory tab

The table below lists the various options available on the **Memory** tab page.

Table 5-9. Memory tab options

Option	Description
Target	Lists the targets and the supported cores.
Memory configuration	Click to specify a memory configuration file for the respective core.
Memory configuration file	Lists the path to the memory configuration file for the respective core. To edit, select a cell, then click the Ellipsis button to open the Memory Configuration File dialog box. The settings for a group of cores can be changed all at once by editing the cell of a common ancestor node in the Target hierarchy.

5.3.3 CodeWarrior command-line debugger

CodeWarrior supports a command-line interface for some of its features including the debugger. You can use the command-line interface together with various scripting engines, such as the Microsoft® Visual Basic® script engine, the Java™ script engine, TCL, Python, and Perl. You can even issue a command that saves your command-line activity to a log file.

You use the Debugger Shell window to issue command lines to the IDE. For example, you enter the command `debug` in this window to start a debugging session. The window displays the standard output and standard error streams of command-line activity.

To open the Debugger Shell window, follow these steps:

1. Switch the IDE to the Debugger perspective and start a debugging session.
2. Select **Window > Show View > Other**.

The **Show View** dialog box appears.

3. Expand the **Debug** tree.
4. Select **Debugger Shell**.
5. Click **OK**.

The **Debugger Shell** view appears in the view stack at the bottom of the IDE.

To issue a command-line command, type the desired command at the command prompt (`%>`) in the **Debugger Shell** window, then press **Enter** or **Return**. The command-line debugger executes the specified command.

NOTE

To display a list of 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.

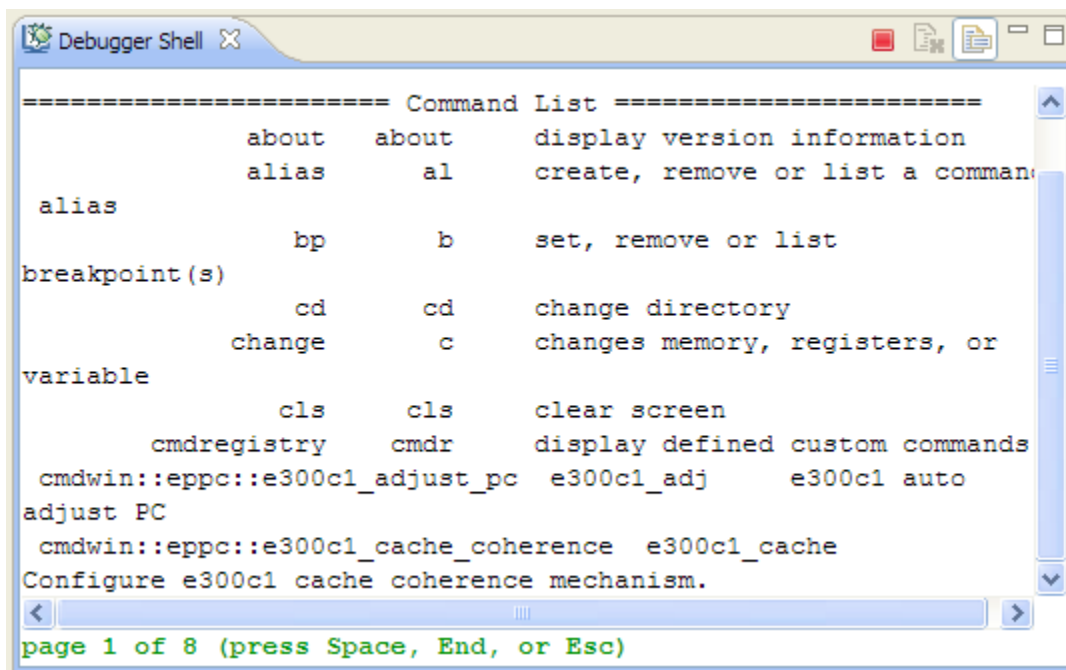


Figure 5-11. Debugger Shell Window

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.

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



from the view toolbar.

The table below lists the instructions for common command-line debugging tasks.

Table 5-10. Common Command-Line Debugging Tasks

Task	Instruction	Comments
Open the Debugger Shell	Select Windows > Show View > Others > Debugger Shell	The Debugger Shell view appears.
Use the help command	1. On the Debugger shell command prompt (%>), type help . 2. Press Enter.	The command list for CodeWarrior appears.
Enter a command	1. On the Debugger shell, type a command followed by a space. 2. Type any valid command-line options, separating each with a space. 3. Press Enter.	You can use shortcuts instead of complete command names, such as k for kill .
View debug command hints	Type alias followed by a space	The syntax for the rest of the command appears.
Review previous commands	Press Up Arrow and Down Arrow keys	
Clear command from the command line	Press the Esc key	
Stop an executing script	Press the Esc key	
Toggle between insert/overwrite mode	Press the Insert key	
Scroll up/ down a page	Press Page Up or Page Down key	
Scroll left/right one column	Press Ctrl-Left Arrow or Ctrl-Right Arrow keys	
Scroll to beginning or end of buffer	Press Ctrl-Home or Ctrl-End keys	

5.3.4 Memory configuration file

When debugging a bareboard system, the memory configuration file instructs the debugger about non 1:1 MMU translations. The file is also required for the 36-bit physical addresses because of the 32-bit virtual address space to 36-bit physical address space translation.

In general, a processor core has virtual memory support for 232 bytes of effective address space and real memory support for 236 bytes of physical address space. Therefore, only the physical address space is 36-bit wide, while the effective address space remains 32-bit wide.

The processor executes in the effective address space. Therefore, to have the processor utilize the entire 36-bit physical address space, you define a Memory Management Unit (MMU) to translate 32-bit effective addresses to 36-bit real addresses.

Tip

A memory configuration file must not be related directly/only to the 36-bit addressing features.

For more information on memory configuration files, refer to the appendix.

5.3.5 Displaying memory contents

The debugger allocates multiple memory spaces in the IDE for flexible control over the memory access. The number of supported memory spaces and their properties depends upon the debugged processor.

You can display and access the supported memory spaces for a target in the Memory and Memory Browser Views, in the Import/Export/Fill Memory Action Task View or in the Debugger Shell using the designated memory space prefix. Use the `mem -ms` command to list the supported memory spaces for a processor.

To display the **Memory** view, select **Window > Show View > Other... > Debug > Memory**. [Figure 5-12](#) shows a **Memory** view displaying physical memory address space.

Address	0 - 3	4 - 7	8 - B	C - F
0100002420	480032E1	4BFFFFBD	3BDE0001	4BFFFFFC
0100002430	9421FFE0	7C0802A6	90010024	93E1001C
0100002440	93C10018	7C7E1B78	7FE000A6	67FF0200
0100002450	0x100002430	2C1E0C00	40820020	48000051
0100002460	7C641B78	3C600000	38636EA0	4CC63182
0100002470	48003291	48000020	48000035	7C641B78
0100002480	3C600000	38636ECC	7FC5F378	4CC63182
0100002490	48003271	83E1001C	83C10018	80010024
01000024A0	7C0803A6	38210020	4E800020	7C7E42A6
01000024B0	4E800020	9421FFFF	38210010	4E800020
01000024C0	9421FFFF	7C0802A6	90010014	48000015

Figure 5-12. Memory View

NOTE

The **Memory** view seamlessly displays 32-bit and 36-bit addresses depending upon the selected memory space and the target processor currently under debug process.

To display the supported memory spaces for a target in the Memory view, perform the following steps:

1. In the Memory view, click the **Add Memory Monitor** icon .

The **Monitor Memory** dialog box appears.

2. Specify the address in the **Enter address or expression to monitor** drop-down list.
3. Select one of the supported memory spaces from the **Memory space** drop-down list.
 - SoC Internal Address Space (s)

Select to directly access LS SoC Internal Address Map in an external mode (not through MC or AIOP core). The SoC Internal Address Map refers the SoC internal address space directly accessible by an internal transaction source such as a general purpose processor core.

- Virtual (v)

Select to indicate that the specified address space is same as the address space in which the processor executes. When you select the *Virtual* option, the debugger performs virtual to physical translations based on the translate directives specified in the memory configuration file (for bare-board debugging) or the kernel awareness plug-in (for Linux debugging). In addition, the *Virtual* memory space is the one that is relevant for the Program Counter (PC) and the Stack

Pointer (SP) registers. The width of the *Virtual* memory space is determined by the target processor's effective address size. For MC and AIOP cores the width of the *Virtual* memory space is 32-bit. For 6500 cores, the width of the *Virtual* memory is 64-bit. Note that the *Virtual* memory space is the default memory space in the **Disassembly** view.

5.3.6 Displaying register contents

Use the **Registers** view to display and modify the contents of the registers of the processor on your target board. To display this view, select **Window > Show View > Other... > Debug > Registers**.

The **Registers** view displays categories of registers in a tree format. To display the contents of a particular category of registers, expand the tree element of the register category of interest. [Figure 5-13](#) shows the **Registers** view with the General Purpose Registers tree element expanded.

Tip

You can also view and update registers by issuing the reg, change, and display commands in the CodeWarrior Debugger Shell window.

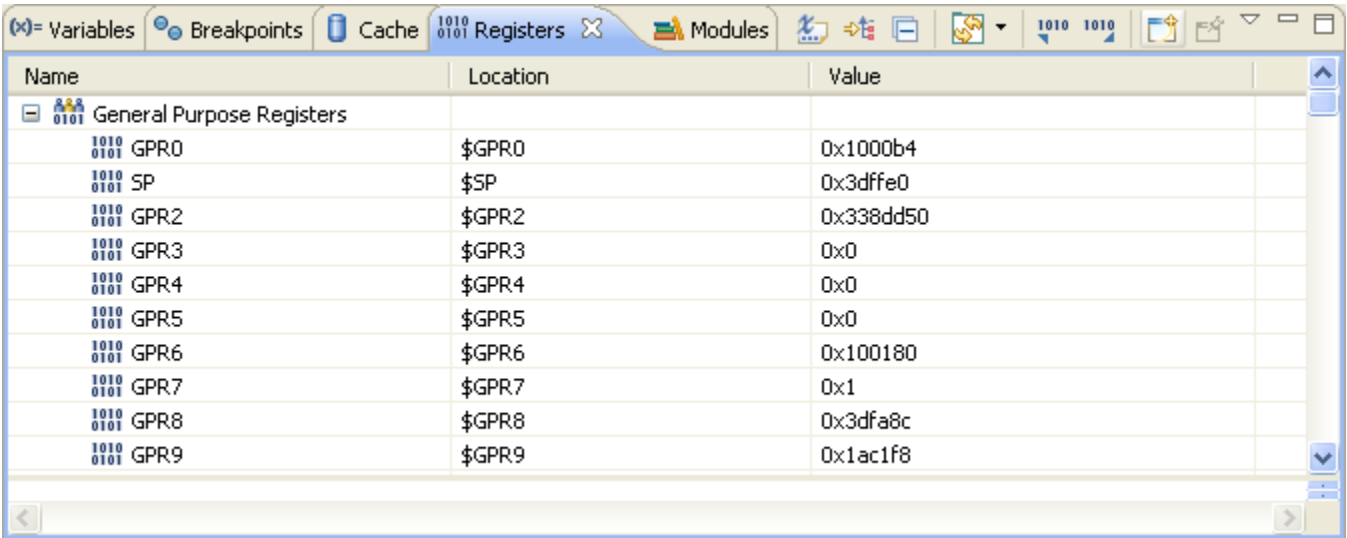


Figure 5-13. Registers View

5.3.6.1 Adding register group

The default display of the Registers view groups related registers into a tree structure. You can add a custom group of registers to the default tree structure. To add a new register group:

1. Right-click in the Registers view.

A context menu appears.

2. Select Add Register Group from the context menu.

The Register Group dialog box appears ([Figure 5-14](#)).

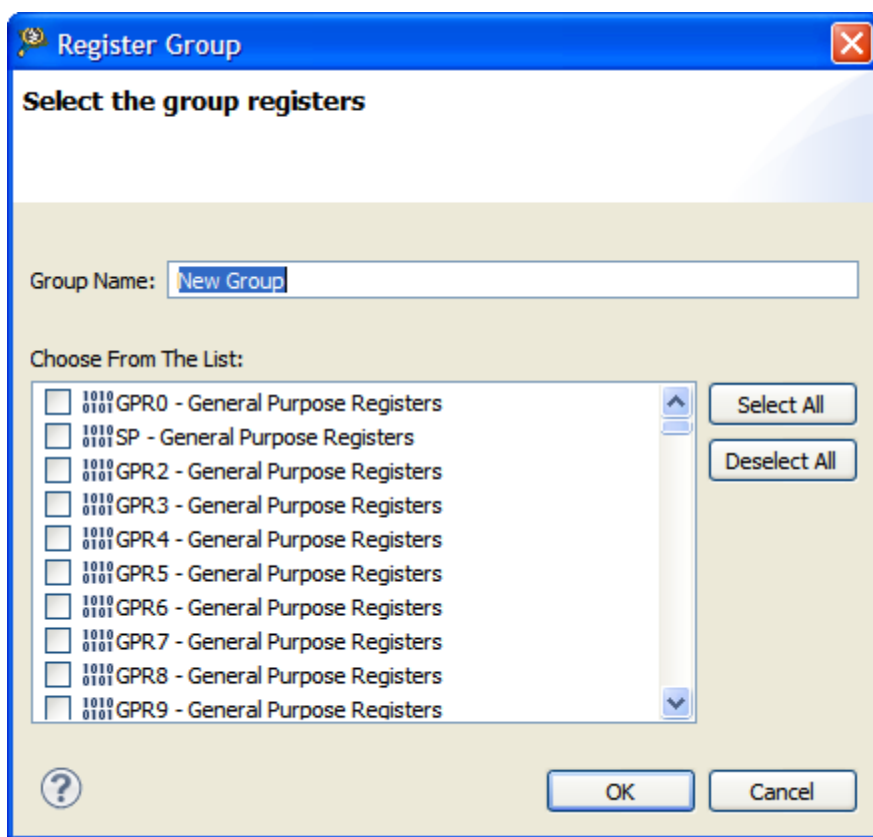


Figure 5-14. Register Group Dialog Box

3. Enter in the Group Name text box a descriptive name for the new group.
4. Check the check box next to each register you want to appear in the new group.

Tip

Click Select All to check all of the check boxes. Click Deselect All to clear all of the check boxes.

5. Click OK.

The Register Group dialog box closes. The new group name appears in the Registers view.

5.3.6.2 Editing register group

In the Registers view, you can edit both the default register groups and the groups that you add. To do so:

1. In the Register view, right-click the name of the register group you want to edit.

A context menu appears.

2. Select Edit Register Group from the context menu.

The Register Group dialog box appears ([Adding register group](#)).

3. If you wish, enter in the Group Name text box a new name for the group.
4. Check the check box next to each register you want to appear in the group.

Tip

Click Select All to check all of the check boxes. Click Deselect All to clear all of the check boxes.

5. Click OK.

The Register Group dialog box closes. The new group name appears in the Registers view.

5.3.6.3 Removing register group

In the Registers view, you can remove register groups. To remove a register group:

1. In the Registers view, right-click the name of register group that you wish to remove.

A context menu appears.

2. Select Remove Register Group from the context menu.

The selected register group disappears from the Registers view.

5.3.6.4 Changing register bit value

To change a bit value in a register, first switch the IDE to the Debugger perspective and start a debugging session. Now proceed as follows:

1. Open the Registers view by selecting Window > Show View > Other > Debug > Registers.
2. In the Registers view, expand the register group that contains the register with the bit value that you want to change.
3. Click on the register's current bit value in the view's Value column.

The value becomes editable.

4. Type in the new value.
5. Press the Enter key.

The debugger updates the bit value. The bit value in the **Value** column changes to reflect your modification.

5.3.7 Using register details window

The default state of the **Registers** view is to provide details on the processor's registers. The following figure shows the **Register** view with detailed information.

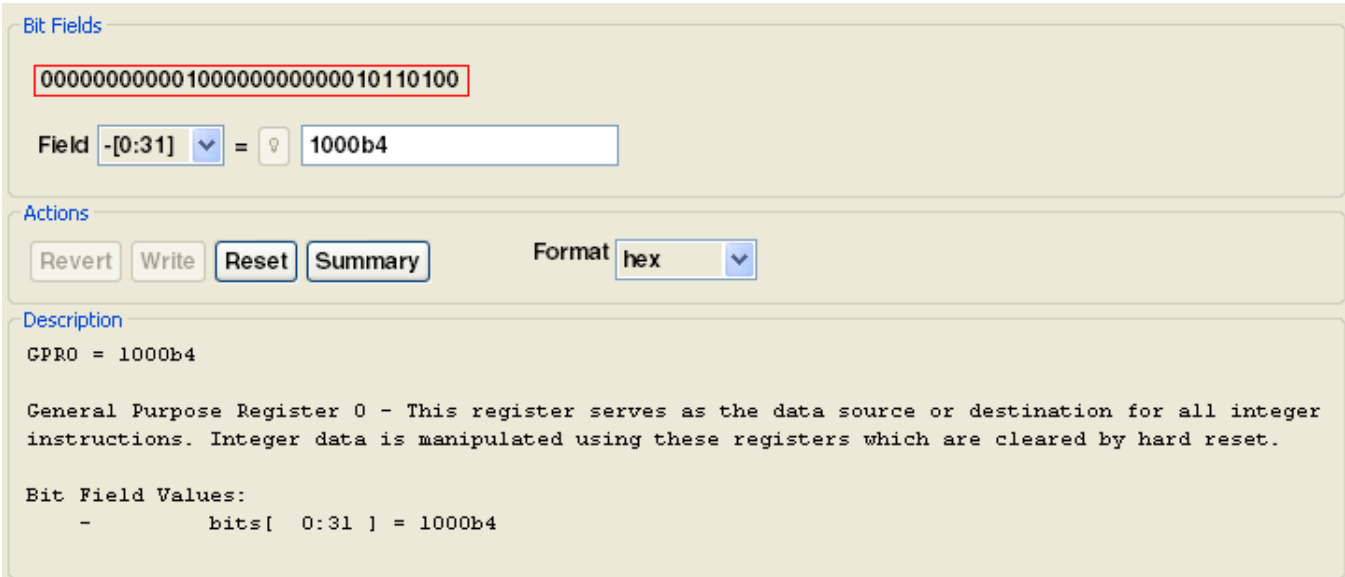


Figure 5-15. Register view with detailed information

The Registers view displays several types of register details:

- [Bit Fields](#)
- [Actions](#)
- [Description](#)
- [Viewing register details](#)
- [Changing bit field](#)

NOTE

You have to expand the view then click-and-drag the areas at the bottom of the Registers view to reveal the Bit Field, Actions, and Description portions of the view.

5.3.7.1 Bit Fields

The **Bit Fields** group of the Registers view 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 in order to see additional information.

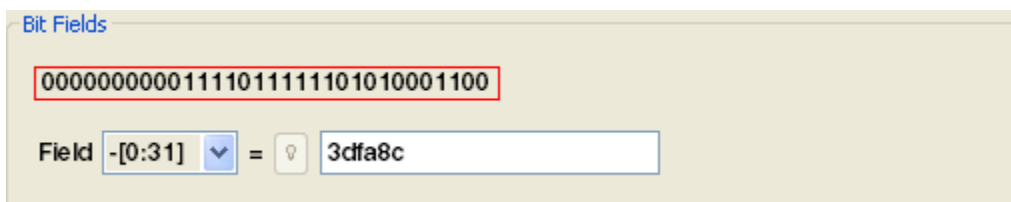


Figure 5-16. Register Details, Bit Fields Group

Tip

You can also view register details by issuing the `reg` command in the Debugger Shell window.

A bit field is either a single bit or a collection of bits within a register. Each bit field has a mnemonic name that identifies it. You can use the **Field** list box to view and select a particular bit field of the selected register. The list box shows the mnemonic name and bit-value range of each bit field. In the Bit Fields graphical representation, a box surrounds each bit field. A red box surrounds the bit field shown in the Field list box.

After you use the Field list box to select a particular bit field, you see its current value in the = text box. If you change the value shown in the text box, the Registers view shows the new bit-field value.

5.3.7.2 Actions

Use the Actions group of the Registers view to perform various operations on the selected register's bit-field values.

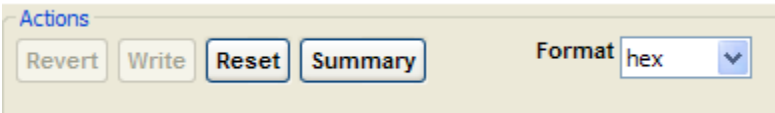


Figure 5-17. Register View, Actions Group

This table lists each item in the Actions group and explains the purpose of each.

Table 5-11. Actions Group Items

Item	Description
Revert	Discard your changes to the current bit-field value and restore the original value. The debugger disables this button if you have not made any changes to the bit-field value.
Write	Save your changes to the current bit-field value and write those changes into the register's bit field. The debugger disables this button after writing the new bit-field value, or if you have not made any changes to that value.
Reset	Change each bit of the bit-field value to its register-reset value. The register takes on this value after a target-device reset occurs. To confirm the bit-field change, click the Write button. To cancel the change, click the Revert button.
Summary	Display Description group content in a pop-up window. Press the Esc key to close the pop-up window.
Format	Specify the data format of the displayed bit-field values.

5.3.7.3 Description

The Description group of the Registers view shows explanatory information for the selected register.

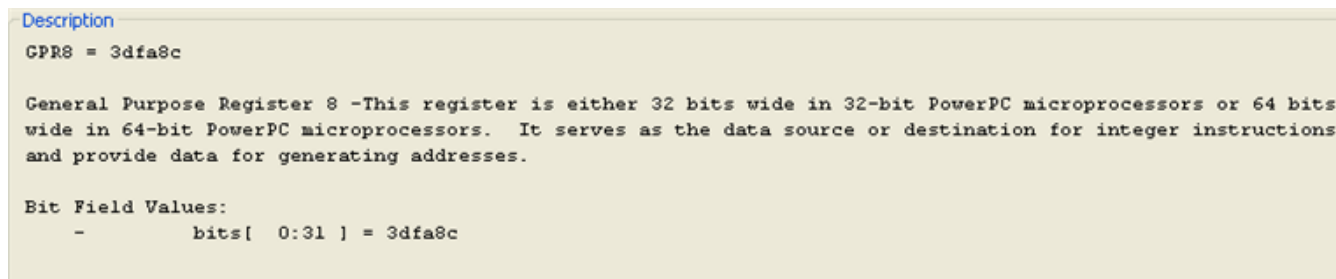


Figure 5-18. Register View, Description Group

The information covers the register's:

- Name
- Current Value
- Description
- Bit field explanations and values

Some registers have multiple modes (meaning that the register's bits can have multiple meanings, depending on the current mode). If the register you examine has multiple modes, you must select the appropriate mode.

5.3.7.4 Viewing register details

To open the Registers view, you must first start a debugging session.

To see the registers and their descriptions:

1. In the Debug perspective, click the Registers tab.

The Registers view appears ([Figure 5-19](#)).

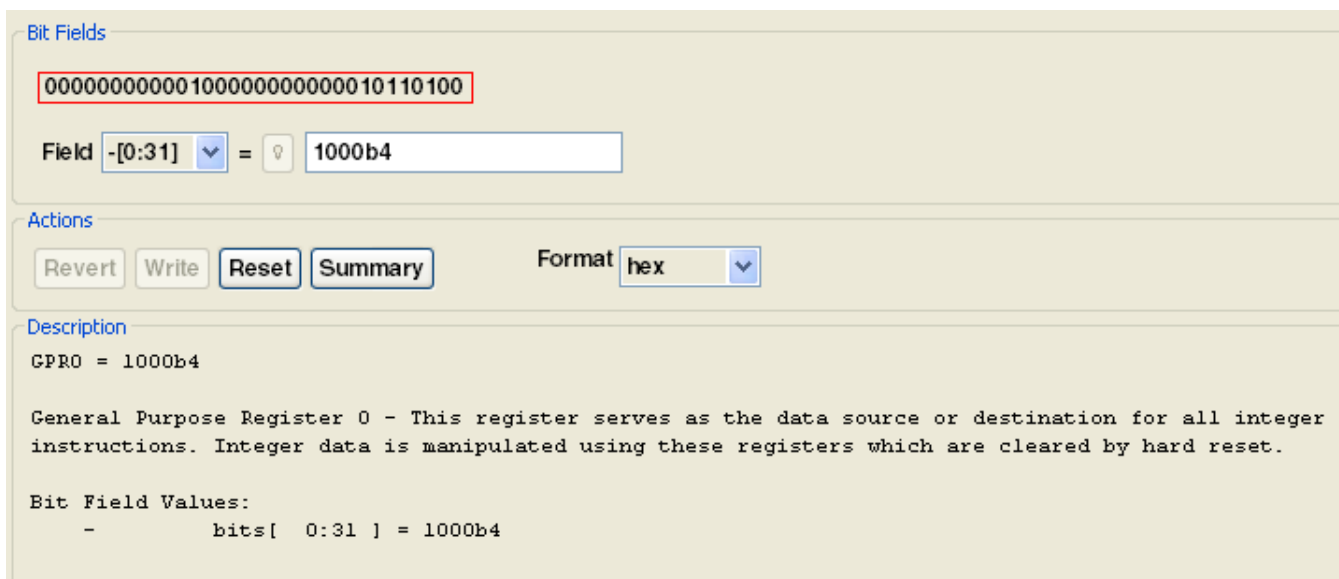


Figure 5-19. Registers view - Register details

2. Click the toolbar's menu button (the inverted triangle).
3. Select Layout > Vertical or Layout > Horizontal to see the register details.

NOTE

Selecting Layout > Registers View Only hides the register details.

4. Expand a register group to see individual registers.
5. Select a specific register by clicking on it.

The debugger enables the appropriate buttons in the Actions group of the Registers view.

NOTE

Use the Format list box to specify the format of data that appears in the Registers view.

6. Use the Register view to examine register details.

For example, examine register details in these ways:

- Expand the Bit Fields group to see a graphical representation of the selected register's bit fields. You can use this graphical representation to select specific bits or bit fields.
- Expand the Description group to see an explanation of the selected register, bit field, or bit value.

Tip

To enlarge the Registers view, click the Maximize button of the view's toolbar. After you finish looking at

the register details, click the Restore button of the view's toolbar to return the view to its previous size. Alternatively, right-click the Registers tab and select Detached. The Registers view becomes a floating window that you can resize. After you finish looking at the register details, right-click the Registers tab of the floating window and select Detached again. You can rearrange the re-attached view by dragging its tab to a different collection of view tabs.

5.3.7.5 Changing bit field

To change a bit field in a register, you must first start a debugging session, then open the Registers view.

To change a bit field:

1. In the Registers view, view register details.
2. Expand the register group that contains the bit field you want to change.
3. Expand the Bit Field and Description groups.

The register details appear ([Figure 5-20](#)) in the Registers view.

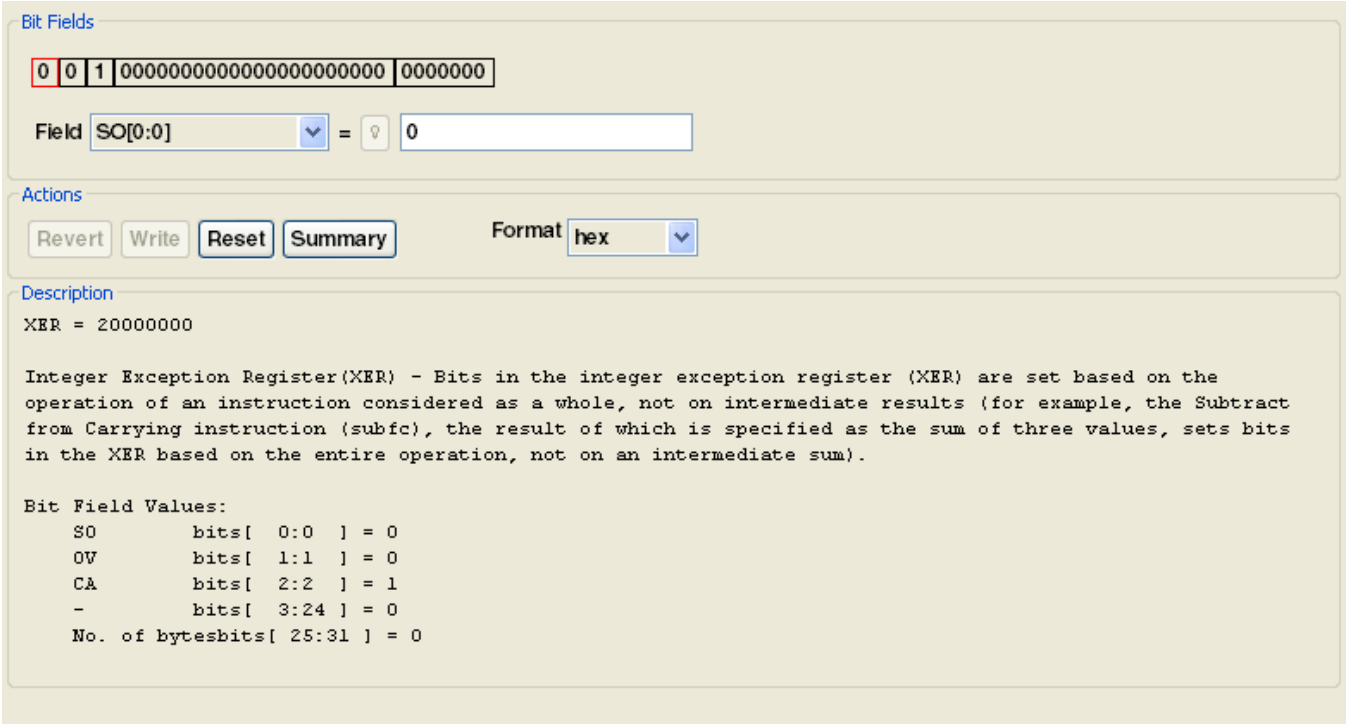


Figure 5-20. Registers view - Register details

- From the expanded register group above the register details, select the name of the register that contains the bit field that you want to change.

The **Bit Fields** group displays a graphical representation of the selected bit field. The **Description** group displays explanatory information about the selected bit field and parent register.

- In the **Bit Fields** group, click the bit field that you want to change. Alternatively, use the **Field** list box to specify the bit field that you want to change.
- In the = text box, type the new value that you want to assign to the bit field.
- In the **Action** group, click Write.

The debugger updates the bit-field value. The bit values in the **Value** column and the Bit Fields group change to reflect your modification.

NOTE

Click Revert to discard your changes and restore the original bit-field value.

5.3.8 Setting watchpoints

A watchpoint is another name for a data breakpoint. The debugger halts execution each time the watchpoint location is read, written, or accessed (read *or* written). The debugger lets you set a watchpoint on an address or range of addresses in memory. You can set the watchpoint from the:

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

Setting the watchpoint type defines the conditions under which the debugger halts execution.

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.

5.3.8.1 Adding watchpoints

Use the Add Watchpoint dialog box to create a watchpoint for a memory range. You can specify these parameters for a watchpoint:

- an address (including memory space)
- an expression that evaluates to an address
- a memory range
- an access type on which to trigger

To open the Add Watchpoint dialog box:

1. Open the Debug perspective.
2. Click one of these tabs:
 - Breakpoints
 - Memory
 - Variables

The corresponding view comes forward.

3. Right-click the appropriate content inside the view. The table below describes the various options available on the **Add Watchpoint** dialog box.

Table 5-12. Opening the Add Watchpoint Dialog Box

In the View...	Right-Click...
Breakpoints	an empty area inside the view.
Memory	the cell or range of cells on which you want to set the watchpoint.
Variables	a global variable. Note that the debugger does not support setting a watchpoint on a stack variable or a register variable.

4. Select Add Watchpoint (C/C++) from the context menu that appears.

The Add Watchpoint dialog box appears ([Figure 5-21](#)). The debugger sets the watchpoint according to the settings that you specify in the Add Watchpoint dialog box. The Breakpoints view shows information about the newly set watchpoint. The Problems view shows error messages when the debugger fails to set the watchpoint.

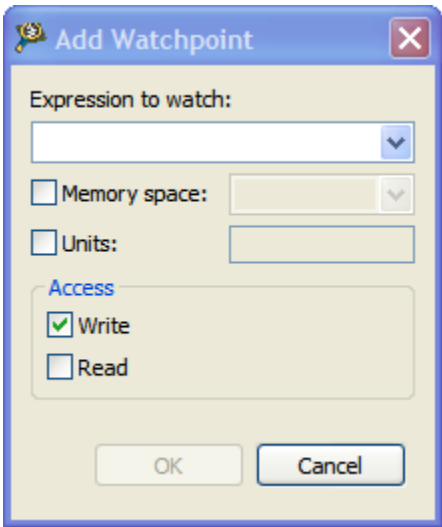


Figure 5-21. Add Watchpoint Dialog Box

[Table 5-13](#) describes the Add Watchpoint dialog box options.

Table 5-13. Add Watchpoint Dialog Box Options

Option	Description
Expression to watch	Enter an expression that evaluates to an address on the target device. The debugger displays an error message when the specified expression evaluates to an invalid address. You can enter these types of expressions:

Table continues on the next page...

Table 5-13. Add Watchpoint Dialog Box Options (continued)

Option	Description
	<ul style="list-style-type: none"> An r-value, such as <code>&variable</code> A register-based expression. Use the <code>\$</code> character to denote register names. For example, enter <code>\$SP-12</code> to have the debugger set a watchpoint on the stack pointer address minus 12 bytes. <p>The Add Watchpoint dialog box does not support entering expressions that evaluate to registers.</p>
Memory space	Check this option to specify an address, including memory space, at which to set the watchpoint. Use the text box to specify the address or address range on which to set the watchpoint. If a debugging session is not active, the text/list box is empty, but you can still type an address or address range.
Units	Enter the number of addressable units that the watchpoint monitors.
Write	Check this option to enable the watchpoint to monitor write activity on the specified memory space and address range. Clear this option if you do not want the watchpoint to monitor write activity.
Read	Check this option to enable the watchpoint to monitor read activity on the specified memory space and address range. Clear this option if you do not want the watchpoint to monitor read activity.

5.3.8.2 Removing watchpoints

To remove a watchpoint:

1. Open the Breakpoints view if it is not already open by choosing Window > Show View > Breakpoints.

The Breakpoints view appears, displaying a list of watchpoints.

2. Right-click on the watchpoint you wish to remove and pick Remove from the menu that appears.

The selected watchpoint is removed, and it disappears from the list in the Breakpoints view.

5.3.9 Setting breakpoints

The different breakpoint types that you can set are listed below:

- Software

The debugger sets a software breakpoint into target memory. When program execution reaches the breakpoint, the processor stops and activates the debugger. The breakpoint remains in the target memory until the user removes it.

The breakpoint can only be set in writable memory like SRAM or DDR. You cannot use this type of breakpoints in ROM.

- Hardware

Selecting the Hardware menu option causes the debugger to use the internal processor breakpoints. These breakpoints are usually very few and can be used with all types of memories (ROM/RAM) because they are implemented by using processor registers.

Tip

You can also set breakpoint types by issuing the `bp` command in the CodeWarrior Debugger Shell.

To set a breakpoint:

1. In the IDE's Debug perspective, click the Debug tab.

The Debug view appears (Figure 5-22).

2. Right clicking on a code line will set a breakpoint.

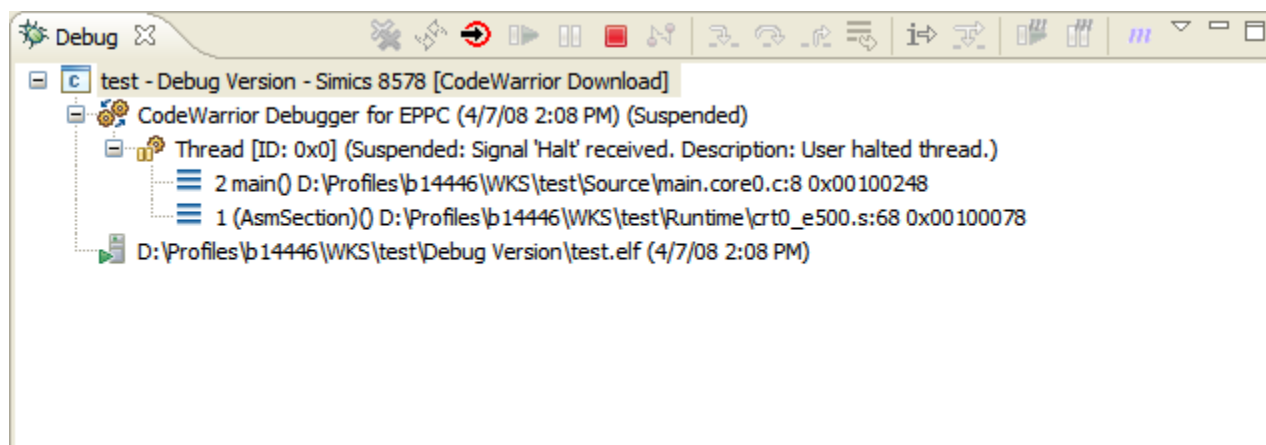


Figure 5-22. Debug view

3. Expand the Thread group.
4. Under the Thread group, select the thread that has the `main()` function.

The source code appears in an editor view (Figure 5-23). The small blue arrow to the left of the source code indicates which code statement the processor's program counter is set to execute next.

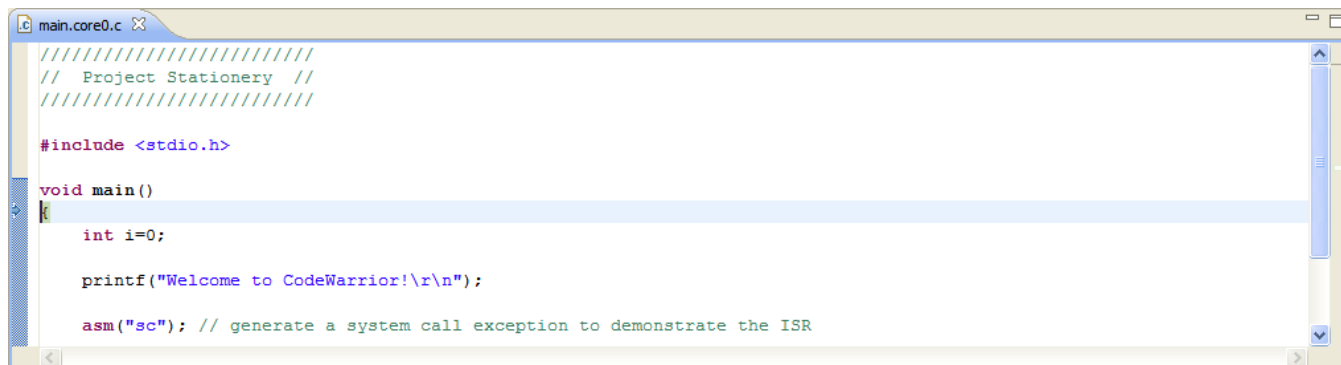


Figure 5-23. Editor view

5. In the editor view, place the cursor on the line that has this statement: `printf("Welcome to CodeWarrior!\r\n");`
6. Select Run > Toggle Line Breakpoint.
7. A blue dot appears in the marker bar to the left of the line (Figure 5-24). This dot indicates an enabled breakpoint. After the debugger installs the breakpoint, a blue checkmark appears beside the dot. The debugger installs a breakpoint by loading into the Java™ virtual machine the code in which you set that breakpoint.

Tip

An alternate way to set a breakpoint is to double-click the marker bar to the left of any source-code line. If you set the breakpoint on a line that does not have an executable statement, the debugger moves the breakpoint to the closest subsequent line that has an executable statement. The marker bar shows the installed breakpoint location. If you want to set a hardware breakpoint instead of a software breakpoint, use the `bp` command in the Debugger Shell. You can also right-click on the marker bar to the left of any source-code line, and select Set Special Breakpoint from the context menu that appears.

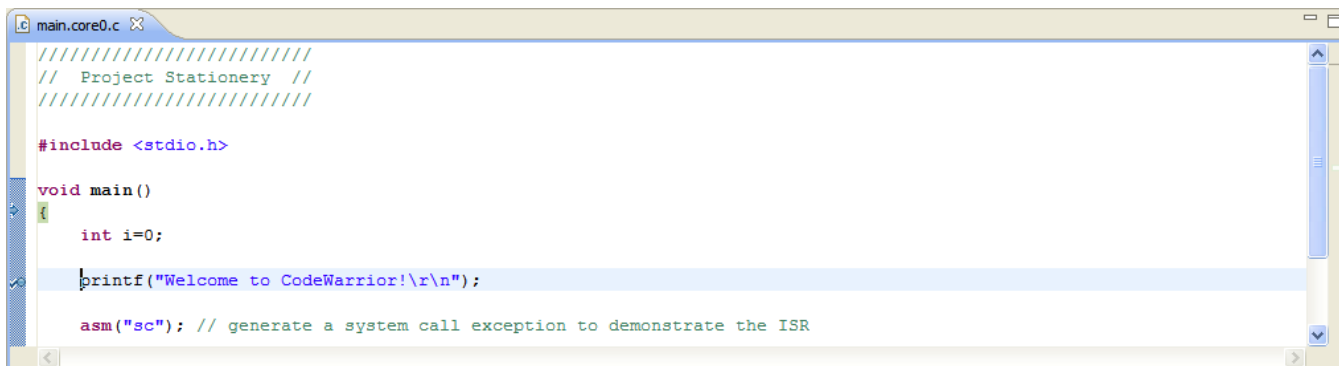


Figure 5-24. Editor view after setting breakpoints

8. From the menu bar, select Run > Resume.

The debugger executes all lines up to, but not including, the line at which you set the breakpoint. The editor view highlights the line at which the debugger suspended execution (Figure 5-25). Note also that the program counter (blue arrow) is positioned here.

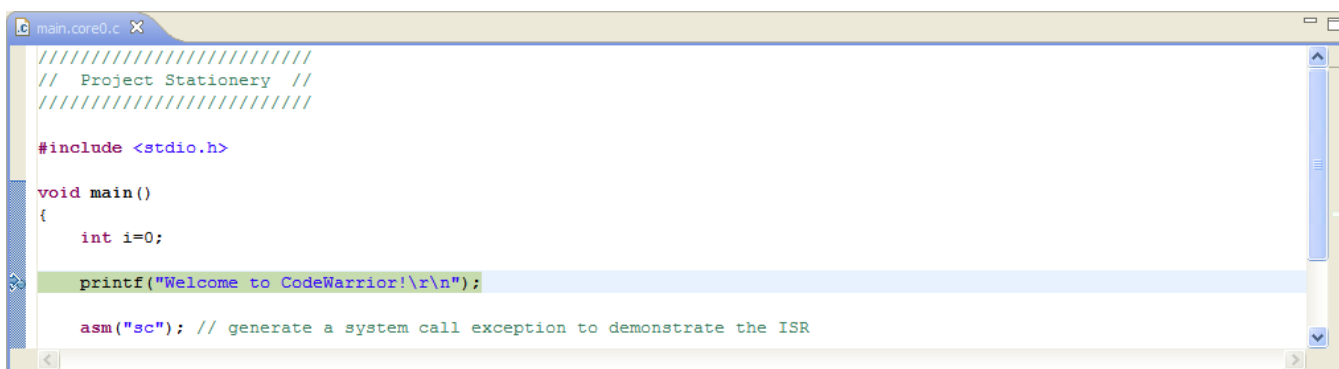


Figure 5-25. Editor view after reaching breakpoint

5.3.9.1 AIOP task specific breakpoints

AIOP is intended to be a programmable highly efficient engine targeted at packet processing applications. The fundamental unit of operation in an AIOP instance is the task. Tasks are created and terminated by the hardware. A finite number of tasks can exist and execute simultaneously inside an AIOP. The life cycle of an AIOP task is usually very short, and at any given point of time, on a core, we can have tasks from different sources and different networking protocols. AIOP specific breakpoints provide a mechanism for debugging only the selected tasks, without affecting other tasks.

The table below lists the AIOP task specific breakpoint variants supported by the current CodeWarrior release.

Table 5-14. AIOP task specific breakpoint variants

Type	Scope	Effect	Description
AIOP, Any Task, Global Halt, Software	Any Task	Global Halt	Any task may hit the breakpoint and when it does it triggers the AIOP global halt.
AIOP, Any Task, Global Halt, Hardware	Any Task	Global Halt	Any task may hit the breakpoint and when it does it triggers the AIOP global halt. This method uses a hardware breakpoint.
AIOP, Any Task, Task Halt, Software	Any Task	Task Halt	Any task may hit the breakpoint and when it does it is suspended from execution.
AIOP, One Task, Global Halt, Software	Current Task	Global Halt	Only current task may hit the breakpoint and when it does it triggers the AIOP global.
AIOP, One Task, Task Halt, Software	Current Task	Task Halt	Only current task may hit the breakpoint and when it does it is placed in debug inhibited for scheduling.

5.3.9.2 Setting hardware breakpoints

There are two ways to set hardware breakpoints:

- [Setting hardware breakpoint using editor view](#)
- [Setting hardware breakpoint using debugger shell](#)

5.3.9.2.1 Setting hardware breakpoint using editor view

In either the C/C++ perspective or the Debug perspective, select the source line in the Editor view where you want to place the breakpoint. Go to the marker bar on the left side of the Editor view. Right-click on it to display a menu. Choose Set Special Breakpoint > Hardware to set a hardware breakpoint.

5.3.9.2.2 Setting hardware breakpoint using debugger shell

To set hardware breakpoints using the Debugger Shell:

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.

5.3.9.3 Removing breakpoints

To remove a breakpoint from your program, you have two options:

- [Removing breakpoint using marker bar](#)
- [Removing breakpoint using Breakpoints view](#)

NOTE

For more information on removing hardware breakpoints, see [Removing hardware breakpoints](#).

5.3.9.3.1 Removing breakpoint using marker bar

To remove an existing breakpoint using the marker bar:

1. Right-click on the existing breakpoint in the marker bar.
2. Select Toggle Breakpoint from the menu that appears.

5.3.9.3.2 Removing breakpoint using Breakpoints view

To remove an existing breakpoint using the Breakpoints view:

1. Open the Breakpoints view if it is not already open by choosing Window > Show View > Breakpoints.

The Breakpoint view appears, displaying a list of breakpoints.

2. Right-click on the breakpoint you wish to remove and pick Remove from the menu that appears.

The selected breakpoint is removed, and it disappears from the both the marker bar and the list in the view.

NOTE

To remove all of the breakpoints from the program at once, select Remove All from the menu.

5.3.9.4 Removing hardware breakpoints

There are two ways to remove existing hardware breakpoints:

- [Removing hardware breakpoint using editor view](#)
- [Removing hardware breakpoint using Debugger Shell.](#)

5.3.9.4.1 Removing hardware breakpoint using editor view

To remove a hardware breakpoint using the editor view:

1. Right-click on the existing breakpoint in the marker bar.
2. Select Toggle Breakpoint from the menu that appears.

Alternatively, to remove the breakpoint from the Breakpoint view:

1. Open the Breakpoints view if it is not already open by choosing Window > Show View > Breakpoints.

The Breakpoint view appears, displaying a list of breakpoints.

2. Right-click on the hardware breakpoint you wish to remove and pick Remove from the menu that appears.

The selected breakpoint is removed, and it disappears from the both the marker bar and the list in the view.

5.3.9.4.2 Removing hardware breakpoint using Debugger Shell

To remove a hardware breakpoint using the Debugger Shell:

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 remove the hardware breakpoint.

For example, to remove a breakpoint at line 6 in your program, type:

```
bp -hw 6 off
```

4. Press the Enter key.

The debugger shell executes the command and removes the hardware breakpoint.

5.3.10 Setting stack depth

Select **Window > Preferences > C/C++ > Debug > Maximum Stack crawl depth** command to set the depth of the stack to read and display. Showing all levels of calls when you are examining function calls several levels deep can sometimes make stepping through code more time-consuming. Therefore, you can use this menu option to reduce the depth of calls that the debugger displays.

5.3.11 Changing program counter value

To change the program-counter value:

1. Start a debugging session.
2. In the Editor view, place the cursor on the line that you want the debugger to execute next.
3. Right-click in the Editor view.

A context menu appears.

4. From the context menu, select **Move To Line**.

CodeWarrior IDE modifies the program counter to the specified location. The Editor view shows the new location.


5.3.12 Hard resetting

Use the reset hard command in the Debugger Shell to send a hard reset signal to the target processor.

NOTE

The Hard Reset command is enabled only if the debug hardware you are using supports it.

Tip

You can also perform a hard reset by clicking the Multicore Reset () button from the Debug perspective toolbar.

5.3.13 Loading and saving memory

The Load/Save Memory command:

- Loads the specified amount of data from a binary or S-Record file on the host and writes this data to the target board's memory starting at the specified address.
- Reads the specified amount of data from the specified address of the target board's memory and saves this data in a binary file on the host.

To load/save memory:

1. Select Window > Show View > Other

The Show View dialog box appears.

2. From the **Debug** group, select **Target Tasks**.

The Target Tasks view appears ([Figure 5-26](#)) at the bottom-right of the IDE window.

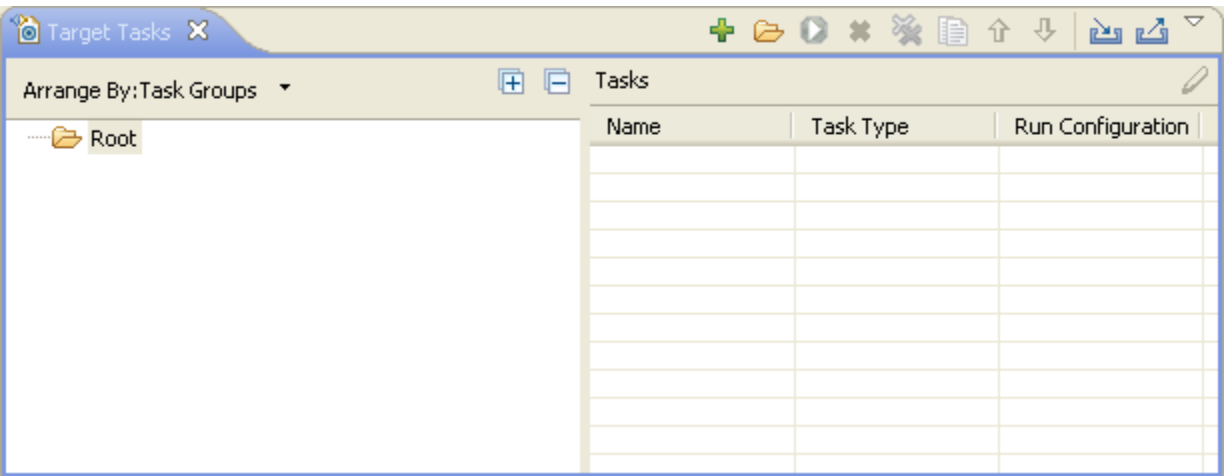


Figure 5-26. Target Tasks view

3. Right-click in the Target Tasks view and select New Task from the context menu that appears.

The Create New Target Task dialog box appears ([Figure 5-27](#)).

4. From the Task Type drop-down list box, select Import/Export/Fill Memory.

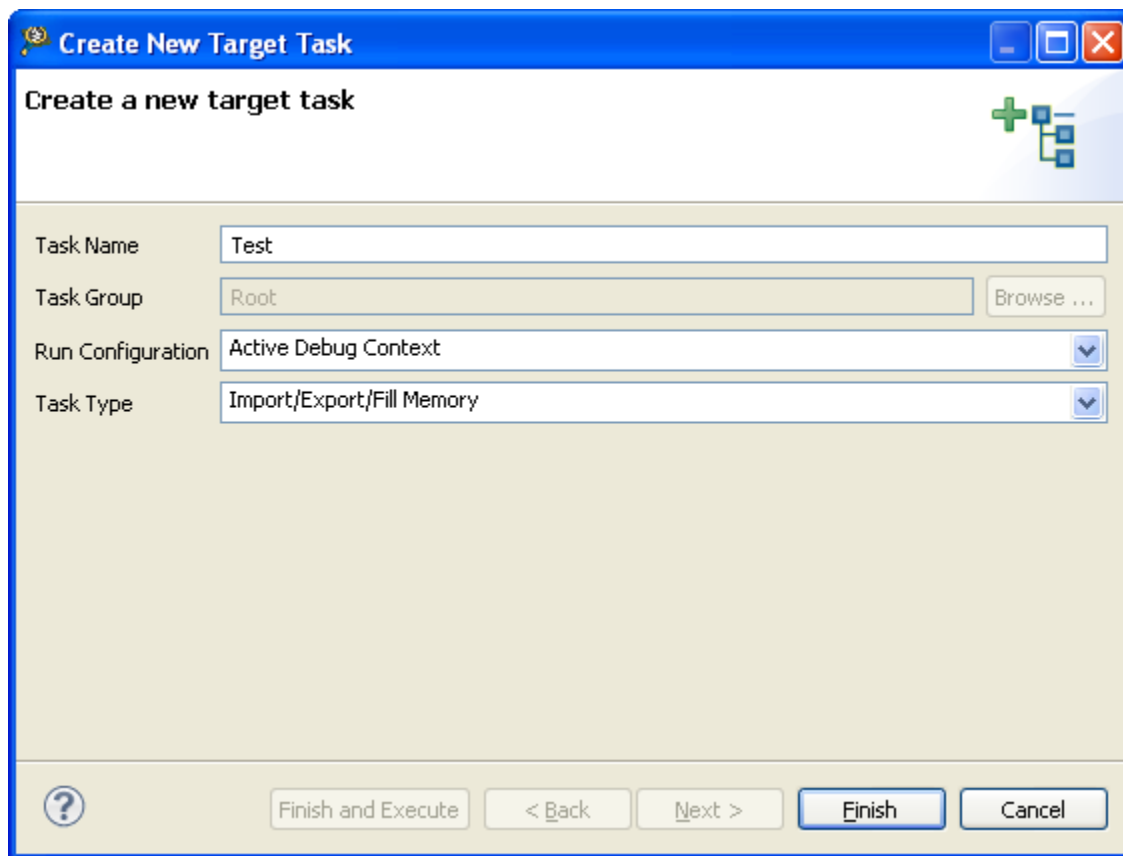


Figure 5-27. Create New Target Task dialog

5. Click Finish.

The new task appears in the Tasks panel.

6. Select the Fill Memory with a data pattern option from this window to fill memory.

Tip

You can also load and save memory by issuing the `restore` and `save` commands in the CodeWarrior Debugger Shell.

If you load an S-Record file, the loader behaves as follows:

- The loader uses the offset field to shift the address contained in each S-Record to a lower or higher address. The sign of the offset field determines the direction of the shift.

- The address produced by this shift is the memory address at which the loader starts writing the S-Record data.
- The loader uses the address and size fields as a filter. The loader applies these fields to the initial S-Record (not to its shifted version) to make sure that only the zone defined by these fields is actually written to.

5.3.14 Filling memory

Use this command to fill a particular memory location with data of particular size and type. This command lets you write a set of characters to a particular memory location on the target by repeatedly copying the characters until the specified fill size has been reached.

NOTE

For more information, see *Import/Export/Fill Memory* section in the *Debugger* chapter of the *CodeWarrior Common Features Guide* available in the `<CWInstallDir>\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.

5.3.15 Controlling cross triggering

The AIOP core debugging can be performed by enabling or disabling the debug halt cross-triggering. You can configure the cross-triggering using the **Disable Halt Groups** button under **Multicore Groups** from the **Debug** view.

To enable or disable the cross-triggering between AIOP cores:

1. Open the **Debug** view.
 - a. Select **Window > Show View > Other**.

The **Show View** dialog appears.

- b. Select **Debug** from the **Debug** group.
- c. Click **OK**.

The **Debug** view appears.

- d. Open the **Multicore Groups** drop-down menu from the toolbar menu.
- e. Select to check/clear the **Disable Halt Groups**. [Figure 5-28](#) shows **Debug Halt Group** option.

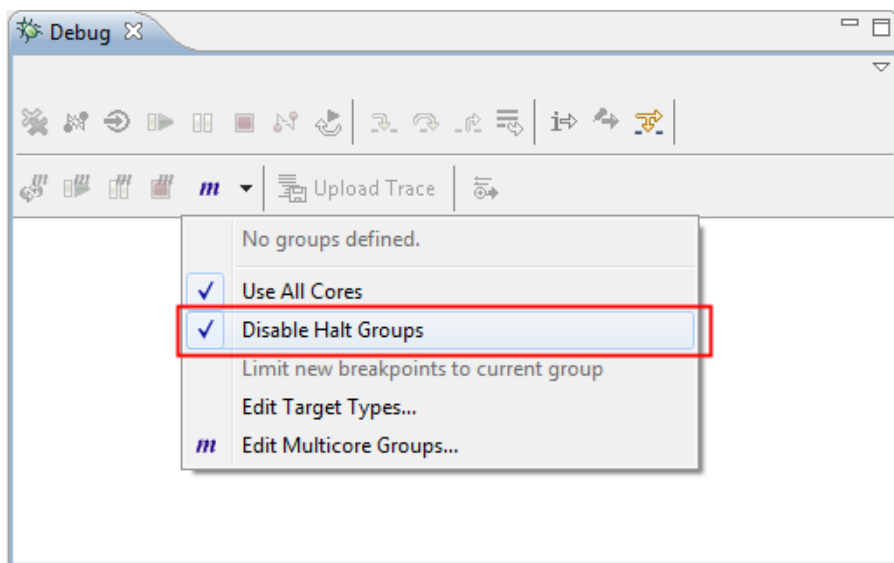


Figure 5-28. Debug view - Debug Halt Groups option

NOTE

In case you have selected **AIOP** in the **OS Awareness** tab, the cross-triggering will be enabled by default while performing task-debugging.

The CodeWarrior IDE disables or enables the cross-triggering between AIOP cores.

5.4 CodeWarrior Executable Importer wizard

You can use the CodeWarrior debugger to debug an executable (.elf) file that has no associated CodeWarrior project using the CodeWarrior Executable Importer wizard.

To use the **Import a CodeWarrior Executable file** wizard:

1. Start the CodeWarrior IDE.
2. From the main menu bar, select **File > Import**.

The Import wizard launches and the **Select** page appears.

3. Expand the **CodeWarrior** group.
4. Select the **CodeWarrior Executable Importer** option to debug a executable .elf file.
5. Click **Next**.

The wizard name changes to **Import a CodeWarrior Executable file** and the **Import a CodeWarrior Executable file** page appears. For more information, see [Import a CodeWarrior Executable file page](#).

This section describes the various pages that the wizard displays as it assists you in debugging an executable (.elf) file.

- [Import a CodeWarrior Executable file page](#)
- [Import C/C++/Assembler Executable Files page](#)
- [Processor page](#)
- [Debug Target Settings page](#)
- [Configurations page](#)

5.4.1 Import a CodeWarrior Executable file page

Use this page to select an executable file or a folder to search for the executable files.

The table below describes the options available on the page.

Table 5-15. Import a CodeWarrior Executable file page settings

Option	Description
Project name	Specify the name of the project. The specified name identifies the project created for debugging (but not building) the executable file.
Use default location	Check this option to store the project in the default location. Clear this option to specify the location for the imported project.
Location	Specifies the location to save the imported project file. Click Browse to navigate to the specified folder location. Ensure that you append the path with the name of the project to create a new location for your project.

5.4.2 Import C/C++/Assembler Executable Files page

Use this page to select the elf file to import.

The table below lists the options available on the page.

Table 5-16. Import C/C++/Assembler Executable page settings

Option	Description
File to import	Specifies the C/C++/Assembler executable file. Click Browse to choose an executable file.
Copy the selected file to the current project folder	Check this option to copy the selected executable file to an already existing project folder.

5.4.3 Processor page

Use this page to specify the processor family for the imported executable file and specify the toolchain to be used.

The table below describes the options available on the page.

Table 5-17. Processor page settings

Option	Description
Processor	Expand the processor family and select the specific target from this pull-down menu. The toolchain uses this choice to generate code that makes use of processor-specific features, such as multiple cores. Note: You can also use type filter text box as a shortcut to specify a processor. Start typing the processor name into the text box.
Toolchain	Chooses the compiler, linker, and libraries used to build the program. Each toolchain generates code targeted for a specific platform. These are: <ul style="list-style-type: none"> Bareboard Application-Targets a hardware board without an operating system.
Target OS	Specify the operating system that runs on the target device. This option is applicable only for Bareboard Application projects.

5.4.4 Debug Target Settings page

This page displays the debugger connection types supported by the current installation. Use this page to specify connection type and launch configurations created for the new project.

The table below describes the options available on the page.

Table 5-18. Debug Target Settings page settings

Option	Description
Debugger Connection Types	Specifies what target the program executes on. <ul style="list-style-type: none"> Simulator - Select to execute the program on a software simulator.
Board	Specifies the hardware supported by the selected processor.
Launch	Specifies the launch configurations and corresponding connection, supported by the selected processor.
Connection Type	Specifies the interface to communicate with the hardware.
TAP address	Specify the IP address of the selected TAP device.
Simulator remote IP	Specify the IP address of the remote Linux 64-bit machine, the simulator is started on.
Simulator port number	Specify the port number that the debugger will use to communicate with the simulator launched on the remote Linux host.

NOTE

The **Debug Target Settings** page may prompt you to either create a new remote system configuration or select an existing one.

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. For more information, refer to the *CodeWarrior Development Studio Common Features Guide* available in the `<CodeWarrior-Install-Dir>\LS\Help\PDF\` folder.

5.4.5 Configurations page

Use this page to select the processor core that executes the project.

The table below lists the options available on the page.

Table 5-19. Configurations page

Options	Description
Core Index	Select the processor core that executes the project.

5.5 Debugging externally built executable file

You can use the **CodeWarrior Executable Importer** wizard to debug an executable (.elf) file that has no associated CodeWarrior project. For example, you can debug an .elf file that a different IDE generated. To debug these externally built executable files:

1. **Specifying the Executable File** - Specify the externally built executable file that you want to debug in the CodeWarrior IDE. The IDE imports the executable file into a new project.

To specify the executable file:

- a. Launch the CodeWarrior IDE.
- b. From the main menu bar, select **File > Import**.

The Import wizard appears.

- c. Expand the **CodeWarrior** group.
- d. Select **CodeWarrior Executable Importer**.
- e. Click Next.

The Import a CodeWarrior Executable file page appears.

- f. In the **Project Name** text box specify a name for the imported project.
- g. Click Next.

The Import C/C++/Assembler Executable files page appears.

- h. Click Browse to select the elf file to import.
- i. The Select file dialog box appears.
- j. Use the dialog box to navigate to the executable file that you want to debug.
- k. Click Open.

The Select file dialog box closes. The path to the executable file appears in the **File to Import** text box.

Tip

You can also drag and drop an elf file in the CodeWarrior Eclipse IDE. When you drop the .elf file in the IDE, the **Import a CodeWarrior Executable file** wizard appears with the .elf file already specified in the **Project Name** and **File to Import** text box.

2. **Creating a Project for the Executable File** - Create a new CodeWarrior project that you use to debug the executable file. Use the project's associated launch configuration to specify debugging parameters for the executable file.

If you want to use an already existing project work space for the imported executable file, check the Copy the selected file to current project folder check box.

- a. Click Next.

The Processor page appears.

- b. Select the processor family for the executable file.
- c. Select a toolchain from the **Toolchain** group.

Selected toolchain sets up the default compiler, linker, and libraries used to build the new project. Each toolchain generates code targeted for a specific platform.

- d. Select if the board runs no operation system or imports a linux kernel project to be executed on the board. The **Target OS** options are applicable only for Bareboard Application projects.
- e. Click Next.

The **Debug Target Settings** page appears.

- f. Select a supported connection type, from the Debugger Connection Types group. Your selection determines the launch configurations that you can include in your project.
- g. Select the hardware or simulator, you plan to use, from the **Board** drop-down list.

NOTE

Hardware or Simulators that supports the target processor selected on the **Processors** page are only available for selection.

- h. Select the launch configurations, that you want to include in your project and the corresponding connection.
- i. Select the interface to communicate with the hardware, from the **Connection Type** drop-down list.
- j. Enter the IP address of the TAP device in the **TAP address** text box. This option is disabled and cannot be edited, if you select **USB TAP** from the **Connection Type** drop-down list.
- k. Click Next.

The Configurations page appears.

- l. Select the processor core that executes the project, from the Core index list.

- m. Click Finish.

The **Import a CodeWarrior Executable file** wizard ends. The project for the imported `.elf` file appears in the CodeWarrior Projects view. You can now open the Debug Configurations window by selecting Run > Debug Configurations. The Debug Configurations window shows the current settings for the launch configuration that you just created. A remote system is created with details of all the connection, initialization, and target parameters you had set while importing the elf file.

3. **Editing the Launch Configuration** - Modify the default settings of the launch configuration. Specify the appropriate debugger, target processor, initialization files, connection protocol, and other debugger-related options for the executable file.

If you want to change the settings for the launch configuration from what you had set during importing the elf file, you can do it through the various tabs available in the Debug Configurations window.

To edit the launch configuration for your executable file:

- a. On the Main tab, click Edit in the Connection panel.

The corresponding Connection page appears.

- b. Use the Connection type list box to modify the current connection type.
- c. Configure the various connection options as appropriate for your executable file by using the various tabs available on the Connection page.

For example, specify the appropriate target processor, any initialization files, and connection protocol.

- d. Click **OK** to close the **Connection** page.

NOTE

For more information on how to modify settings using the remote system explorer, refer to the CodeWarrior Common Features Guide from the <CodeWarrior-Install-Dir>\LS\Help\PDF\ folder.

4. **Specifying a Source Lookup Path** - Specify the source lookup path in terms of the compilation path and the local file-system path. 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.

Before you specify a source lookup path, make sure that you edit the launch configuration for the executable file.

To specify a source lookup path for your executable file:

- a. Click the Source tab of the Debug Configurations window.

The corresponding page appears.

- b. Click Add.

The Add Source dialog box appears.

- c. Select Path Mapping from the available list of sources.
- d. Click OK.

The Add Source dialog box closes. The Path Mappings dialog box appears.

- e. In the Name text box, enter the name of the new path mapping.
- f. Click Add.

The cursor blinks in the Compilation path column.

- g. In the Compilation path column, enter the path to the parent project of the executable file, relative to the computer that generated the file.

Suppose 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 column.

- h. In the Local file system path text box, enter the path to the parent project of the executable file, relative to your computer. Click the ellipsis 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.

- i. Click OK.

The Path Mapping dialog box closes. The mapping information now appears under the path mapping shown in the **Source Lookup Path** list of the Source page.

- j. 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.

- k. Click Apply.

The IDE saves your changes.

5. **Debugging the Executable File**-Use the CodeWarrior debugger to debug the externally built executable file. To debug the executable file, click the Debug button of the Debug Configurations window.

5.6 Multi-core operations

This sections explains the various features available to you when debugging a multi-core processor.

- [Multi-core operations in IDE](#)
- [Multi-core operations in Debugger Shell](#)

5.6.1 Multi-core operations in IDE

When you start a multi-core debug session, multi-core commands are enabled on the IDE's Run menu. These commands, when issued, affect all cores simultaneously. [Table 5-20](#) describes each menu choice. For more information on these commands, refer to the Debugger chapter in CodeWarrior Common Features Guide from the <CodeWarrior-Install-Dir>\LS\Help\PDF\ folder.

Table 5-20. Multi-Core Commands

Command	Description
Multicore Resume	Starts all cores of a multi-core system running simultaneously.
Multicore Suspend	Stops execution of all cores of a multi-core system simultaneously.
Multicore Restart	Restarts all the debug sessions for all cores of a multi-core system.
Multicore Terminate	Kills all the debug sessions for all cores of a multi-core system.

To use the multi-core commands from the Debugger perspective:

1. Start a debugging session by selecting the appropriately configured launch configurations.
2. Click the Debug tab of the Debug perspective.
3. If necessary, expand the desired core's list of active threads by clicking on the tree control.
4. Click on the thread you want to use with multi-core operations.

NOTE

Selecting a thread uses the Multicore Group the core is part of in the multicore operation. For more information on the Multicore Groups feature, refer to the CodeWarrior Common Features Guide from the <CodeWarrior-Install-Dir>\LS\Help\PDF\ folder.

5. From the Run menu, specify the multi-core operation to perform on the thread.

NOTE

The keyboard shortcut for the Multi-core Resume operation is Shift-F8.

5.6.2 Multi-core operations in Debugger Shell

In addition to the multicore-specific toolbar buttons and menu commands available in the Debugger view, the Debugger Shell has multi-core specific commands that can control the operation of one or more processor cores at the same time. Like the menu commands, the multi-core Debugger Shell commands allow you to select, start, and stop a specific core. You can also restart or kill sessions executing on a particular core. [Table 5-21](#) lists the commands and their purpose.

Table 5-21. Multi-core Commands for the Debugger Shell

Category	Multi-core Operation	Analogous Single-core Operation
run control	mc::go	go
	mc::stop	stop
session control	mc::restart	restart
	mc::kill	kill

NOTE

For more information on Debugger Shell, see *CodeWarrior Common Features Guide* available in the `<CWInstallDir>\LS\Help\PDF\` folder, where `<CWInstallDir>` is the path where you have installed your CodeWarrior software.



Chapter 6

Working with hardware tools

This chapter explains how to use the CodeWarrior hardware tools. Use these tools for board bring-up, test, and analysis.

This chapter explains:

- [Working with hardware diagnostics](#)
- [Manipulating target memory](#)

6.1 Working with hardware diagnostics

Use the hardware-diagnostic tools of the CodeWarrior IDE to run one or more generic memory operations on a target device. For example, you can run an individual read or write operation to or from a memory location. Also, you can run a number of algorithms based access operations on a chunk of memory. After you run the memory operations, you can observe whether the operation succeeded and view log information for additional details.

Following sections will help you with more details on working with hardware-diagnostic tools:

- [Creating new hardware diagnostics task](#)
- [Executing hardware diagnostics task](#)
- [Editing hardware diagnostics task](#)
- [Hardware Diagnostics Action editor](#)

6.1.1 Creating new hardware diagnostics task

In order to run a hardware-diagnostic or memory operation, you must first open the Target Tasks view. To open the view, follow these steps:

1. Select Window > Show View > Other.

The Show View dialog box appears.

2. Expand the Debug group.
3. Select Target Tasks.
4. Click OK.

The Target Tasks view appears in the Debug perspective.

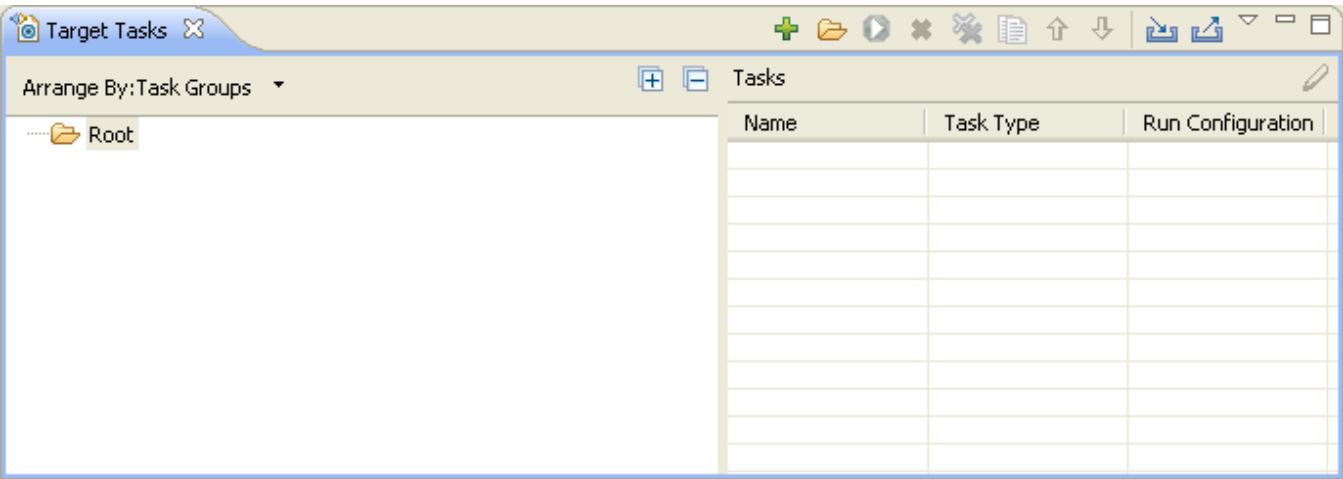


Figure 6-1. Target Task View

5. 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 dialog box appears.

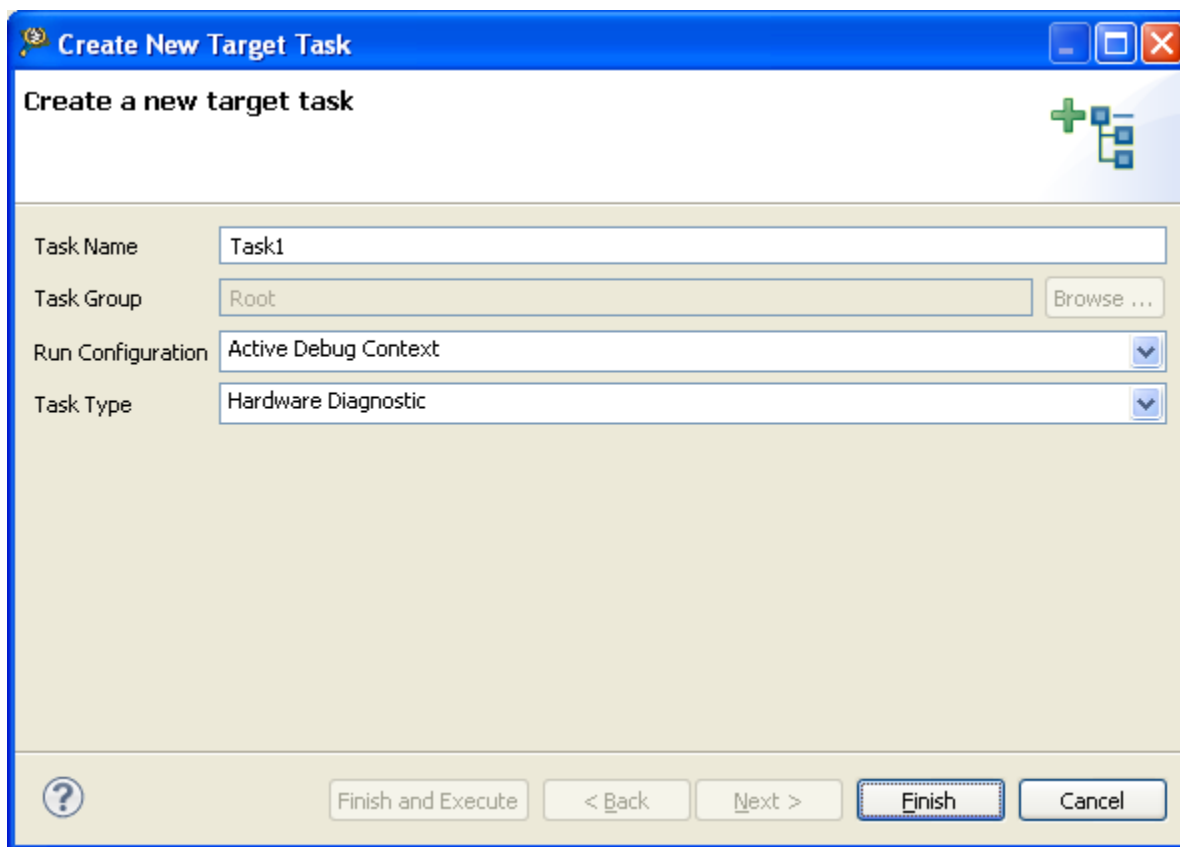


Figure 6-2. Create New Target Task Dialog Box

6. Specify a name to the new task in the Target Name text box.
7. Specify the configuration that the task launches and uses to connect to the target from the Run Configuration drop-down list.
8. Select Hardware Diagnostic from the Task Type list box.

NOTE

Select Active Debug Context from the Run Configuration drop-down list, if you want to use flash programmer over an active debugger session, else select any of the specified debug context from the list.

The Hardware Diagnostics users can choose a Download or a Connect debug context from the **Run Configuration** drop-down list because the Active Debug Context will not be valid in this case as the licenses to use the same are limited. The debug contexts are automatically created when creating a new project using the New Project Wizard. It is mandatory to select a debug context.

9. Click Finish.

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

6.1.2 Executing hardware diagnostics task

To execute a pre-defined hardware diagnostic task, follow these steps:

1. Open the Target Tasks view.
2. Click **Arrange By: <Type>** to display the appropriate list of tasks.

The Target Tasks View shows display options in the Debug perspective.

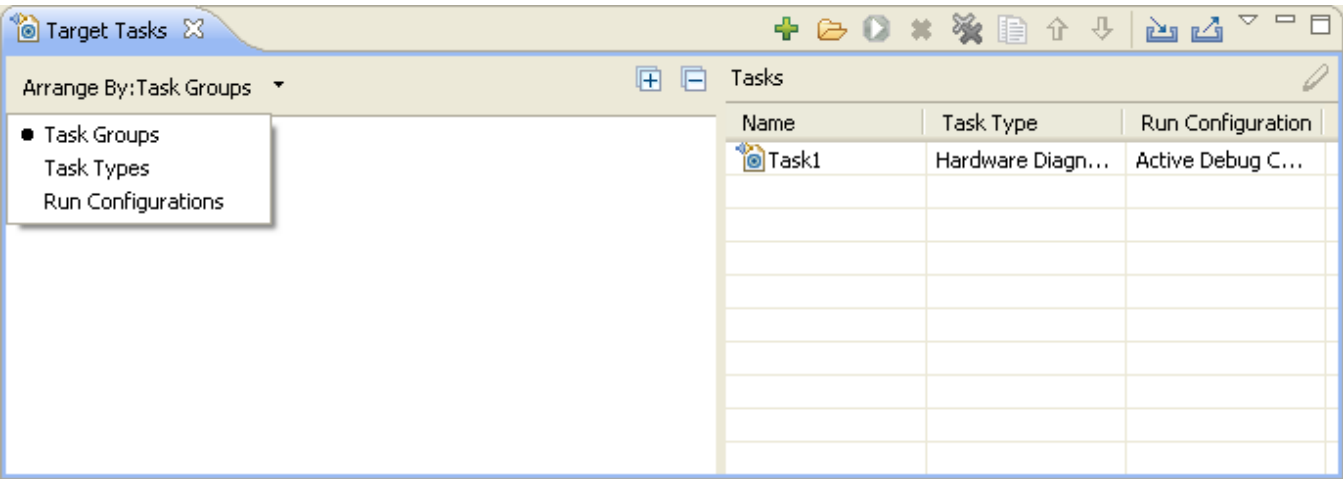


Figure 6-3. Target Task View - Display Options

3. Right-click the task and select **Execute** from the context menu. Alternatively, click the **Execute** icon on the **Target Task** view toolbar to execute the selected task.

The hardware diagnostic task is executed. The Executing Target Task dialog box displays the progress of execution of the task. Upon completion, the Hardware Diagnostics action editor for setting the task specific configuration settings is displayed.

6.1.3 Editing hardware diagnostics task

To edit a Hardware Diagnostics task, follow these steps:

1. Double click a hardware-diagnostic task in the Target Tasks view to edit it.
Alternatively, right-click on the task and select Edit Task Configuration from the context menu.

The Hardware Diagnostics Action editor appears.

Figure 6-4. Hardware Diagnostics Action Editor

2. Select the type of the task you want to perform from the Action Type. You may perform the following hardware diagnostic tasks:
 - **Memory read/write:** Performs a read/write action on the target of the selected size to the selected address. Selecting this, enables the Memory Access group.
 - **Scope loop:** Performs read or write action of the selected size at the selected address unit until you stop the task. (Press Cancel in the Task Monitor to stop the Task). Selecting this, enables the Memory Access and Loop Speed group.
 - **Memory Test:** Performs a number of tests on the target at the selected memory range: Memory starting from Target Address and of a size equal to Test area size. These tests are Walking 1's, Bus Noise, and Address. These can be

executed number of times equal to Number of Passes. Selecting this task enables the Access Size and Target Address from the Memory Access group and the Memory Tests group.

6.1.4 Hardware Diagnostics Action editor

The table below lists and defines all the User Interface (UI) options present within Hardware Diagnostics Action editor.

Table 6-1. Hardware Diagnostics Action Editor - Options

Action Type	Description
Target Address	The address on the target that we want to test. It is used in all 3 of test types
Access Type	Specify the access type to test: reading or writing to memory.
Value	Specify a value to write to the Target Address
Access Size	Specify the number of bytes which are going to be read or written by one access to the memory: 1 Byte, 2 Byte, 4 Byte
Loop Speed	Specify the time between two consecutive memory accesses
Test Area Size	Specify the size of memory to test. This setting along with Target Address defines the memory range to test.
Number of passes	Specify the number of times a test is to be executed.
Tests to Run	Specify the tests to be executed: Walking 1's, Bus Noise, Address
Use Target CPU	Check: Do the tests directly on the target by downloading a test driver to the memory specified by Download algorithm to address preference. Unchecked: Do the tests by accessing the memory from the host by means of the protocol.
Download Algorithm to Address	Specify the address where the test driver is downloaded in case the Use Target CPU activated.

6.2 Manipulating target memory

You can manipulate the target's memory by exporting memory contents to a file or importing data from a file into the memory. The **Import/Export/Fill Memory** utility also lets you fill a specified memory range with user provided data pattern.

Following section will help you with more details on manipulating target memory:

- [Creating target task to manipulate memory](#)

- [Editing import/export/fill memory task](#)
- [Import/Export/Fill Memory Action editor](#)

6.2.1 Creating target task to manipulate memory

Follow these steps to create a target task to manipulate memory:

1. Select Window > Show View > Other.

The Show View dialog box appears.

2. From the Debug group, select Target Tasks.

The Target Tasks view appears.

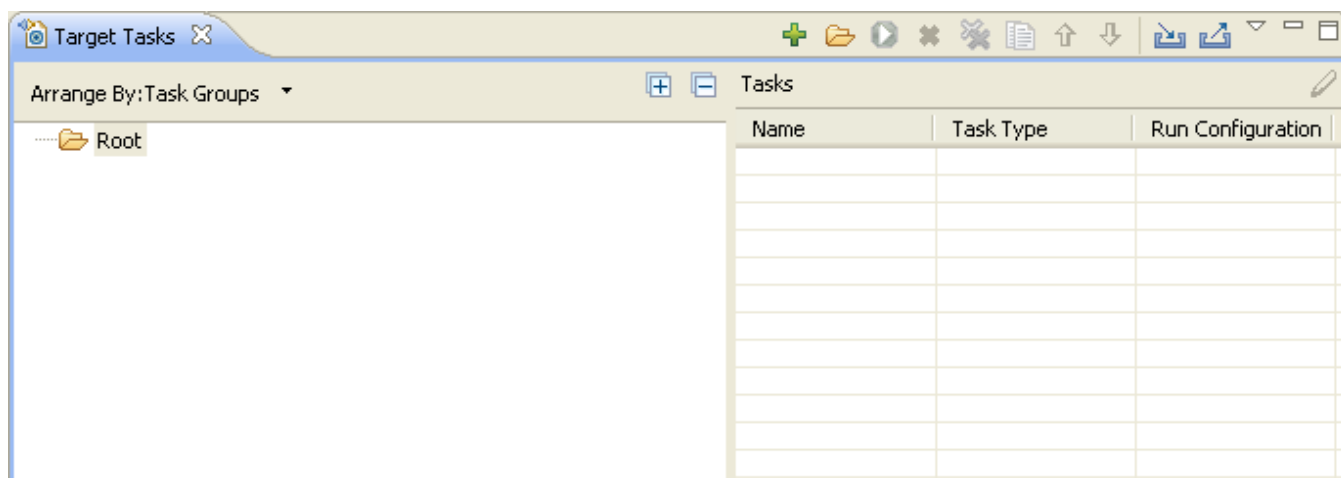


Figure 6-5. Target Task View

3. 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 dialog box appears.

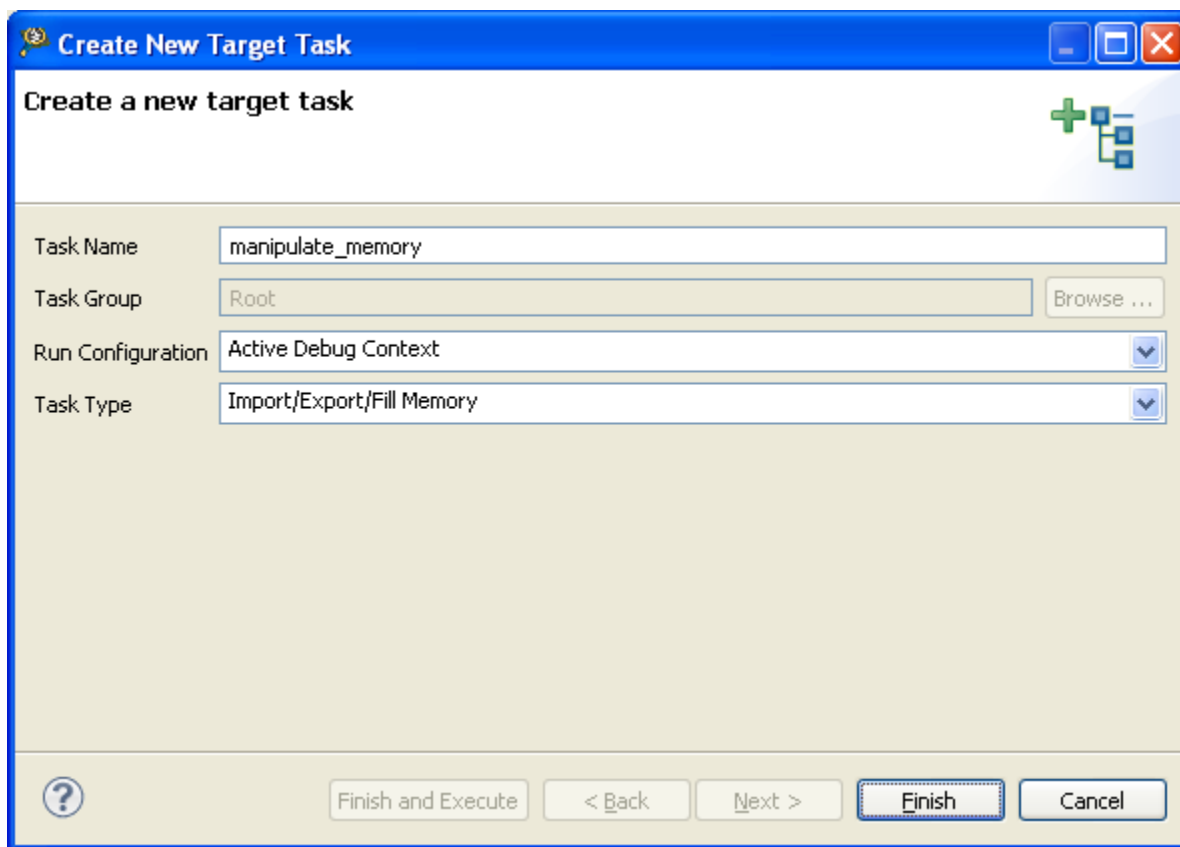


Figure 6-6. Create New Target Task Dialog Box

4. Specify a name to the new task in the Target Name text box.
5. Specify the configuration that the task launches and uses to connect to the target from the Run Configuration drop-down list.
6. Select Import/Export/Fill Memory from the Task Type list box.
7. Click **Finish**.

A new memory task is created in the Target Tasks view.

6.2.2 Editing import/export/fill memory task

To edit a memory task, follow these steps:

1. Double click a memory task in the Target Tasks view to edit it. Alternatively, right-click on the task and select Edit Task Configuration from the context menu.

The Import/Export/Fill Memory Action editor appears.

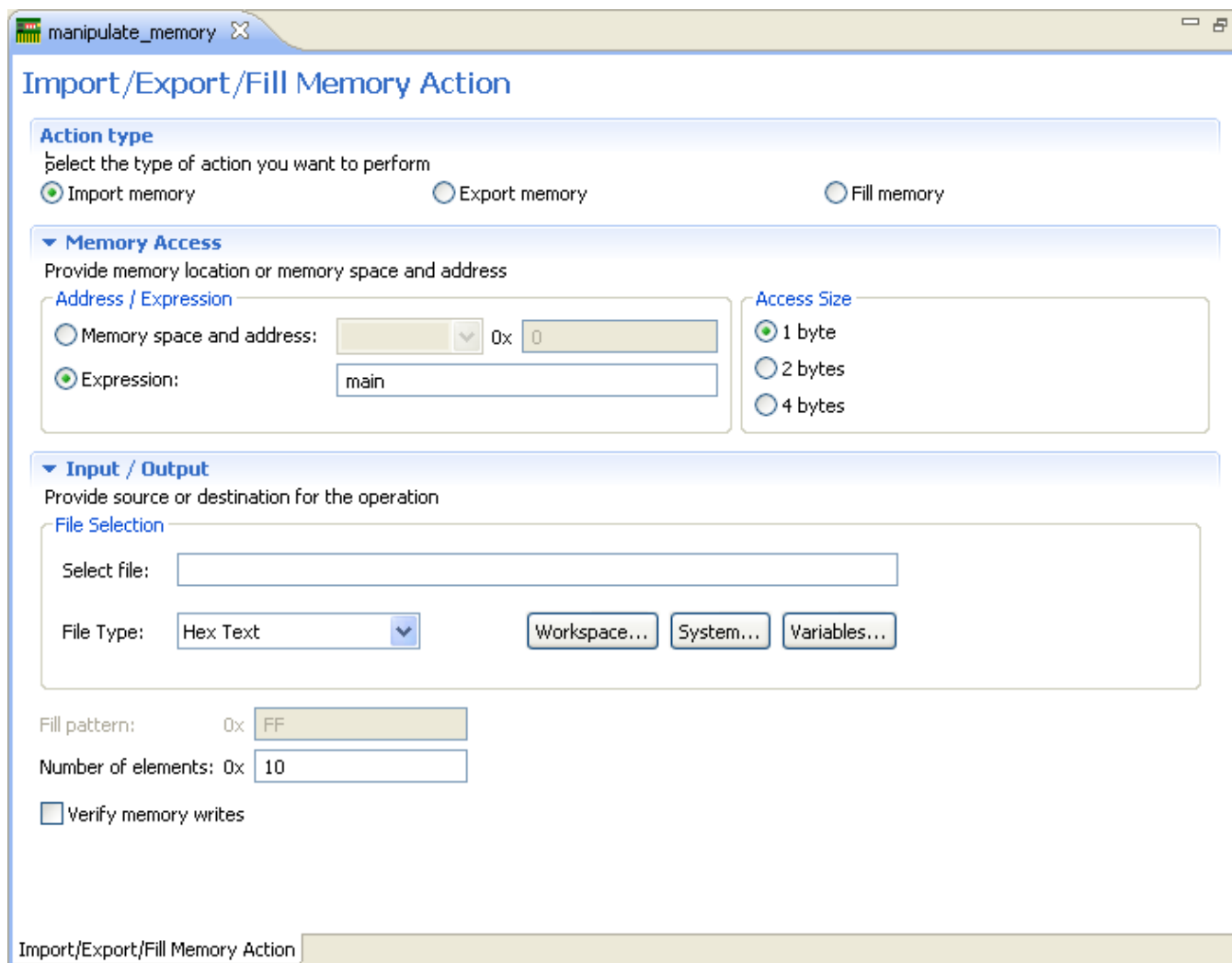


Figure 6-7. Import/Export/Fill Memory Action Editor

2. Select the type of the task you want to perform from the Action Type. You may perform the following tasks:
 - **Import memory:** Enables reading encoded data from a user specified file, decoding it, and copying it into a user specified memory range.
 - **Export memory:** Enables reading data from a user specified memory range, encoding it in a user specified format, and store this encoded data in a user specified output file.
 - **Fill memory:** Enables filling a user specified memory range with an user specified data pattern.

6.2.3 Import/Export/Fill Memory Action editor

The table below lists and defines all the User Interface (UI) options present within **Import/Export/Fill Memory Action** editor.

Table 6-2. Import/Export/Fill Memory Action Editor - Options

Item	Description
Import memory	Select the option to read the encoded data from a user specified file, decode it, and copy it into a user specified memory range.
Export memory	Select the option to read data from a user specified memory range, encode it in a user specified format, and store this encoded data in an output file.
Fill memory	Select the option to fill a selected memory range with specified data pattern.
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, 4, and 8 units. When target information is available, this list shall be filtered to display the access sizes that are supported by the target.
Select File	Enter the path of the file to export data. or 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.
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"> Annotated Hex Hex Text Motorola S-Record Raw Binary Signed decimal Text Unsigned decimal Text
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.
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.

Chapter 7

JTAG configuration files

This appendix explains about JTAG configuration files that pass specific configuration settings to the debugger and support chaining of multiple devices. A JTAG configuration file is a text file, specific to the CodeWarrior debugger, which describes a custom JTAG scan chain. You can specify the file in the remote system settings.

This appendix explains:

- [JTAG configuration file syntax](#)
- [Using JTAG configuration file to override RCW](#)
- [Using JTAG configuration file to specify multiple linked devices on a JTAG chain](#)
- [Setting remote system to use JTAG configuration file](#)

7.1 JTAG configuration file syntax

You can create a JTAG configuration file that specifies the type, the chain order, and various settings for the devices you want to debug. The listing below shows the complete syntax for a JTAG configuration file.

Listing 7-1. JTAG Configuration File Syntax

```
cfgfile:
    '\n'
    '#' 'any other characters until end of line'
    line
    cfgfile line
line:
    target
    target filter_list_or_params
target:
```

Using JTAG configuration file to override RCW

```
TARGET_NAME

TARGET_NAME = TARGET_ID

'Generic' NUMBER NUMBER NUMBER

filter_list_or_params:

    filter_list_entity

    filter_list_or_params filter_list_entity

filter_list_entity:

    '(' NUMBER NUMBER ')'

    FILTER_NAME

%
```

NOTE

Use the name of the processor as TARGET_NAME, such as P1010, P2020, P4080 and so on.

7.2 Using JTAG configuration file to override RCW

You can use JTAG configuration files to override Reset Configuration Word (RCW) for P4080 and other derivatives. The JTAG configuration files are used in the following situations for:

- target boards that do not have RCW already programmed
- new board bring-up
- recovering boards with blank or damaged flash

NOTE

For more information on RCW, see the Reference Manual for your processor.

The CodeWarrior IDE includes examples of JTAG configuration files that can be used for overriding the RCW ([Listing 7-2 on page 154](#)). The JTAG Configuration files are available at the following location:

<CWInstallDir>\CW_APP\LS\AIOP_Support\Initialization_Files\jtag_chains

Listing 7-2. Sample JTAG Configuration File for Overriding RCW

```
# Example file to allow overriding the whole RCW or only parts of it
#
# Syntax:
# P4080 (2 RCW_option) (RCWn value) ...
#
# where:
# RCW_option = 0 [RCW Override disabled]
```

```
#          1 [RCW Override enabled]
#          2 [Reset previous RCW Override parts]
#      0x80000001 [RCW Override + PLL Override]
#      NOTE: Enabling PLL Override could lead to hanging the chip
#
#      RCWn = 21000+n (n = 1 .. 16; index of RCW value)
#
#      value = 32bit value
```

The JTAG configuration files, as specified in [Listing 7-2 on page 154](#), can be used to override a portion or the complete RCW for P4080, by specifying (index, value) pairs, for some (or all) of the 16 x (32bit words) of the RCW.

NOTE

You can use the Pre-Boot Loader (PBL) tool to configure the various settings that make up the RCW and output the RCW in multiple formats, including CodeWarrior JTAG configuration files. For more information on the PBL tool, see the documentation provided with the BSP as the PBL tool is a part of the BSP.

7.3 Using JTAG configuration file to specify multiple linked devices on a JTAG chain

The listing and figure below shows a sample JTAG initialization file with a single core.

Listing 7-3. Sample JTAG Initialization File for P2020 Processor

```
# A single device in the chain
P2020
```

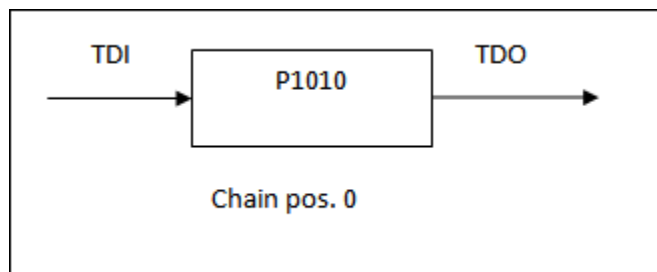


Figure 7-1. A Single Device in a JTAG Chain

The listing and figure below shows a sample JTAG initialization file with two devices in one JTAG chain.

Listing 7-4. Sample JTAG Initialization File for P2010 and P2020 Processors

```
# Two devices in one JTAG chain
P1023
```

P2020

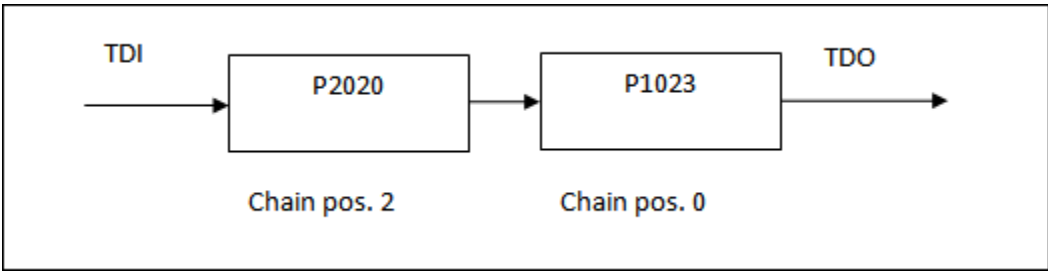


Figure 7-2. Two Devices in a JTAG Chain

NOTE

The devices are enumerated in the direction starting from TDO output to TDI input.

The listing and figure below shows two devices connected in one JTAG chain. Configuration data is passed to the first processor. Configuration data indexes and values are CCS-template specific and dependent on the type of core/processor.

Listing 7-5. Sample JTAG Initialization File for P4080 and P4040 Processors

```
# Two devices in one JTAG chain
P1010 (0x80000000 1)

P4080 (2 1) (210005 0x90404000) (210010 0x00000000)
```

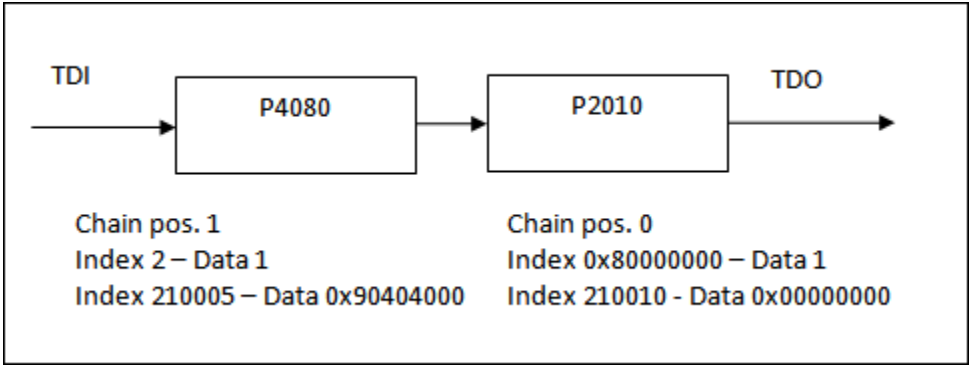


Figure 7-3. Two Devices in a JTAG Chain

The listing and figure below show two devices connected in one JTAG chain with a filter applied for the second device.

Listing 7-6. Sample JTAG Initialization File for Two Devices with Filter for Second Device

```
# Two devices in one JTAG chain
P2010 (0x80000000 1)

P4080 log
```

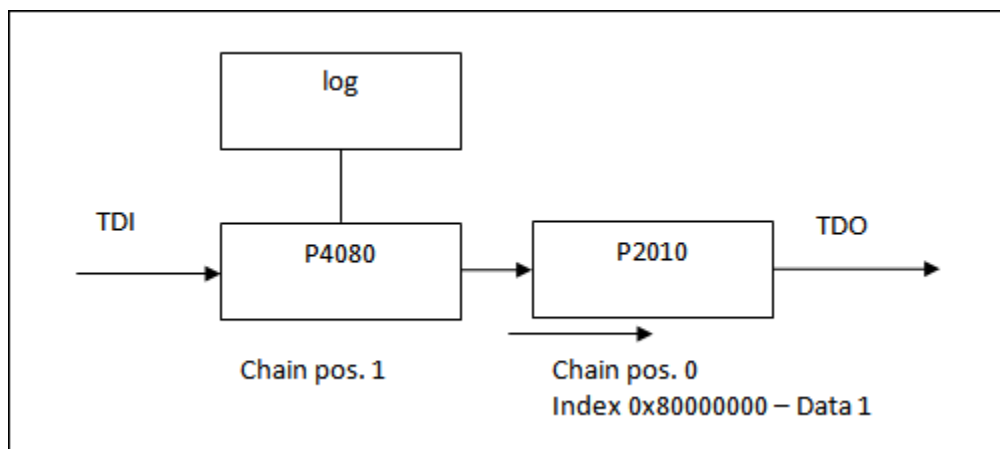


Figure 7-4. Two Devices in a JTAG Chain with Filter Applied to Second Device

7.4 Setting remote system to use JTAG configuration file

To connect to a JTAG chain, specify these settings in the launch configurations:

1. Create a JTAG initialization file that describes the items on the JTAG chain. For more information on how to create a JTAG initialization file, see [JTAG configuration file syntax](#) and [Using JTAG configuration file to specify multiple linked devices on a JTAG chain](#).
2. Open the CodeWarrior project you want to debug.
3. Select Run > Debug Configurations.

The Debug Configurations dialog box appears with a list of debug configurations that apply to the current application.

4. Expand the CodeWarrior tree control.
5. From the expanded list, select the debug configuration for which you want to modify the debugger settings.

The Debug view shows the settings for the selected configuration.

6. Select a remote system from the Connection drop-down list.
7. Select a core from the Target list.
8. In the Connection group, click the Edit button.

The Properties for <project> window appears.

9. Click Edit next to the Target list.

The Properties for <remote system> window appears.

Setting remote system to use JTAG configuration file

10. Click Edit next to the Target type drop-down list.

The Target Types dialog box appears.

11. Click Import.
12. The Import Target Type dialog box appears.
13. Select the JTAG initialization file that describes the items on the JTAG chain from this location: <CWInstallDir>\CW_APP\LS\AIOP_Support\Initialization_Files\jtag_chains
14. Click OK.

The items on the JTAG chain described in the file appear in the Target Types dialog box.

15. Click OK.

The selected JTAG configuration file appears on the Advanced tab (Figure 7-5).

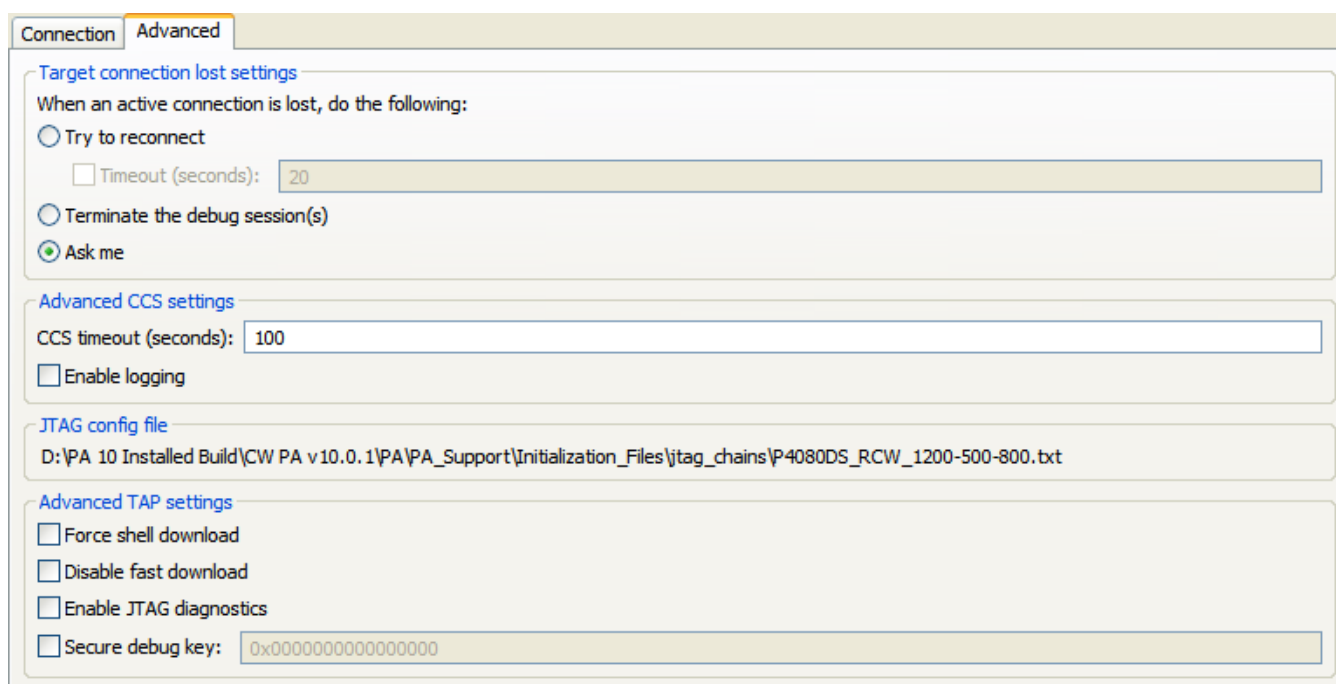


Figure 7-5. Advanced Tab Showing the JTAG Configuration File

16. Click **OK**.
17. Click the Debugger tab.

The Debugger page appears.

18. Ensure that Stop on startup at check box is checked and `main` is specified in the User specified text box.
19. Click Apply to save the changes.

You have successfully configured a debug configuration.

Chapter 8

Target initialization files

A target initialization file is a file that 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 bareboard binary to a target board. The commands in a target initialization file put a board in the state required to debug a bareboard 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).

This appendix explains:

- [Using target initialization files](#)
- [Target initialization file commands](#)
- [Target initialization using Tcl script](#)

8.1 Using target initialization files

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 instruct the CodeWarrior debugger to use a target initialization file:

1. Start the CodeWarrior IDE.
2. Open a bareboard project.
3. Select one of this project's build targets.
4. Select **Run > Debug Configurations**.

The Debug Configurations window appears.

5. Select the appropriate launch configuration from the left panel.
6. In the Main tab, from the Connection panel click the Edit button next to the Connection drop-down list.

The Properties for <Launch Configuration Name> window appears.

7. Click Edit next to the Target drop-down list.

The Properties for <remote system> window appears.

8. In the Initialization tab, check the appropriate cores check boxes from the Initialize target column as seen in [Figure 8-1](#).

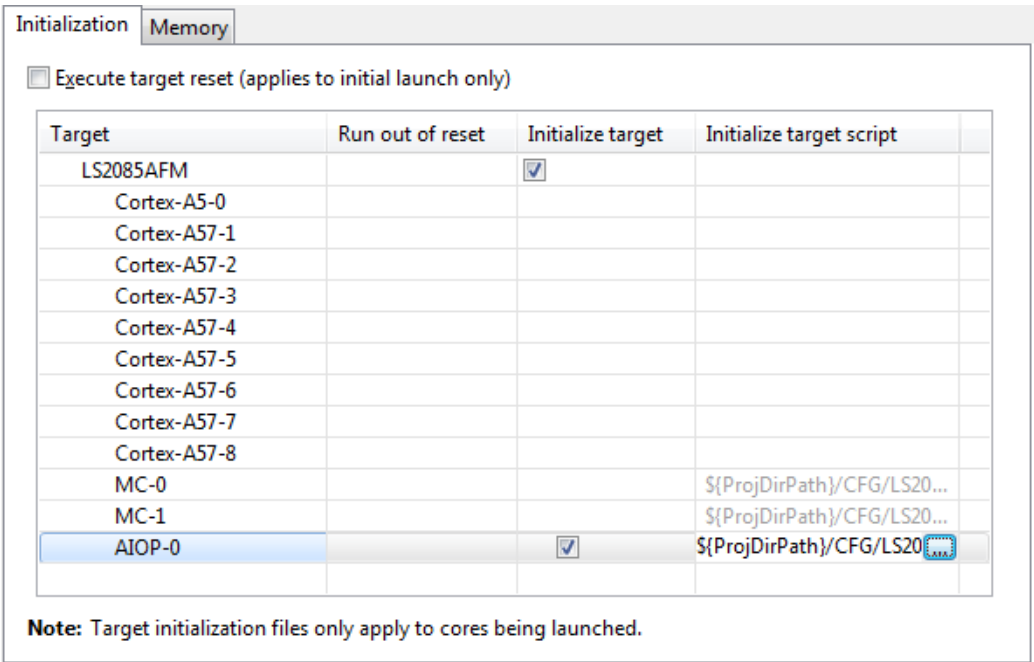


Figure 8-1. Initialization Tab

9. In the Initialize target script column, click the Ellipsis button as seen in [Figure 8-1](#).

Tip

Single-click in the specified cell of the Initialize target script column for the Ellipsis button to appear.

The Target Initialization File dialog box appears as seen in [Figure 8-2](#).

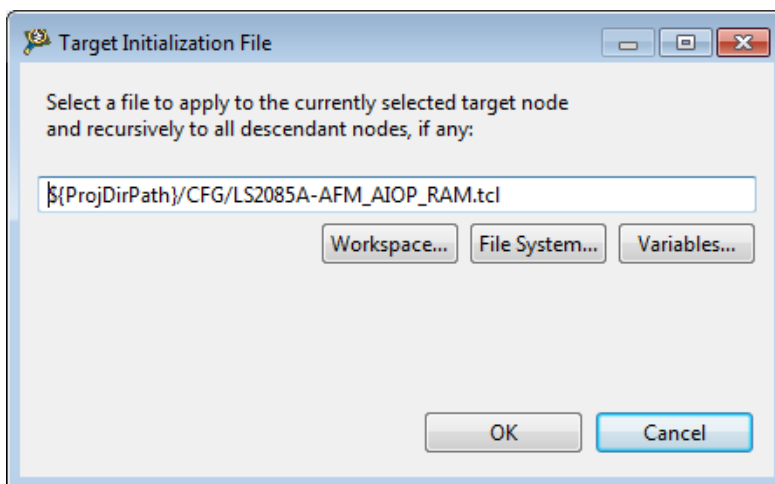


Figure 8-2. Target Initialization File Dialog Box

10. Select the target initialization file by using the buttons provided in the dialog box and Click OK.

The target initialization files are available at the following path:

```
CWInstallDir\CW_APP\LS\AIOP_Support\Initialization_Files\
```

You can also write your own target initialization files. The next section documents the commands that can appear in such files.

8.2 Target initialization file commands

The syntax of target initialization file commands follows these rules:

- Spaces and tabs (white space) are ignored
- Character case is ignored
- Unless otherwise noted, values may be specified in hexadecimal, octal, or decimal:
 - Hexadecimal values are preceded by 0x (for example, 0xDEADBEEF)

- Octal values are preceded by 0 (for example, 01234567)
- Decimal values start with a non-zero numeric character (for example, 1234)
- Comments start with a semicolon (;) or pound sign (#), and continue to the end of the line

Following sections will help you with more details on the target initialization file commands:

- [Access to named registers from within scripts](#)
- [Cfg target initialization commands](#)

8.2.1 Access to named registers from within scripts

Some commands described in the [Cfg target initialization commands](#) section (below) allow access to memory-mapped register by name as well as address. Based on the processor selection in the debugger settings, these commands will accept the register names shown in the debugger's Registers window. There are also commands to access built-in registers of a processor core, for example, 'writereg'. The names of these registers follow the architectural description for the respective processor core for general purpose and special purpose registers. Note that these names (for example, GPR5) might be different from names used in assembly language (for example, r5). You can identify the registers names by looking at the debugger's Registers window.

8.2.2 Cfg target initialization commands

Target initialization files are of two types: .cfg files and .tcl files. This section discusses the commands that are used in the cfg target initialization files.

NOTE

The tcl file format offers some advantages over the cfg file format, for example, it implements a better memory management approach and allows you to use memory address ranges higher than 32-bit and use flow control statements. The tcl file format is the recommended target initialization file format. The tcl file format is described in [Target initialization using Tcl script](#).

For each command, the section provides a brief statement of what the command does, the command's syntax, a definition of each argument that can be passed to the command, and examples showing how to use the command.

Table 8-1 lists each command that can appear in a target initialization file.

Table 8-1. Target Initialization File Commands

alternatePC	reset	stop	writemem.b
ANDmmr	resetCoreID	writemem.l	writemem.w
ANDmem.l	run	writemmr	writereg
IncorMMR	setCoreID	writereg64	writereg128
ORmem.l	sleep	writereg192	

8.2.2.1 alternatePC

Sets the initial program counter (PC) register to the specified value, disregarding any entry point value read from the ELF application being debugged.

Syntax

```
alternatePC
address
```

Arguments

address

The 32-bit address to assign to the program counter register.

This address may be specified in hexadecimal (for example, 0xABCD0000), octal (for example, 025363200000), or decimal (for example, 2882338816).

Example

This command assigns the address 0xc28737a4 to the program counter register:

```
alternatePC 0xc28737a4
```

8.2.2.2 ANDmem.l

Performs a bit AND using the 32-bit value at the specified memory address and the supplied 32-bit mask and writes the result back to the specified address.

No read/write verify is performed.

Syntax

```
ANDmem.l
address
mask
```

Arguments

address

The address of the 32-bit value upon which to perform the bit AND operation.

This address may be specified in hexadecimal (for example, 0xABCD0000), octal (for example, 025363200000), or decimal (for example, 2882338816).

mask

32-bit mask to use in the bit AND operation.

Example

The command below performs a bit AND operation using the 32-bit value at memory location 0xC30A0004 and the 32-bit mask 0xFFFFFFFF. The command then writes the result back to memory location 0xC30A0004.

```
ANDmem.l 0xC30A0004 0xFFFFFFFF
```

8.2.2.3 ANDmmr

Performs a bit AND using the contents of the specified memory-mapped register (MMR) and the supplied 32-bit mask and writes the result back to the specified register.

Syntax

```
ANDmmr
regName
```

mask

Arguments

regName

The name of the memory-mapped register upon which to perform a bit AND.

NOTE

For more information on the memory-mapped register names accepted by this command, refer to [Access to named registers from within scripts](#).

mask

32-bit mask to use in the bit AND operation.

Example

This command bit ANDs the contents of the `ACFG` register with the value `0x00002000`:

```
ANDmmr ACFG 0x00002000
```

8.2.2.4 IncorMMR

Performs a bitwise OR using the contents of the specified memory-mapped register (MMR) and the supplied 32-bit mask and writes the result back to the specified register.

Syntax

```
incorMMR  
regName  
mask
```

Arguments

regName

The name of the MMR register upon which to perform a bit OR.

NOTE

For more information on the memory-mapped register names accepted by this command, refer to [Access to named registers from within scripts](#).

mask

32-bit mask to use in the bit inclusive OR operation.

Example

This command bit ORs the contents of the `ACFG` register with the value `0x00002000`:

```
incorMMR ACFG 0x00002000
```

8.2.2.5 ORmem.l

Performs a bit OR using the 32-bit value at the specified memory address and the supplied 32-bit mask and writes the result back to the specified address.

No read/write verify is performed.

Syntax

```
ORmem.l  
address  
mask
```

Arguments

address

The address of the 32-bit value upon which to perform the bit OR operation.

This address may be specified in hexadecimal (for example, `0xABCD0000`), octal (for example, `025363200000`), or decimal (for example, `2882338816`).

mask

32-bit mask to use in the bit OR operation.

Example

The command below performs a bit OR operation using the 32-bit value at memory location `0xC30A0008` and the 32-bit mask `0x01000800`. The command then writes the result back to memory location `0xC30A0004`.

```
ORmem.1 0xC30A0008 0x01000800
```

8.2.2.6 reset

Resets the processor on the target board.

Syntax

```
reset
code
```

Arguments

code

Number that defines what the debugger does after it resets the processor on the target board.

[Table 8-2](#) describes the Post Reset Actions. Use any one of the values specified.

Table 8-2. Post Reset Actions

Value	Description
0	reset the target processor, then run
1	reset the target processor, then stop

8.2.2.7 run

Starts program execution at the current program counter (PC) address.

Syntax

```
run
```

8.2.2.8 setCoreID

Tells the debugger to issue all subsequent commands on the specified core index, disregarding the actual core index on which the initialization is executed.

NOTE

Please make sure to reset the core index after the sequence of commands intended to execute on the other core is finished. Refer to the [resetCoreID](#) command.

Tip

This command can be useful in cases where you need to execute a command sequence on other cores than the current one, for example in a SMP initialization scenario.

Syntax

```
setCoreID core
```

Arguments

core

The core index on which to execute.

Example

This command tells the debugger to issue all subsequent commands on the core index 1:
setCoreID 1

8.2.2.9 resetCoreID

Tells the debugger to revert to executing commands on the current core, thus cancelling the effect of a previous setCoreID command.

Syntax

```
resetCoreID
```


8.2.2.10 sleep

Causes the debugger to pause the specified number of milliseconds before executing the next instruction.

Syntax

```
sleep  
milliseconds
```

Arguments

milliseconds

The number of milliseconds (in decimal) to pause the debugger.

Example

This command pauses the debugger for 10 milliseconds:

```
sleep 10
```

8.2.2.11 stop

Stops program execution and halts the processor on the target board.

Syntax

```
stop
```

8.2.2.12 writemem.b

Writes a byte (8 bits) of data to the specified memory address.

Syntax

```
writemem.b
address
value
```

Arguments

address

The 32-bit memory address to which to assign the supplied 8-bit value.

This address may be specified in hexadecimal (for example, 0xABCD), octal ((for example, 0125715), or decimal (43981).

value

The 8-bit value to write to the specified memory address.

This value may be specified in hexadecimal (for example, 0xFF), octal (for example, 0377), or decimal (for example, 255).

Example

This command writes the byte 0x1A to the memory location 0x0001FF00:

```
writemem.b 0x0001FF00 0x1A
```

8.2.2.13 writemem.w

Writes a word (16 bits) of data to the specified memory address.

Syntax

```
writemem.w
address
value
```

Arguments

address

The 32-bit memory address to which to assign the supplied 16-bit value.

This address may be specified in hexadecimal (for example, 0xABCD0000), octal (for example, 025363200000), or decimal (for example, 2882338816).

value

The 16-bit value to write to the specified memory address.

This value may be specified in hexadecimal (for example, 0xFFFF), octal (for example, 0177777), or decimal (for example, 65535).

Example

This command writes the word 0x1234 to memory location 0x0001FF00:

```
writemem.w 0x0001FF00 0x1234
```

8.2.2.14 writemem.l

Writes a long integer (32 bits) of data to the specified memory location.

Syntax

```
writemem.l  
address  
value
```

Arguments

address

The 32-bit memory address to which to assign the supplied 32-bit value.

This address may be specified in hexadecimal (for example, 0xABCD0000), octal (for example, 025363200000), or decimal (for example, 2882338816).

value

The 32-bit value to write to the specified memory address.

This value may be specified in hexadecimal (for example, 0xFFFFABCD), octal (for example, 037777725715), or decimal (for example, 4294945741).

Example

This command writes the long integer 0x12345678 to the memory location 0x0001FF00:

```
writemem.w 0x0001FF00 0x12345678
```

8.2.2.15 writemmr

Writes a value to the specified memory-mapped register (MMR).

Syntax

```
writemmr regName value
```

Arguments

regName

The name of the memory-mapped register to which to assign the supplied value.

NOTE

This command accepts most of the processor memory-mapped register names. For more information on the memory-mapped register names accepted by this command, refer to [Access to named registers from within scripts](#).

value

The value to write to the specified memory-mapped register.

This value may be specified in hexadecimal (for example, 0xFFFFABCD), octal (for example, 037777725715), or decimal (for example, 4294945741).

Example

This command writes the value 0xffffffffc3 to the SYPCR register:

```
writemmr SYPCR 0xffffffffc3
```

This command writes the value 0x0001 to the RMR register:

```
writemmr RMR 0x0001
```

This command writes the value 0x3200 to the MPTPR register:

```
writemmr MPTPR 0x3200
```

8.2.2.16 writereg

Writes the supplied data to the specified register.

Syntax

```
writereg  
regName value
```

Parameters

regName

The name of the register to which to assign the supplied value.

value

The value to write to the specified register.

This value may be specified in hexadecimal (for example, 0xFFFFABCD), octal (for example, 037777725715), or decimal (for example, 4294945741).

Example

This command writes the value 0x00001002 to the MSR register:

```
writereg MSR 0x00001002
```

8.2.2.17 writereg64

Writes the supplied 32-bit values to the specified 64-bit register.

NOTE

This command is applicable only to 64-bit Book E cores like the e5500.

Syntax

```
writereg regName value1 value2
```

Arguments

regName

The name of the 64-bit register to which to assign the supplied value.

target initialization file commands

value1, value2

The two 32-bit values that together make up the 64-bit value to assign to the specified register.

Each value may be specified in hexadecimal (for example, 0xFFFFABCD), octal (for example, 037777725715), or decimal (for example, 4294945741).

Example

This command writes the 64-bit value 0x0123456789ABCDEF to the 64-bit GPR5 register:

```
writereg64 GPR5 0x01234567 0x89ABCDEF
```

8.2.2.18 writereg128

Writes the supplied 32-bit values to the specified TLB register.

NOTE

This command is applicable only to Book E cores like the e500 or e500mc variants.

Syntax

```
writereg128  
regName value1 value2 value3 value4
```

Arguments

regName

The name (or number) of the TLB register to which to assign the specified values.

Tip

Valid TLB0 register names range from L2MMU_TLB0 through L2MMU_TLB255 (L2MMU_TLB511 for e500v2 and e500mc).

Tip

Valid TLB1 register names range from L2MMU_CAM0 through L2MMU_CAM15, and L2MMU_CAM63 for e500mc.

value1, value2, value3, value4

The four 32-bit values that together make up the 128-bit value to assign to the specified TLB register.

Each value must be specified in hexadecimal (for example, 0xFFFFFABCD).

Example

This command writes the values 0xA1002, 0xB1003, 0xC1004, and 0xD1005 to the L2MMU_CAM0 TLB register:

```
writereg128 L2MMU_CAM1 0x7000000A 0x1C080000 0xFE000000 0xFE000001
```

8.2.2.19 writereg192

Writes the supplied 32-bit values to the specified TLB register.

NOTE

This command is applicable only to 64-bit Book E cores like the e5500 variant.

Syntax

```
writereg192
regName value1 value2 value3 value4 value5 value6
```

Arguments

regName

The name (or number) of the TLB register to which to assign the specified values.

Tip

Valid TLB0 register names range from L2MMU_TLB0 through L2MMU_TLB511.

Tip

Valid TLB1 register names range from L2MMU_CAM0 through L2MMU_CAM63.

value1, value2, value3, value4, value5, value6

The six 32-bit values that together make up the 192-bit value to assign to the specified TLB register.

Each value must be specified in hexadecimal (for example, 0xFFFFABCD).

Example

This command writes the values 0x7000000A 0x1C080000 0x00000000 0xFE000000 0x00000000 0xFE000001 to the L2MMU_CAM1 TLB register:

```
writereg192 L2MMU_CAM1 0x7000000A 0x1C080000 0x00000000 0xFE000000
0x00000000 0xFE000001
```

8.2.2.20 writespr

Writes the specified value to the specified SPR register.

NOTE

This command is similar to the `writereg SPRxxx` command, except that `writespr` lets you specify the SPR register to modify by number (in hexadecimal, octal, or decimal).

Syntax

```
writespr regNumber value
```

Arguments

regNumber

The number of the SPR register to which to assign the supplied value.

This value may be specified in hexadecimal (for example, 0x27E), octal (for example, 01176), or decimal (for example, 638).

value

The value to write to the specified SPR register.

This value may be specified in hexadecimal (for example, 0xFFFFABCD), octal (for example, 03777725715), or decimal (for example, 4294945741).

Example

This command writes the value 0x0220000 to SPR register 638:

```
writespr 638 0x0220000
```


8.3 Target initialization using Tcl script

The CodeWarrior debugger allows you to specify a Tcl-based script to be run instead of a .cfg initialization file. Similar to a .cfg initialization file, the Tcl-based initialization file can contain target-specific initialization, processor core initialization, or debugger-specific initialization.

NOTE

The current releases do not include .cfg files. But the .cfg files continue to be supported to provide backward compatibility.

The debugger automatically executes the Tcl script when you debug the launch configuration. You can also execute the script manually at any time from the Debugger Shell, by using the `source` command. The Tcl-based target initialization is basically a debugger shell script and implicitly supports all debugger shell commands. For more details on the debugger shell commands, refer to the CodeWarrior Common Features manual.

[Table 8-3](#) lists the equivalent Debugger Shell commands that you can include in a Tcl script for target initialization.

Table 8-3. Tcl Commands for Target Initialization

Target Initialization Commands	Debugger Shell Equivalent
writereg, writereg64, writereg128, writereg192	reg or change
writespr	reg or change (partial equivalence - uses the register name instead of the spr number)
writemem.l	mem 32bit or change 32bit
writemem.w	mem 16bit or change 16bit
writemem.b	mem 8bit or change 8bit
sleep	wait
writemmr	reg or change
IncOrmmr	change regName [format %x [expr [reg regName %d -np] [expr mask]]] or reg regName = [format %x [expr [reg regName %d -np] [expr mask]]]
ANDmmr	change regName [format %x [expr [reg regName %d -np] & [expr mask]]] or reg regName = [format %x [expr [reg regName %d -np] [expr mask]]]
setCoreID	aiop::setcoreid
resetCoreID	aiop::setcoreid default
run	go

Table continues on the next page...

Table 8-3. Tcl Commands for Target Initialization (continued)

Target Initialization Commands	Debugger Shell Equivalent
stop	stop
reset	reset
ANDmem.l	change address [format %x [expr [mem address %d -np] & [expr mask]]] or mem address = [format %x [expr [mem address %d -np] & [expr mask]]]
ORmem.l	change address [format %x [expr [mem address %d -np] [expr mask]]] or mem address = [format %x [expr [mem address %d -np] & [expr mask]]]
alternatePC	N/A

Tip

When accessing registers, for best performance you can add the register group name followed by '/' before the name of the register, for example:

reg "e500mc Special Purpose Registers"/MSR =0x00002000

Chapter 9

Memory configuration files

A memory configuration file contains commands that define the rules the debugger follows when accessing a target board's memory.

NOTE

Memory configuration files do not define the memory map for the target. Instead, they define how the debugger should treat the memory map the target has already established. The actual memory map is initialized either by a target resident boot loader or by a target initialization file. For more information, refer to .

If necessary, you can have the CodeWarrior debugger execute a memory configuration file immediately before the debugger downloads a bareboard binary to a target board. The memory configuration file defines the memory access rules (restrictions, translations) used each time the debugger needs to access memory on the target board.

NOTE

Assign a memory configuration file to bareboard build targets only. The memory of a board that boots embedded Linux® is already set up properly. A memory configuration file defines memory access rules for the debugger; the file has nothing to do with the OS running on a board. If needed, a memory configuration file should be in place at all times. The Linux Kernel Aware Plugin performs memory translations automatically, relieving the user from specifying them in the memory configuration file.

This appendix explains:

- [Using memory configuration files](#)
- [Memory configuration file commands](#)

9.1 Using memory configuration files

A memory configuration file is a command file that contains memory access rules that the CodeWarrior debugger uses each time the build target to which the configuration file is assigned is debugged.

You specify a memory configuration file in the **Memory** tab of the remote system configuration.

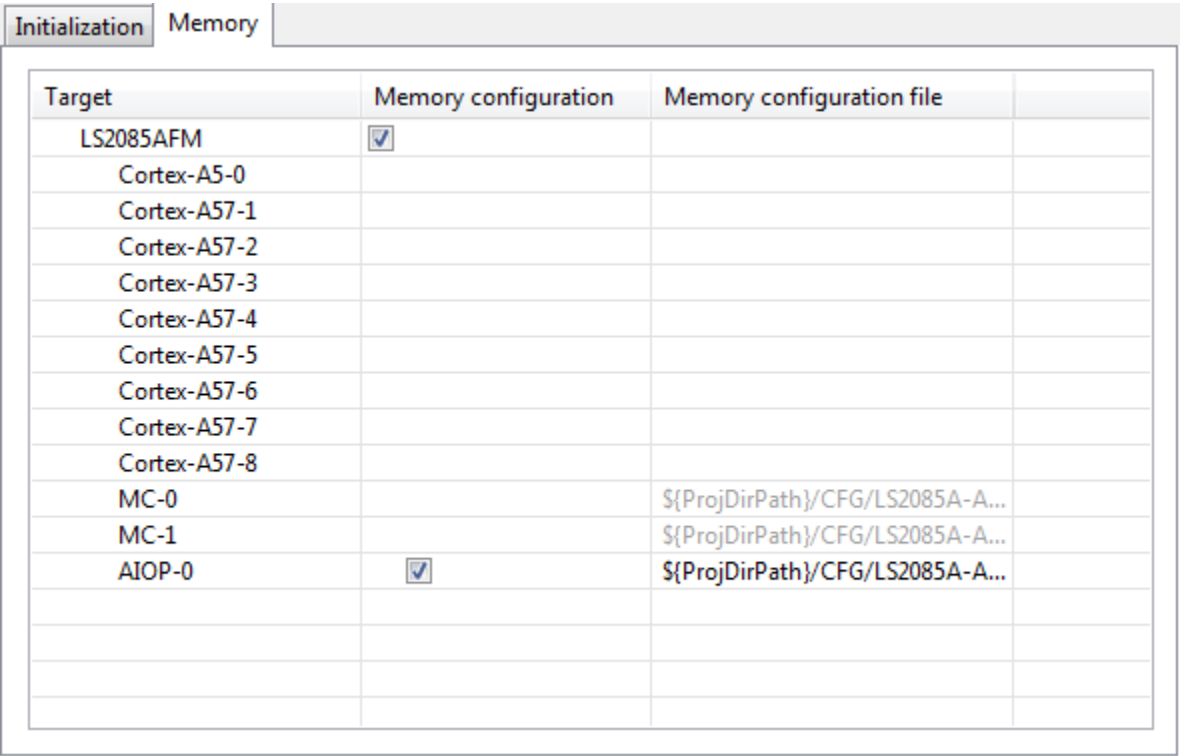


Figure 9-1. Specifying a memory configuration file

You can also write your own memory configuration files. The next section documents the commands that can appear in such files.

9.2 Memory configuration file commands

In general, the syntax of memory configuration file commands follows these rules:

- Spaces and tabs (white space) are ignored
- Character case is ignored
- Unless otherwise noted, values may be specified in hexadecimal, octal, or decimal:
 - hexadecimal values are preceded by 0x (for example, 0xDEADBEEF)

- octal values are preceded by 0 (for example, 01234567)
- decimal values start with a non-zero numeric character (for example, 1234)
- Addresses are values that might be prefixed with the memory space command line prefix: [<MemSP>:]<value>. For example: p:0x80000004 or 0x80000004.
- Comments start with standard C and C++ comment characters, and continue to the end of the line

The table below lists each command that can appear in a memory configuration file. For each command, the section provides a brief statement of what the command does, the command's syntax, a definition of each argument that can be passed to the command, and examples showing how to use the command.

Table 9-1. Memory Configuration Commands

autoEnableTranslations	range
reserved	reservedchar
translate	

9.2.1 autoEnableTranslations

The autoEnableTranslations command configures if the translate commands are considered by the debugger or not.

Syntax

```
autoEnableTranslations enableFlag
```

Arguments

enableFlag

Pass true to instruct the debugger to consider the translate commands.

If this command is not present, the translations will not be considered, so this command should usually be present and have a "true" argument.

Examples

This command enables the debugger to consider the translate commands:

```
AutoEnableTranslations true
```

9.2.2 range

This command sets debugger access to a block of memory. Note that the `range` command must have both the `loAddress` and `hiAddress` in the same memory space.

Syntax

```
range
loAddress hiAddress size access
```

Arguments

`loAddress`

the starting address of the memory range

`hiAddress`

the ending address of the memory range

`size`

the size, in bytes, the debug monitor or emulator uses for memory accesses

`access`

controls what type of access the debugger has to the memory block - supply one of: `Read`, `Write`, or `ReadWrite`

Examples

To set memory locations `0xFF000000` through `0xFF0000FF` to read-only with a size of 4 bytes:

```
range 0xFF000000 0xFF0000FF 4 Read
```

To set memory locations `0xFF0001000` through `0xFF0001FF` to write-only with a size of 2 bytes:

```
range 0xFF000100 0xFF0001FF 2 Write
```

To set memory locations `0xFF0002000` through `0xFFFFFFFF` to read and write with a size of 1 byte:

```
range 0xFF000200 0xFFFFFFFF 1 ReadWrite
```

9.2.3 reserved

This command allows you to specify a reserved range of memory. If the debugger attempts to read reserved memory, the resulting buffer is filled with the reserved character. If the debugger attempts to write to reserved memory, no write takes place. Note that the `reserved` command must have both the `loAddress` and `hiAddress` in the same memory space.

NOTE

For information showing how to set the reserved character, refer to [reservedchar](#).

Syntax

```
reserved  
loAddress hiAddress
```

Arguments

`loAddress`

the starting address of the memory range

`hiAddress`

the ending address of the memory range

Examples

To reserve memory starting at 0xFF000024 and ending at 0xFF00002F:

```
reserved 0xFF000024 0xFF00002F
```

9.2.4 reservedchar

This command sets the reserved character for the memory configuration file. When the debugger attempts to read a reserved or invalid memory location, it fills the buffer with this character.

Syntax

```
reservedchar rChar
```

Arguments

rChar

the one-byte character the debugger uses when it accesses reserved or invalid memory

Example

To set the reserved character to "x":

```
reservedchar 0x78
```

9.2.5 translate

This command lets you configure how the debugger performs virtual-to-physical memory address translations. Typically, you use address translations to debug programs that use a memory management unit (MMU) that performs block address translations.

Syntax

```
translate  
virtualAddress  
physicalAddress  
numBytes
```

Arguments

virtualAddress

the address of the first byte of the virtual address range to translate

physicalAddress

the address of the first byte of the physical address range to which the debugger translates virtual addresses

numBytes

the size (in bytes) of the address range to translate

Example

The following `translate` command:

- defines a one-megabyte address range (`0x100000` bytes is one megabyte)
- instructs the debugger to convert a virtual address in the range `0xC0000000` to `0xC0100000` to the corresponding physical address in the range `0x00000000` to `0x00100000`

```
translate v:0xC0000000 p:0x00000000 0x100000
```



Chapter 10

Debugger limitations and workarounds

This appendix documents processor-specific CodeWarrior debugger limitations and workarounds.

10.1 MC/AIOP cores

Limit for the number of MC/AIOP cores

Using a JTAG configuration file, limitations on the number of MC or AIOP cores can be specified in order to be able to limit debugging to a specific number of cores (regardless the actual target but limited to the actual hardware resources).

A JTAG configuration file that can be used to specify restrictions must respect the following syntax:

```
<TargetName> (<LimitationID> <ActualLimitation>) and a new line at the end
```

Where <LimitationId> must be:

- 0xAB0FF11B – limitation for the number of AIOP cores
- 0xAB0FF11A – limitation for the number of MC cores

Following is an example of JTAG config file specifying a restriction:

```
LS2085AFM (0xAB0FF11B 4)
```

The above example instructs CodeWarrior to debug only 4 cores out of 16 (available for LS2085AFM) for the AIOP

Working with Software Breakpoints

For MC/AIOP cores, the debugger implements software breakpoints by using the dedicated debug notify halt (dnh) instruction. When the dnh opcode is encountered, the target stops without taking an exception.

Working with Watchpoints

The MC cores implement four and the AIOP cores implement two data address compare registers. The CodeWarrior debugger uses these registers to place a single watchpoint on any variable or memory range. The variable or memory range is 1-byte aligned.

As opposed to hardware, simulators are not usually limited by the available comparator resources and allow a much higher number of watchpoints (1024).

Working with Hardware Breakpoints

The MC cores implement eight and the AIOP cores implement four address compare registers that can be used in a debug session.

As opposed to hardware, simulators are not usually limited by the available comparator resources and allow a much higher number of hardware breakpoints(1024).

Working with Uninitialized Stack

Debugging while the stack is not initialized can cause uninitialized memory accesses errors. This situation occurs when the debugger tries to construct the stack trace.

To avoid this problem, stop the debugger from constructing a stack trace by adding a command to your target initialization file that sets the stack pointer (SP) register to an unaligned address.

For example, you could put this command in your target initialization file:

```
reg SP = 0x3
```

Index

A

Access to Named Registers from within Scripts [162](#)
 Accompanying Documentation [11](#)
 Actions [113](#)
 Activating task awareness services [88](#)
 Adding a Register Group [109](#)
 Add Watchpoint Dialog Box [118](#)
 AIOP Cores [187](#)
 AIOP Target OS [88](#)
 alternatePC [163](#)
 ANDmem.l [164](#)
 ANDmmr [164](#)
 Arguments [40](#)
 Assembler [77](#)
 Auto-Build Mode [32](#)
 autoEnableTranslations [181](#)

B

Bit Fields [112](#)
 Building Projects [31](#)
 Build Properties [59](#)
 Build Properties for APP [60](#)
 Build Settings Page [23](#)

C

C/C++ Compiler [13](#)
 C/C++ Language [75](#)
 CCSIM2 ISS [98](#)
 Cfg Target Initialization Commands [162](#)
 Changing a Bit Field [116](#)
 Changing a Register's Bit Value [111](#)
 Changing Build Properties [59](#)
 Changing Program Counter Value [127](#)
 CodeWarrior Bareboard Project Wizard [19](#)
 CodeWarrior Command-Line Debugger [103](#)
 CodeWarrior Development Process [15](#)
 CodeWarrior Development Tools [12](#)
 CodeWarrior Executable Importer Wizard [131](#)
 CodeWarrior Profiling and Analysis Tools [15](#)
 CodeWarrior TAP [99](#)
 Common [55](#)
 Compiler [68](#)
 Compiling [16](#)
 Configurations Page [24, 134](#)
 Connection Types [97](#)
 Coverage Analysis Page [26](#)
 CPU [61](#)
 Create a CodeWarrior Bareboard Project Page [20](#)
 Creating CodeWarrior Bareboard Application Project [27](#)

Creating CodeWarrior Bareboard Library Project [30](#)
 Creating New Hardware Diagnostics Task [143](#)
 Creating Projects [26](#)
 Creating Target Task to Manipulate Memory [149](#)
 Customizing Debug Configurations [57](#)
 Custom Memory Locations [91](#)

D

Debug [41](#)
 Debug Configuration [35](#)
 Debugger [14, 40](#)
 Debugger Limitations and Workarounds [187](#)
 Debugging [17, 62](#)
 Debugging an Externally Built Executable File [135](#)
 Debugging Projects [32](#)
 Debug Target Settings Page [22, 133](#)
 Deleting a Project [33](#)
 Description [113](#)
 Disassembler [79](#)
 Disassembler Settings [79](#)
 Displaying Memory Contents [106](#)
 Displaying Register Contents [108](#)
 Download [43](#)

E

Eclipse IDE [12](#)
 Editing a Register Group [110](#)
 Editing Code [17](#)
 Editing Hardware Diagnostics Task [146](#)
 Editing Import/Export/Fill Memory Task [150](#)
 Editing System Configuration [101](#)
 Environment [54](#)
 Executing Hardware Diagnostics Task [146](#)

F

Filling Memory [130](#)

G

GCov Analysis [52](#)
 General [66, 78](#)

H

Hard Resetting [127](#)
 Hardware Diagnostics Action Editor [148](#)

I

Import/Export/Fill Memory Action Editor [151](#)
Import a CodeWarrior Executable file Page [132](#)
Import C/C++/Assembler Executable Files [132](#)
IncorMMR [165](#)
Initialization [101](#)
Input [64](#), [69](#), [77](#)
Introduction [11](#)

J

JTAG Configuration Files [153](#)
JTAG Configuration File Syntax [153](#)

L

Linker [14](#), [63](#)
Linking [16](#)
Link Order [65](#)
Loading and Saving Memory [128](#)

M

Main [36](#)
Manipulating Target Memory [148](#)
Manual-Build Mode [32](#)
MC Cores [187](#)
Memory [102](#)
Memory Configuration File [105](#)
Memory Configuration File Commands [180](#)
Memory Configuration Files [179](#)
Messages [63](#)
Multi-Core Operations [139](#)
Multi-Core Operations in the Debugger Shell [140](#)
Multi-Core Operations in the IDE [139](#)

O

Optimization [72](#)
ORmem.l [166](#)
OS Awareness [50](#)
Other Executables [47](#)
Output [66](#)

P

Performing run control operations [95](#)
PIC [45](#)
Preprocessor [69](#), [80](#)
Preprocessor Settings [80](#)
Processor [73](#)
Processor Page [21](#), [133](#)
Project Files [15](#)

R

range [182](#)
Release Notes [11](#)
Remove Breakpoint Using Breakpoints View [125](#)
Remove Breakpoint Using Marker Bar [125](#)
Remove Hardware Breakpoint Using Debugger Shell [126](#)
Remove Hardware Breakpoint Using IDE [126](#)
Removing a Register Group [110](#)
Removing Breakpoints [125](#)
Removing Hardware Breakpoints [126](#)
Removing Watchpoints [120](#)
reserved [183](#)
reservedchar [183](#)
reset [167](#)
resetCoreID [168](#)
Restoring Build Properties [60](#)
Reverting Debug Configuration Settings [58](#)
run [167](#)

S

setCoreID [168](#)
Setting a Remote System to Use a JTAG Configuration File [157](#)
Setting Breakpoints [120](#)
Setting Hardware Breakpoints [124](#)
Setting the Stack Depth [127](#)
Setting Watchpoints [117](#)
sleep [169](#)
Source [53](#)
Standalone Assembler [13](#)
Standard Debugging Features [97](#)
stop [169](#)
Symbolics [49](#)
System Browser [91](#)
System Call Services [46](#)

T

Targeting AIOP tasks [94](#)
Target Initialization File Commands [161](#)
Target Initialization Files [159](#)
Target Initialization Using a Tcl Script [177](#)
translate [184](#)

U

Use the Debugger Shell to Set a Hardware Breakpoint [124](#)
Use the IDE to Set a Hardware Breakpoint [124](#)
Using CodeWarrior Debug Configuration Tabs [35](#)
Using JTAG Configuration File to Override RCW [154](#)
Using JTAG Configuration File to Specify Multiple Linked Devices on a JTAG Chain [155](#)

Using Memory Configuration Files [180](#)
 Using Target Initialization Files [159](#)
 Using the Register Details Window [111](#)

V

Viewing AIOP tasks [89](#)
 Viewing order scope manager data [93](#)
 Viewing Register Details [114](#)
 Viewing task entry point data [93](#)

W

Warnings [71](#)
 Working with Hardware Diagnostics [143](#)
 Working with Hardware Tools [143](#)
 Working with Projects [19](#)
 Working with the Debugger [83](#)
 writemem.b [169](#)
 writemem.l [171](#)
 writemem.w [170](#)
 writemmr [172](#)
 writereg [172](#)
 writereg128 [174](#)
 writereg192 [175](#)
 writereg64 [173](#)
 writespr [176](#)





How to Reach Us:

Home Page:

freescale.com

Web Support:

freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 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 does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, and CodeWarrior are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. QorIQ is trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, Cortex, Cortex-A53, Cortex-A57, and TrustZone are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2013–2016 Freescale Semiconductor, Inc. All rights reserved.

Document Number CWAPPTM
Revision 10.2, 01/2016

