CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA, Targeting Manual

Document Number: CWARMv8TM
Rev. 04/2015

freescale™

Contents

Section number **Title** **Page**

Chapter 1 Introduction

Chapter 2 Working with Projects

Chapter 3 ARMv8 Build Properties

Chapter 4
Preparing target

Chapter 5
Configuring Target

Section number Title Page

Chapter 6
FSL Debugger References

Chapter 7
Flash programmer

Chapter 8
Use Cases

Chapter 9
Troubleshooting

Chapter 1 Introduction

This manual explains how to use the CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA product. This chapter presents an overview of the manual.

The topics in this chapter are:

- Release Notes - Lists new features, bug fixes, and incompatibilities
- About this Manual - Describes the contents of this manual
- Accompanying Documentation - Describes supplementary CodeWarrior documentation, third-party documentation, and references.

1.1 Release Notes

Release nots lists new features, bug fixes, and incompatibilities.

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

NOTE

> The release notes for specific components of the CodeWarrior IDE are located in the `ARMv8` folder in the CodeWarrior for CW4NET installation directory.

1.2 About this Manual

This topic lists each chapter of this manual, which describes a different area of software development.

The following table lists the contents of this manual.

Table 1-1. Manual contents

Chapter	Description
Introduction	This chapter.
Working with Projects	Lists the various project types and explains how to create projects.
ARMv8 Build Properties	Explains the CodeWarrior build tools and build tool configurations.
Preparing target	Explains how to prepare for debug various target types.
Configuring Target	Explains Target Connection Configuration (TCC) feature.
FSL Debugger References	Explains debugger features.
Flash programmer	Explains how to configure, start, and use flash programmer
Use Cases	Lists U-Boot debug, Linux application debug, and Linux kernel debug use cases.
Troubleshooting	Lists troubleshooting information.

1.3 Accompanying Documentation

The Documentation page describes the documentation included in this version of CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA. You can access the Documentation page by:

- Opening START_HERE.html in <*CWInstallDir*>*CW_ARMv8\ARMv8\Help* folder
- Selecting **Help > Documentation**.

To view the online help for the CodeWarrior tools select **Help > Help Contents** from the IDE menu bar.

Chapter 2
Working with Projects

This chapter lists the various project types and explains how to create and work with projects.

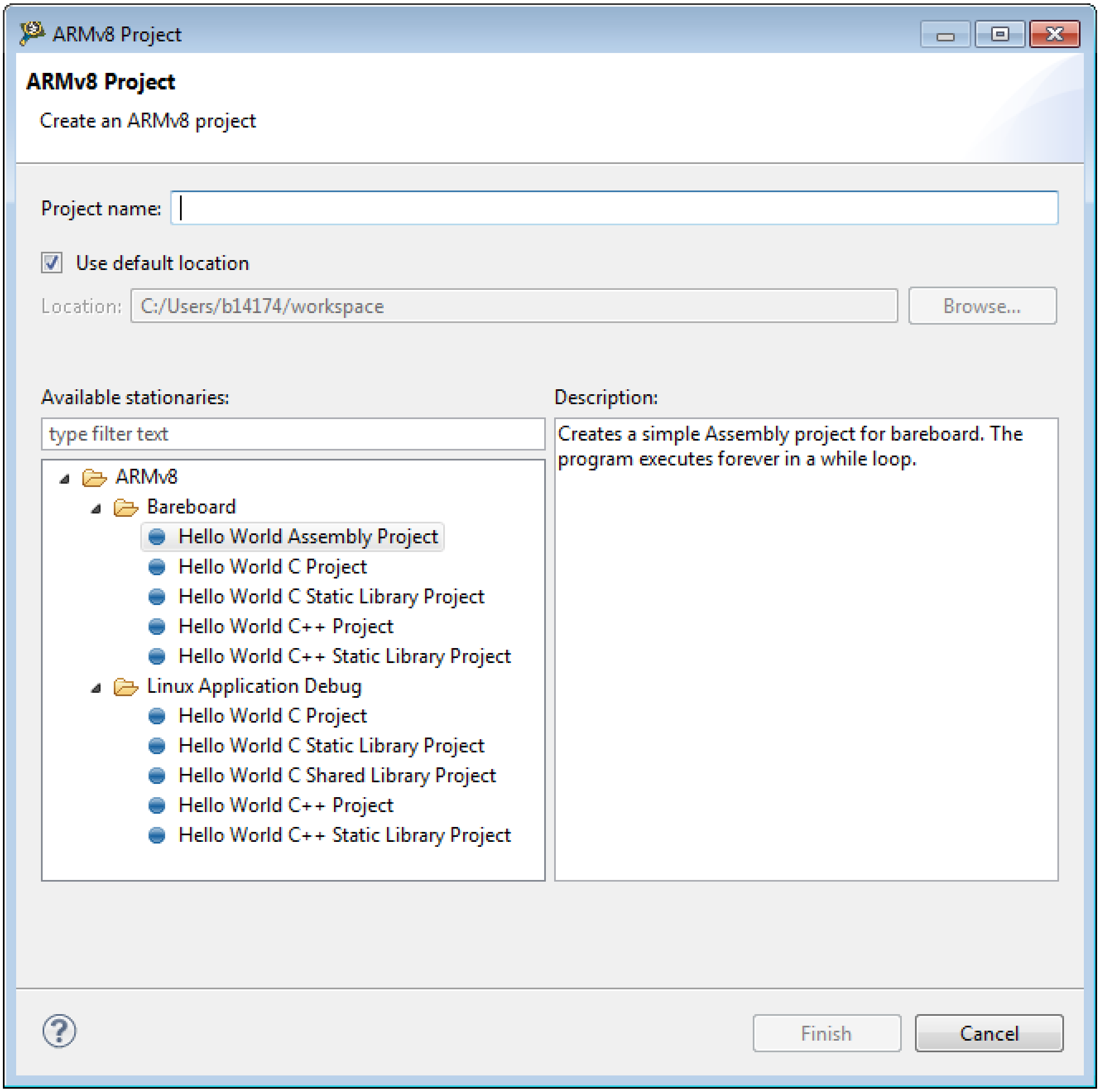
2.1 ARMv8 New Project wizard

The New Project wizard presents a selection of sample projects preconfigured for build using the bundled Linaro GCC toolchains.

Hello World projects for bareboard and Linux oriented (C, C++, ASM, static and shared library) build/debug scenarios are enclosed with the product. As compared to the existing CodeWarrior products, the New Project wizard functionality in CodeWarrior for ARMv8 has been refined to generating copies of the existing pre-configured projects.

All the debugger connection settings are refactored in the Target Connection Configuration dialog.

The ARMv8 New Project wizard enables you to create both bareboard and Linux Application projects. To access the ARMv8 New Project wizard, in the Workbench window, select **File > New > ARMv8 Stationary**.



ARMv8 Project

ARMv8 Project

Create an ARMv8 project

Project name:

☑ Use default location

Location: C:/Users/b14174/workspace

Available stationaries:

type filter text

- ARMv8
  - Bareboard
    - Hello World Assembly Project
    - Hello World C Project
    - Hello World C Static Library Project
    - Hello World C++ Project
    - Hello World C++ Static Library Project
  - Linux Application Debug
    - Hello World C Project
    - Hello World C Static Library Project
    - Hello World C Shared Library Project
    - Hello World C++ Project
    - Hello World C++ Static Library Project

Description:

Creates a simple Assembly project for bareboard. The program executes forever in a while loop.

Figure 2-1. ARMv8 Project wizard

The table lists and explains the ARMv8 New Project wizard options.

Table 2-1. ARMv8 New Project wizard options

Option	Description
Project name	Enter the name for the new project in this text box. **Note:** Do not use the reserved/special characters/symbols such as < (less than), > (greater than), : (colon), " (double quote), / (forward slash), \ (backslash), | (vertical bar or pipe), ? (question mark), @ (at), * (asterisk) in the project name. The special characters/symbols in the project name may result in an unexpected behavior.

Table continues on the next page...

Table 2-1. ARMv8 New Project wizard options (continued)

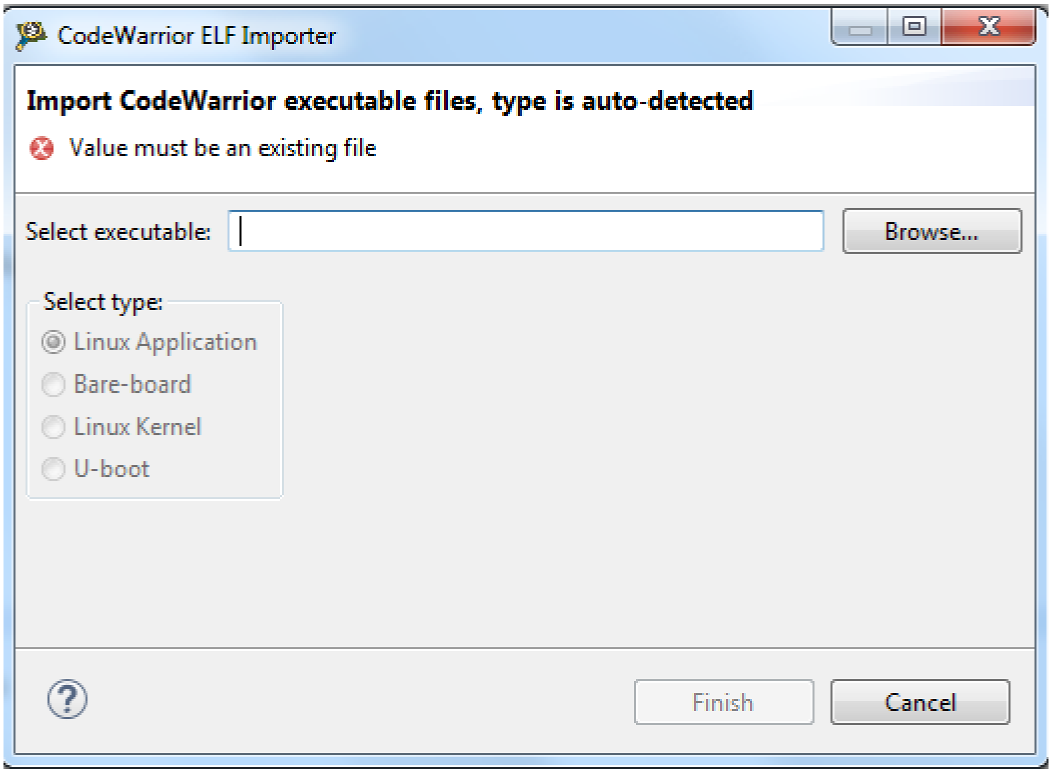
Option	Description
Use default location	Stores the files required to build the program in the current workspace directory. The project files are stored in the default location. **Clear the Use default location** checkbox and click **Browse** to select a new location.
Location	Specifies the directory that contains the project files. Click Browse to navigate to the desired directory. This option is available only when **Use default location** checkbox is clear.
Available Stationaries	List the various stationaries available for you to create a project. The stationaries are categorized under: Bareboard and Linux Application Debug.

2.2 CodeWarrior ELF Importer wizard

The CodeWarrior ELF Importer wizard allows users to import CodeWarrior ELF images of various types.

- Linux Application
- Bare-board
- Linux Kernel
- U-boot

You can access the wizard from **File > New > CodeWarrior ELF Importer**.



CodeWarrior ELF Importer

Import CodeWarrior executable files, type is auto-detected

Value must be an existing file

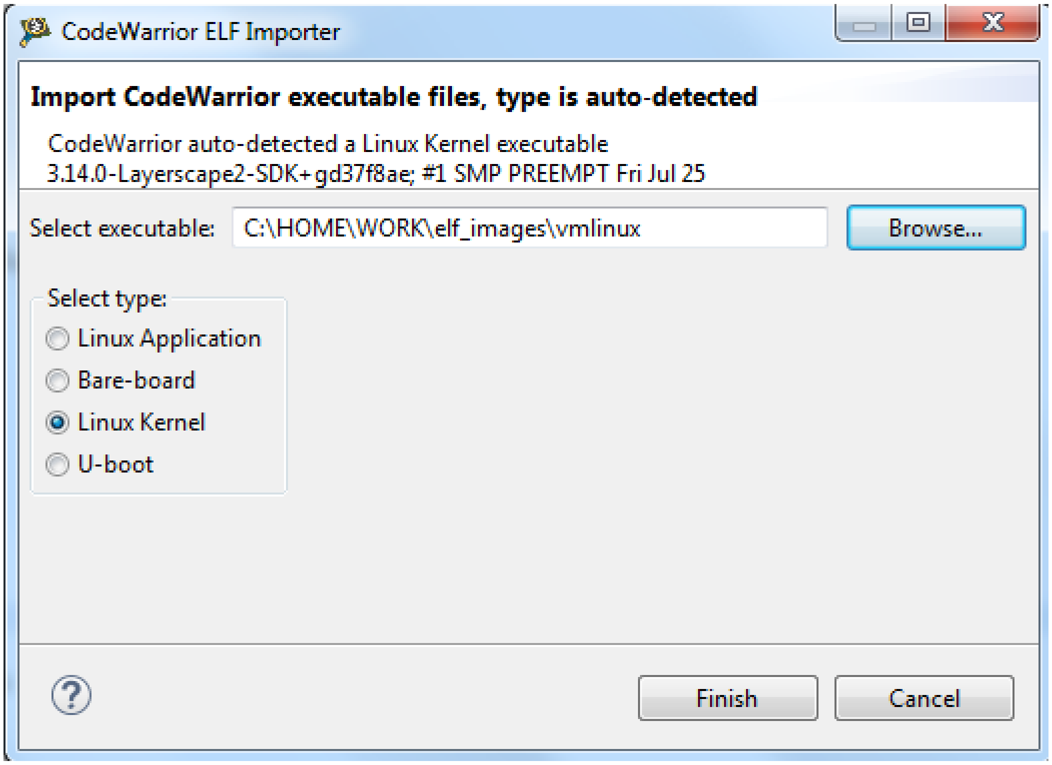
Select executable:

Select type:

● Linux Application
○ Bare-board
○ Linux Kernel
○ U-boot

Figure 2-2. CodeWarrior ELF Importer, Select executable

Once the executable is selected the image type is auto-detected based on the symbol table. The user can overwrite the value by selecting another type.



CodeWarrior ELF Importer

Import CodeWarrior executable files, type is auto-detected

CodeWarrior auto-detected a Linux Kernel executable
3.14.0-Layerscape2-SDK+gd37f8ae; #1 SMP PREEMPT Fri Jul 25

Select executable: C:\HOME\WORK\elf_images\vmlinux

Select type:
- ○ Linux Application
- ○ Bare-board
- ● Linux Kernel
- ○ U-boot

Figure 2-3. CodeWarrior ELF Importer, type is auto-detected

An error message is displayed and the user is not allowed to finish the project creation if the selected executable is not a binary or doesn't have the ELF format.

The created project contains the ELF image as a linked resource and also a default launch configuration file with all the setup ready to debug.

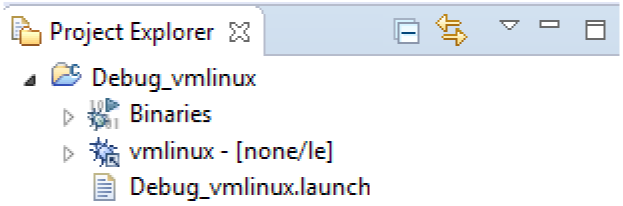


Figure 2-4. Project creation with CodeWarrior ELF Importer wizard

The user only needs to open the launch configuration file, review/change the settings, and start the debug session.

2.3 Creating projects

This section explains how to use the ARMv8 New Project wizard to quickly create new projects with default settings, build and launch configurations.

The section explains:

- Creating CodeWarrior Bareboard project
- Creating CodeWarrior Linux Application project

2.3.1 Creating CodeWarrior Bareboard project

You can create a CodeWarrior Bareboard project using the ARMv8 Stationary wizard.

1. From CodeWarrior IDE menu bar, select **File > New > ARMv8 Stationary**
2. From **Available stationaries**, select **ARMv8 > Bare board > Hello World C Project**.
3. In **Project name** text box, enter `FirstProjectTest`.

NOTE

The **Location** text box shows the default workspace location. To change this location, uncheck the **Use default location** text box and click **Browse** to select a new location.

4. Click **Finish**.

   The new project appears in the **Project Explorer** view.

NOTE

Before you build and debug the project, ensure that the target board is ready. For details, see Preparing target.

5. Build the bare metal project.
6. Debug the bare metal project. Refer Debugging Bareboard project.

You can create a CodeWarrior Bareboard project for following configurations:

- Assembly Project
- C Project
- C Static Library Project
- C++ Project
- C++ Static Library Project

2.3.2 Creating CodeWarrior Linux Application project

You can create a CodeWarrior Bareboard project using the ARMv8 Stationary wizard.

1. From CodeWarrior IDE menu bar, select **File > New > ARMv8 Stationary**.
2. From **Available stationaries**, select **ARMv8 > Linux Application Debug > Hello World C Project**.
3. In **Project name** text box, enter `FirstLinuxProject`.

NOTE

The **Location** text box shows the default workspace location. To change this location, uncheck the **Use default location** text box and click **Browse** to select a new location.

4. Click **Finish**.

   The new project appears in the **Project Explorer** view.

NOTE

Before you build and debug the project, ensure that the target board is ready. For details, see Preparing target.

5. Build the Linux application project.
6. Debug the Linux application project. Refer Debugging projects

You can create a Linux application project for following configurations:

- C Project
- C Static Library Project
- C Shared Library Project
- C++ Project
- C++ Static Library Project

For further details about application debug projects, refer Linux Application Debug.

2.4 Preprocess/Disassemble files

You can access the Preprocess/Disassemble commands from the **Project Explorer** or **Editor** view.

The Preprocess/Disassemble commands are available to the user:

- from the menu that appears when you right-click on a file in the **Project Explorer** view, or
- from the menu that appears when you open the file in the **Editor** view and right-click inside the **Editor** view.

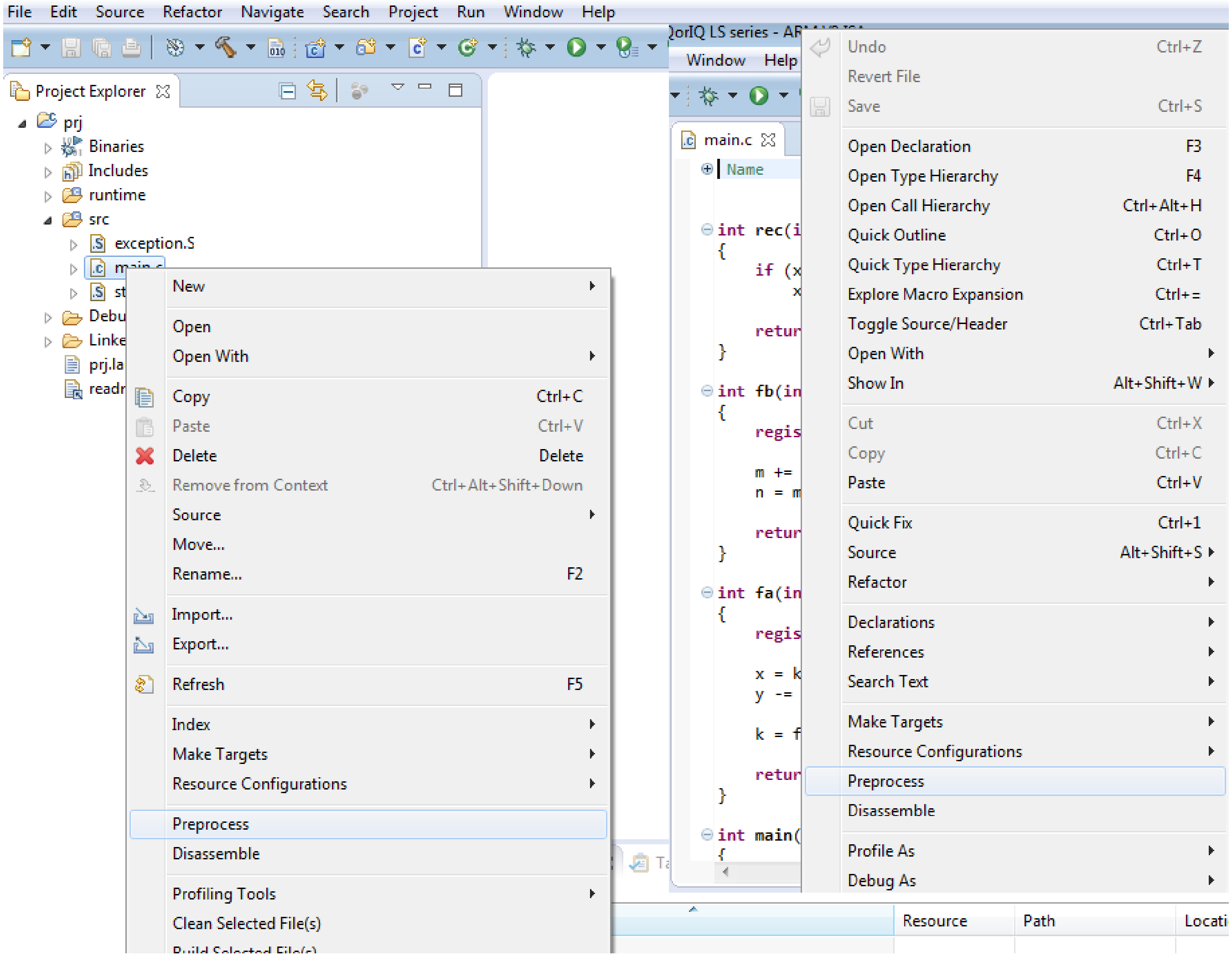
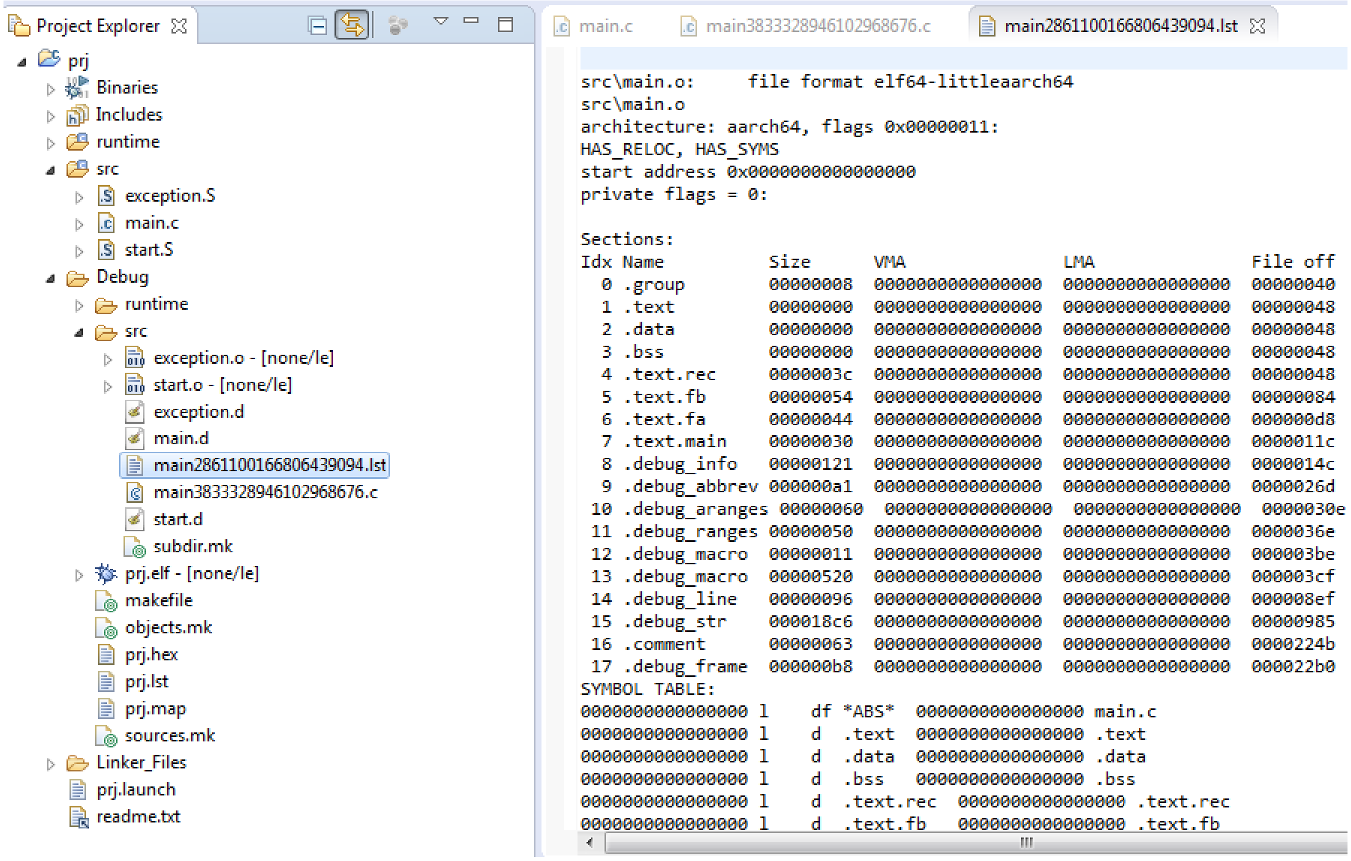


Figure 2-5. Project Explorer view and Editor view

The result of preprocessing a file or disassembling an object code is provided to the user in the Editor. Upon invocation, the Preprocess command preprocesses the C/C++/ASM file and shows the resulting text in a new file. Similarly, upon invocation, the Disassemble command compiles and disassembles the C/C++/ASM file or directly disassembles the binary file. In all the cases, the resulted files are located in the active configuration directory.

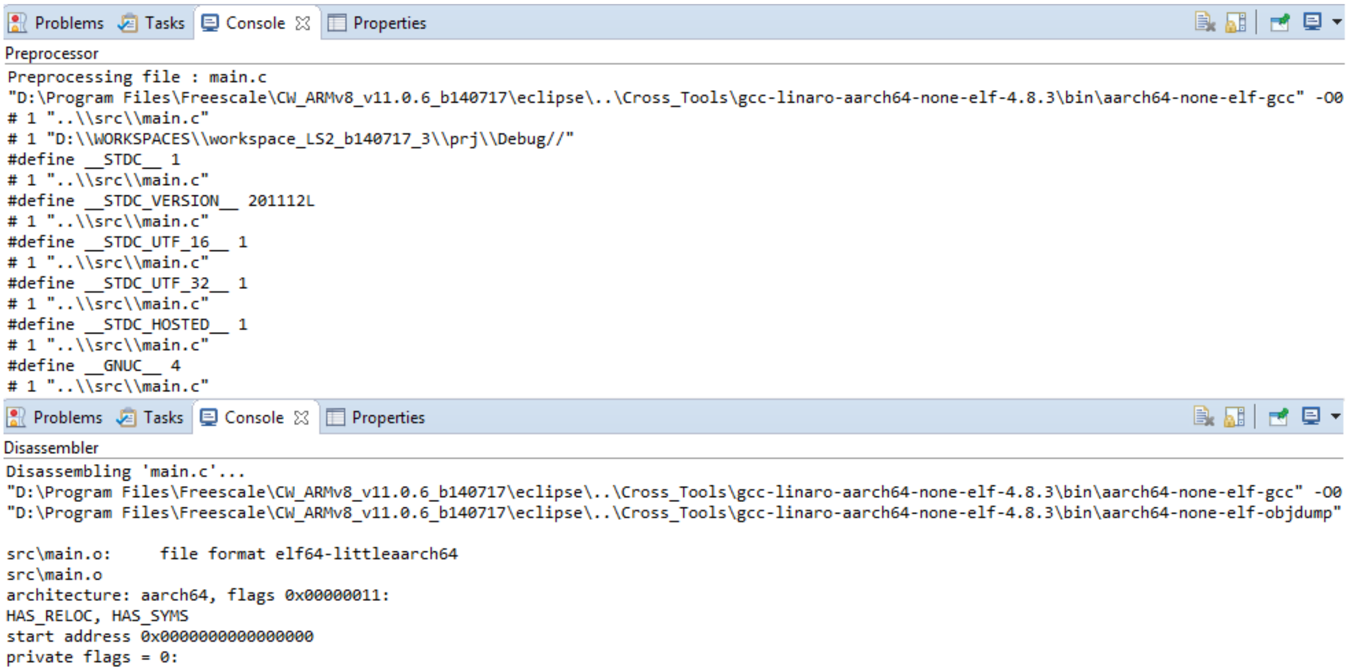


```
src\main.o:     file format elf64-littleaarch64
src\main.o
architecture: aarch64, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000
private flags = 0:

Sections:
Idx Name          Size      VMA               LMA               File off
  0 .group        00000008  0000000000000000  0000000000000000  00000040
  1 .text         00000000  0000000000000000  0000000000000000  00000048
  2 .data         00000000  0000000000000000  0000000000000000  00000048
  3 .bss          00000000  0000000000000000  0000000000000000  00000048
  4 .text.rec     0000003c  0000000000000000  0000000000000000  00000048
  5 .text.fb      00000054  0000000000000000  0000000000000000  00000084
  6 .text.fa      00000044  0000000000000000  0000000000000000  000000d8
  7 .text.main    00000030  0000000000000000  0000000000000000  0000011c
  8 .debug_info   00000121  0000000000000000  0000000000000000  0000014c
  9 .debug_abbrev 000000a1  0000000000000000  0000000000000000  0000026d
 10 .debug_aranges 00000060  0000000000000000  0000000000000000  0000030e
 11 .debug_ranges 00000050  0000000000000000  0000000000000000  0000036e
 12 .debug_macro  00000011  0000000000000000  0000000000000000  000003be
 13 .debug_macro  00000520  0000000000000000  0000000000000000  000003cf
 14 .debug_line   00000096  0000000000000000  0000000000000000  000008ef
 15 .debug_str    000018c6  0000000000000000  0000000000000000  00000985
 16 .comment      00000063  0000000000000000  0000000000000000  0000224b
 17 .debug_frame  000000b8  0000000000000000  0000000000000000  000022b0
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 main.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .text.rec  0000000000000000 .text.rec
0000000000000000 l    d  .text.fb   0000000000000000 .text.fb
```

Figure 2-6. Editor view

NOTE

A new Console is created for each operation.



Preprocessor

```
Preprocessing file : main.c
"D:\Program Files\Freescale\CW_ARMv8_v11.0.6_b140717\eclipse\..\Cross_Tools\gcc-linaro-aarch64-none-elf-4.8.3\bin\aarch64-none-elf-gcc" -O0
# 1 "..\\src\\main.c"
# 1 "D:\\WORKSPACES\\workspace_LS2_b140717_3\\prj\\Debug//"
#define __STDC__ 1
# 1 "..\\src\\main.c"
#define __STDC_VERSION__ 201112L
# 1 "..\\src\\main.c"
#define __STDC_UTF_16__ 1
# 1 "..\\src\\main.c"
#define __STDC_UTF_32__ 1
# 1 "..\\src\\main.c"
#define __STDC_HOSTED__ 1
# 1 "..\\src\\main.c"
#define __GNUC__ 4
# 1 "..\\src\\main.c"
```

Disassembler

```
Disassembling 'main.c'...
"D:\Program Files\Freescale\CW_ARMv8_v11.0.6_b140717\eclipse\..\Cross_Tools\gcc-linaro-aarch64-none-elf-4.8.3\bin\aarch64-none-elf-gcc" -O0
"D:\Program Files\Freescale\CW_ARMv8_v11.0.6_b140717\eclipse\..\Cross_Tools\gcc-linaro-aarch64-none-elf-4.8.3\bin\aarch64-none-elf-objdump"

src\main.o:     file format elf64-littleaarch64
src\main.o
architecture: aarch64, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000
private flags = 0:
```

Figure 2-7. Console view

The user can define or modify preprocessor/disassembler options in the **Project Properties** dialog > **Settings > Tool Settings** page.

2.5 Debugging projects

When you use the ARMv8 Project wizard to create a new project, the 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 debug a project:

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

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

The ARMv8 Project wizard adds a default launch configuration in all application sample projects. The debugger settings are mapped to the default values but you can change these values based on your requirements.

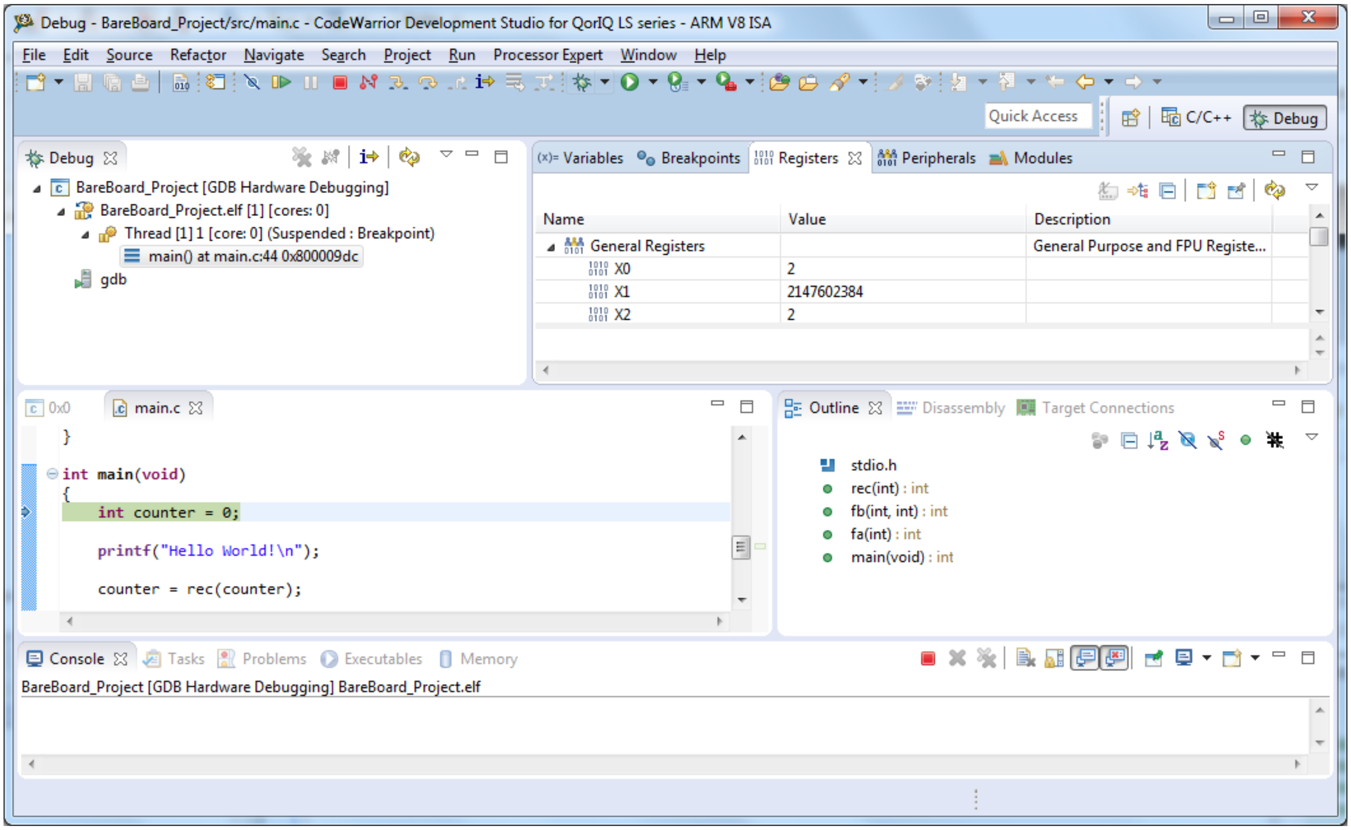
2.5.1 Debugging Bareboard project

This topic describes how to debug a bareboard project.

Ensure that the project contains the default launch configuration file of type GDB Hardware Debugging, named as <projectName>.launch. To start debugging a project:

1. In the **Debug Configuration** dialog, select the available launch configuration.
2. Select a Target Connection Configurator. For details on this, refer Target Connection configurator overview and Configure the target configuration using Target Connection Configurator
3. Click **Apply** in the **Debug Configurations** dialog. The IDE saves your settings.
4. Click **Debug**.

   The IDE switches to the **Debug** perspective. The debugger downloads your program to the target board and halts execution at the first statement of main().


Debug - BareBoard_Project/src/main.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Processor Expert Window Help
Quick Access
C/C++
Debug
Debug
Variables
Breakpoints
Registers
Peripherals
Modules
0x0
main.c
Outline
Disassembly
Target Connections
Console
Tasks
Problems
Executables
Memory

```
BareBoard_Project [GDB Hardware Debugging]
  BareBoard_Project.elf [1] [cores: 0]
    Thread [1] 1 [core: 0] (Suspended : Breakpoint)
      main() at main.c:44 0x800009dc
  gdb
```

Name	Value	Description
General Registers		General Purpose and FPU Registe...
X0	2	
X1	2147602384	
X2	2	

```
}

int main(void)
{
    int counter = 0;

    printf("Hello World!\n");

    counter = rec(counter);
```

```
stdio.h
rec(int) : int
fb(int, int) : int
fa(int) : int
main(void) : int
```

```
BareBoard_Project [GDB Hardware Debugging] BareBoard_Project.elf
```

Figure 2-8. Debugging bareboard project

2.5.2 Debugging Linux Application project

This topic describes how to debug a Linux application project.

Ensure that the project contains the default launch configuration file of type C/C++ Remote Application, named as <projectName>.launch.

The CodeWarrior software creates a default **ssh with scp** connection, named **ScpConnection**, when it is opened for the first time. This connection is available in the **Remote Systems** view. The default launch configuration file used in a Linux Application debug project points to this connection. The user can change the default settings, for example the IP of the Linux target.

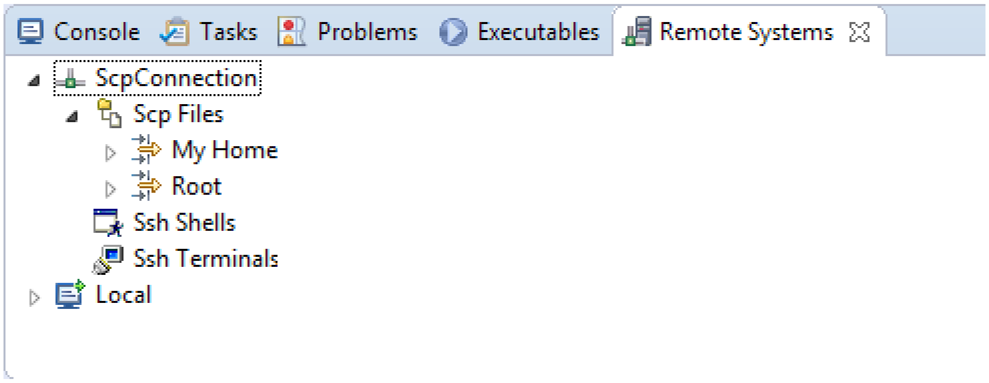


Figure 2-9. Scp Connection

To start debugging a project:

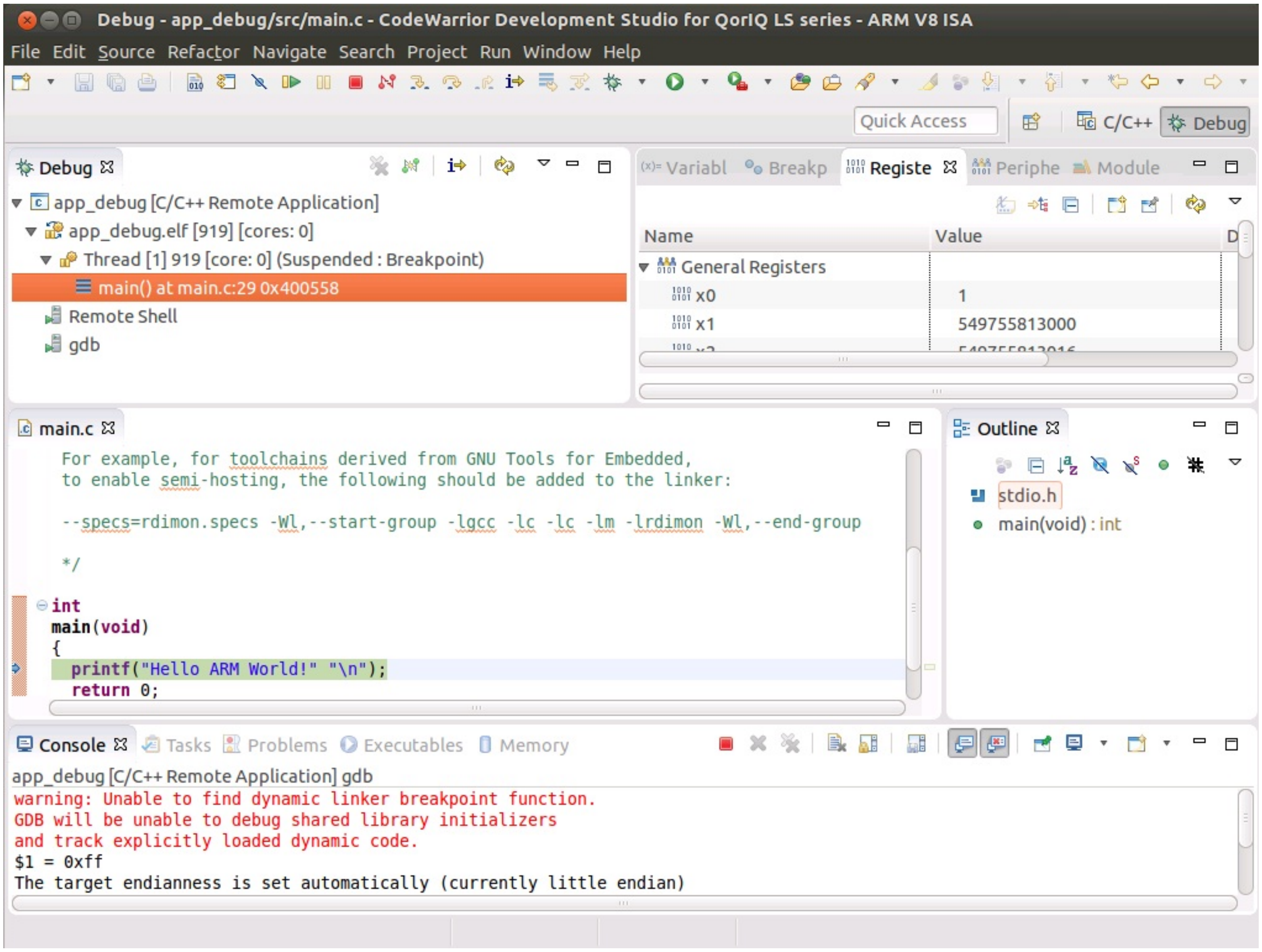
NOTE

If target is accessible on a port different than the default 22, like in the case of the ssh tunnelling to other port, the tunelling port should be specified instead.

1. In the **Debug Configuration** dialog, select the available launch configuration.
2. Click **Debug**.

NOTE

For further details, refer Linux Application Debug.


Debug - app_debug/src/main.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
C/C++
Debug
Debug
Variabl
Breakp
Registe
Periphe
Module
main.c
Outline
stdio.h
main(void) : int
Console
Tasks
Problems
Executables
Memory

```
app_debug [C/C++ Remote Application]
  app_debug.elf [919] [cores: 0]
    Thread [1] 919 [core: 0] (Suspended : Breakpoint)
      main() at main.c:29 0x400558
  Remote Shell
  gdb
```

Name	Value
General Registers	
x0	1
x1	549755813000

```
    For example, for toolchains derived from GNU Tools for Embedded,
    to enable semi-hosting, the following should be added to the linker:

    --specs=rdimon.specs -Wl,--start-group -lgcc -lc -lc -lm -lrdimon -Wl,--end-group

    */

int
main(void)
{
  printf("Hello ARM World!" "\n");
  return 0;
```

```
app_debug [C/C++ Remote Application] gdb
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
$1 = 0xff
The target endianness is set automatically (currently little endian)
```

Figure 2-10. Debugging Linux Application project

Chapter 3
ARMv8 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.

For details about how ARMv8 projects are managed and all the available toolchains, refer ARM GNU Eclipse documentation available at: http://gnuarmeclipse.livius.net/blog/documentation

NOTE

Freescale Semiconductor, Inc. does not own ARM GNU Eclipse documentation. The documents are mentioned solely for the reference purpose.

3.1 Changing Build Properties

The New Bareboard Project wizard creates a set of build properties for the project.

You can modify these build properties to better suit your needs.

Perform these steps to change build properties:

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

The **Properties** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.

4. Expand the **C/C++ Build** property.
5. Select **Settings**.

   The **Properties** window shows the corresponding build properties.

6. Use the Configuration drop-down list to specify the launch configuration for which you want to modify the build properties.
7. Click the **Tool Settings** tab.

   The corresponding page appears.

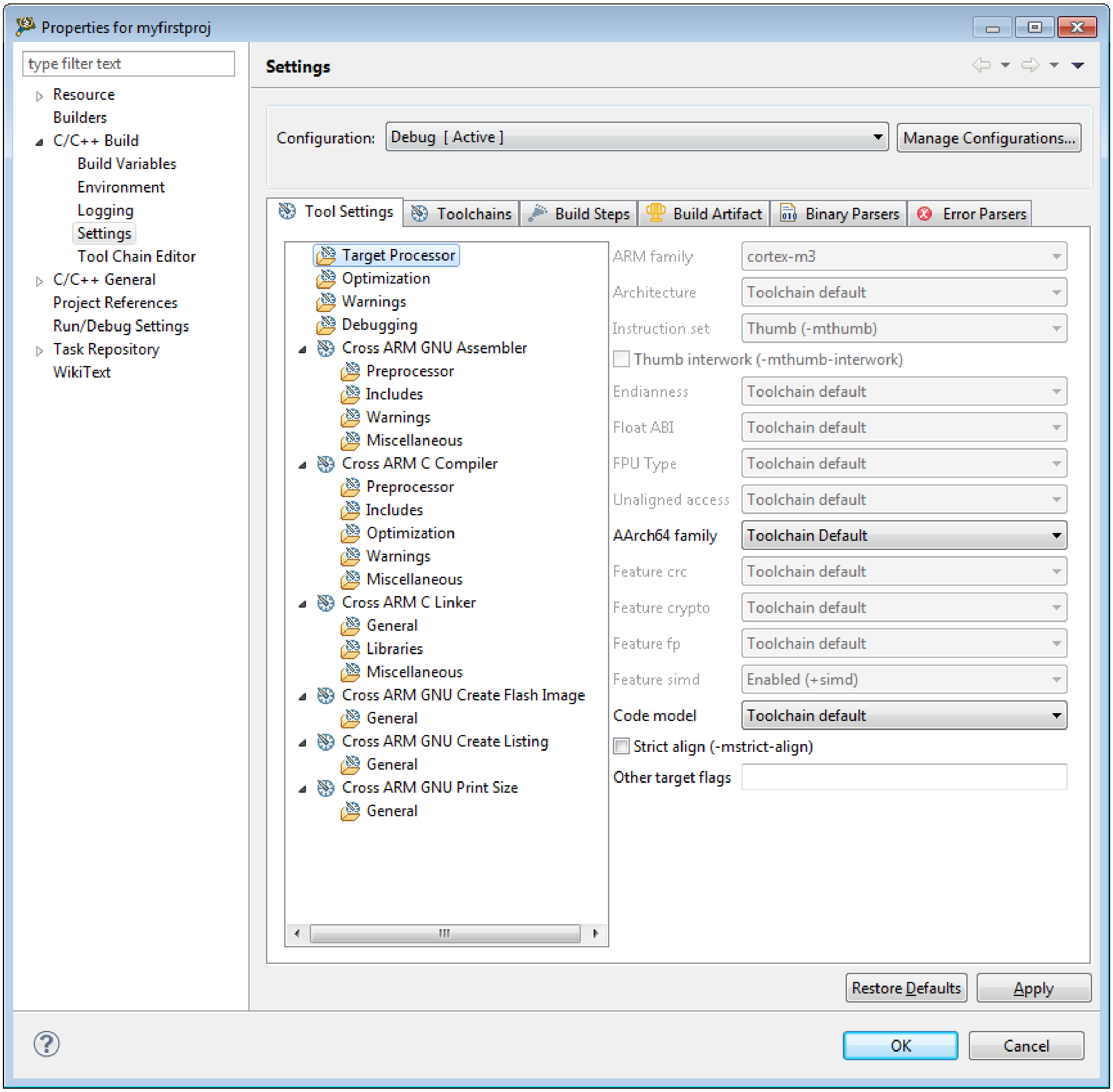
8. From the list of tools on the **Tool Settings** page, select the tool for which you want to modify properties.
9. Change the settings that appear in the page.
10. Click **Apply**.

    The IDE saves your new settings.

You can select other tool pages and modify their settings. When you finish, click OK to save your changes and close the **Properties** window.

3.2 ARMv8 build settings

The **Properties for** <*project*> window shows the corresponding Settings page for a project.



Configuration:	Debug [Active]
ARM family	cortex-m3
Architecture	Toolchain default
Instruction set	Thumb (-mthumb)
☐ Thumb interwork (-mthumb-interwork)	
Endianness	Toolchain default
Float ABI	Toolchain default
FPU Type	Toolchain default
Unaligned access	Toolchain default
AArch64 family	Toolchain Default
Feature crc	Toolchain default
Feature crypto	Toolchain default
Feature fp	Toolchain default
Feature simd	Enabled (+simd)
Code model	Toolchain default
☐ Strict align (-mstrict-align)	
Other target flags	

Figure 3-1. Settings page

The following table lists the build properties specific to developing software for ARM Embedded Processors.

The properties that you specify in the **Tool Settings** panels apply to the selected build tool on the **Tool Settings** page of the **Properties for** <*project*> dialog box.

Table 3-1. Build Properties for Bare Metal project

Tool Settings	Sub Tool Settings
Target Processor	Target Processor
Optimization	Optimization
Warnings	Warnings
Debugging	Debugging
Cross ARM GNU Assembler	Preprocessor
	Includes
	Warnings
	Miscellaneous
Cross ARM C Compiler	Preprocessor
	Includes
	Optimization
	Warnings
	Miscellaneous
Cross ARM C Linker	General
	Libraries
	Miscellaneous
Cross ARM GNU Create Flash Image	General
Cross ARM GNU Create Listing	General
Cross ARM GNU Print Size	General

3.2.1 Target Processor

Use this panel to configure the target processor options.

The following table lists the options in the **Target Processor** panel.

Table 3-2. Target Processor options

Option	Description
ARM family	Use to specify the ARM family name. Default: cortex-m3
Architecture	Use to specify the target hardware architecture or processor name. The compiler can take advantage of the extra instructions that the selected architecture provides and optimize the code to run on a specific processor. The inline assembler might display error messages or warnings if it assembles some processor-specific instructions for the wrong target architecture. Default: Toolchain default

Table continues on the next page...

Table 3-2. Target Processor options (continued)

Option	Description
Instruction set	Use to generate suitable interworking veneers when it links the assembler output. You must enable this option if you write ARM code that you want to interwork with Thumb code or vice versa. The only functions that need to be compiled for interworking are the functions that are called from the other state. You must ensure that your code uses the correct interworking return instructions. Default: Thumb (-mthumb)
Thumb interwork (-mthumb-interwork)	Check to have the processor generate Thumb code instructions. Clear to prevent the processor from generating Thumb code instructions. The IDE enables this setting only for architectures and processors that support the Thumb instruction set. Default: Clear
Endianness	Use to specify the byte order of the target hardware architecture: • Little-little endian; right-most bytes (those with a higher address) are most significant • Big-big endian; left-most bytes (those with a lower address) are most significant Default: Toolchain default
Float ABI	Use to specify the float Application Binary Interface (ABI). Default: Toolchain default
FPU Type	Use to specify the type of floating-point unit (FPU) for the target hardware architecture: The assembler might display error messages or warnings if the selected FPU architecture is not compatible with the target architecture. Default: Toolchain default
Unaligned access	Use to specify unaligned access. Default: Toolchain default
AArch64 family	Use to specify the architecture family: • Generic (-mcpu=generic) • Large (-mcpu=large) • Toolchain default Default: Toolchain default
Feature crc	Use to specify Feature crc.
Feature crypto	Use to specify Feature crypts.
Feature fp	Use to specify Feature fp.
Feature simd	Use to specify Feature simd.
Code model	Specifies the addressing mode that the linker uses when resolving references. This setting is equivalent to specifying the -mcmodel keyword command-line option. • Tiny (-mcmdel=tiny) • Small (-mcmodel=small) • Large (-mcmodel=large) • Toolchain default
Strict align (-mstrict-align)	Controls the use of non-standard ISO/IEC 9899-1990 ("C90") language features.
Other target flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.2.2 Optimization

Use this panel to configure the optimization options.

The following table lists the options in the **Optimization** panel.

Table 3-3. Optimization options

Option	Description
Optimization level	Specify the optimizations that you want the compiler to apply to the generated object code: • None (-O0)-Disable optimizations. This setting is equivalent to specifying the -O0 command-line option. The compiler generates unoptimized, linear assembly-language code. • Optimize (-O1)-The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. This setting is equivalent to specifying the -O1 command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. • Optimize more (-O2)-The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the -O2 command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. • Optimize most (-O3)-The compiler performs all the level 2 optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the that is usually faster than the code generated from level 2 optimizations. • Optimize size (-Os)-The compiler optimizes object code at the specified Optimization Level such that the resulting binary file has a smaller executable code size, as opposed to a faster execution speed. This setting is equivalent to specifying the -Os command-line option. • Optimize for debugging (-Og)-The compiler optimizes object code at the specified Optimization Level such that the resulting binary file has a faster execution speed, as opposed to a smaller executable code size.
Message length (-fmessage-length=0)	Check if you want to specify the maximum length in bytes for the message.
'char' is signed (-fsigned-char)	Check to treat char declarations as signed char declarations.
Function sections (-ffunction-sections)	Check to enable function sections.
Data sections (-fdata-sections)	Check to enable data sections.
No common unitialized (-fno-common)	Controls the placement of uninitialized global variables.
Do not inline functions (-fno-inline-functions)	Suppresses automatic inlining of subprograms.
Assume freestanding environment (-ffeestanding)	Asserts that compilation takes place in a freestanding environment. This implies -fno-builtin.
Disable builtin (-fno-builtin)	Switches off builtin functions.
Single precision constants (-fsingle-precision-constant)	Check to enable single precision constants.
Position independent code (-fPIC)	Select to instruct the build tools to generate position independent-code.
Other optimization flags	Specify additional command line options; type in custom optimization flags that are not otherwise available in the UI.

3.2.3 Warnings

Use this panel to configure the warning options.

The following table lists the options in the **Warnings** panel.

Table 3-4. Warnings options

Option	Description
Check syntax only (-fsyntax-only)	Check this option if you want to check the syntax of commands and throw a syntax error.
Pedantic (-pedantic)	Check if you want warnings like -pedantic, except that errors are produced rather than warnings.
Pedantic warnings as errors (-pedantic-errors)	Check this option if you want to inhibit the display of warning messages.
Inhibit all warnings (-w)	Check this option if you want to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Warn on various unused elements (-Wunused)	Warn whenever some element (label, parameter, function, etc.) is unused.
Warn on uninitialized variables (-Wuninitialised)	Warn whenever an automatic variable is used without first being initialized.
Enable all common warnings (-Wall)	Check this option if you want to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Enable extra warnings (-Wextra)	Check this option to enable any extra warnings.
Warn on undeclared global function (-Wmissing-declaration)	Check to warn if an undeclared global function is encountered.
Warn on implicit conversions (-Wconversion)	Check to warn of implicit conversions.
Warn if pointer arithmetic (-Wpointer-arith)	Check to warn if pointer arithmetic are used.
Warn if padding is not included (-Wpadded)	Check to warn if padding is included in a structure either to align an element of the structure or the whole structure.
Warn if shadowed variable (-Wshadow)	Check to warn if shadowed variable are used.
Warn if suspicious logical ops (-Wlogical-op)	Check to warn in case of suspicious logical operation.
Warn in struct is returned (-Wagreggrate-return)	Check to warn if struct is returned.
Warn if floats are compared as equal (-Wfloat-equal)	Check to warn if floats are compared as equal.
Generate errors instead of warnings (-Werror)	Check to generate errors instead of warnings.
Other warning flags	Specify additional command line options; type in custom warning flags that are not otherwise available in the UI.

3.2.4 Debugging

Use this panel to configure the debugging options.

The following table lists the options in the **Debugging** panel.

Table 3-5. Debugging options

Option	Description
Debug level	Specify the debug levels: • None - No Debug level. • Minimal (-g1) - The compiler provides minimal debugging support. • Default (-g) - The compiler generates DWARF 1.xconforming debugging information. • Maximum (-g3) - The compiler provides maximum debugging support.
Debug format	Specify the debug formats for the compiler.
Generate prof information (-p)	Generates extra code to write profile information suitable for the analysis program prof. You must use this option when compiling the source files you want data about, and you must also use it when linking.
Generate gprof information (-pg)	Generates extra code to write profile information suitable for the analysis program gprof. You must use this option when compiling the source files you want data about, and you must also use it when linking.
Other debugging flags	Specify additional command line options; type in custom debugging flags that are not otherwise available in the UI.

3.2.5 Cross ARM GNU Assembler

Use this panel to configure the ARM GNU assembler options.

The following table lists the options in the **Cross ARM GNU Assembler** panel.

Table 3-6. Cross ARM GNU Assembler options

Option	Description
Command	Shows the location of the assembler executable file. Default: `${cross_prefix}${cross_c}${cross_suffix}`
All Options	Shows the actual command line the assembler will be called with. Default: `-x assembler-with-cpp -Xassembler -g`
Expert settings	
Command line pattern	Shows the expert settings command line parameters. Default: `${COMMAND} ${cross_toolchain_flags} ${FLAGS} -c ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

3.2.5.1 Preprocessor

Use this panel to configure the ARM GNU assembler preprocessor options.

The following table lists the options in the **Cross ARM GNU Assembler Preprocessor** panel.

Table 3-7. Cross ARM GNU Assembler Preprocessor options

Option	Description
Use preprocessor	Check this option to use the preprocessor for the assembler.
Do not search system directories (-nostdinc)	Check this option if you do not want the assembler to search the system directories. By default, this checkbox is clear. The assembler performs a full search that includes the system directories.
Preprocess only (-E)	Check this option if you want the assembler to preprocess source files and not to run the compiler. By default, this checkbox is clear and the source files are not preprocessed.
Defined symbols (-D)	Use this option to specify the substitution strings that the assembler applies to all the assembly-language modules in the build target. Enter just the string portion of a substitution string. The IDE prepends the -D token to each string that you enter. For example, entering opt1 x produces this result on the command line: -Dopt1 x. Note: This option is similar to the DEFINE directive, but applies to all assembly-language modules in a build target.
Undefined symbols (-U)	Undefines the substitution strings you specify in this panel.

3.2.5.2 Includes

Use this panel to configure the ARM GNU assembler includes options.

The following table lists the options in the Cross ARM GNU Assembler Includes panel.

Table 3-8. Cross ARM GNU Assembler Includes options

Option	Description
Include paths (-l)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search #include files in several different ways. You can also set the search order as follows: For include statements of the form #include"xyz", the compiler first searches user paths, then the system paths For include statements of the form #include<xyz>, the compiler searches only system paths This option is global.
Include files (-include)	Use this option to specify the include file search path.

3.2.5.3 Warnings

Use this panel to configure the ARM GNU assembler warning options.

The following table lists the options in the **Cross ARM GNU Assembler Warnings** panel.

Table 3-9. Warnings options

Option	Description
Other warning flags	Specify additional command line options; type in custom warning flags that are not otherwise available in the UI.

3.2.5.4 Miscellaneous

Use this panel to configure the ARM GNU assembler miscellaneous options.

The following table lists the options in the **Cross ARM GNU Assembler Miscellaneous** panel.

Table 3-10. Cross ARM GNU Assembler Miscellaneous options

Option	Description
Assembler flags	Specify the flags that need to be passed with the assembler.
Generates assembler listing (-Wa, -adhlns="$@.lst")	Enables the assembler to create a listing file as it compiles assembly language into object code.
Save temporary files (--save-temps Use with caution!)	Store the usual “temporary” intermediate files permanently.
Verbose (-v)	Check this option if you want the IDE to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the -v command-line option. By default this checkbox is clear. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages.
Other assembler flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.2.6 Cross ARM C Compiler

Use this panel to configure the ARM C compiler options.

The following table lists the options in the **Cross ARM C Compiler** panel.

Table 3-11. Cross ARM C Compiler options

Option	Description
Command	Shows the location of the compiler executable file. Default: `${cross_prefix}${cross_c}${cross_suffix}`

Table continues on the next page...

Table 3-11. Cross ARM C Compiler options (continued)

Option	Description
All Options	Shows the actual command line the compiler will be called with.
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: `${COMMAND} ${cross_toolchain_flags} ${FLAGS} -c ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

3.2.6.1 Preprocessor

Use this panel to configure the ARM C compiler preprocessor options.

The following table lists the options in the **Cross ARM C Compiler Preprocessor** panel.

Table 3-12. Cross ARM GNU compiler Preprocessor options

Option	Description
Use preprocessor	Check this option to use the preprocessor for the compiler.
Do not search system directories (-nostdinc)	Check this option if you do not want the compiler to search the system directories. By default, this checkbox is clear. The compiler performs a full search that includes the system directories.
Preprocess only (-E)	Check this option if you want the compiler to preprocess source files and not to run the compiler. By default, this checkbox is clear and the source files are not preprocessed.
Defined symbols (-D)	Use this option to specify the substitution strings that the compiler applies modules in the build target. Enter just the string portion of a substitution string. The IDE prepends the -D token to each string that you enter. For example, entering opt1 x produces this result on the command line: -Dopt1 x. Note: This option is similar to the DEFINE directive, but applies to all assembly-language modules in a build target.
Undefined symbols (-U)	Undefines the substitution strings you specify in this panel.

3.2.6.2 Includes

Use this panel to configure the ARM C compiler includes options.

The following table lists the options in the Cross ARM C Compiler Includes panel.

Table 3-13. Cross ARM C Compiler Includes options

Option	Description
Include paths (-l)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search #include files in several different ways. You can also set the search order as follows: For include statements of the form #include"xyz", the compiler first searches user paths, then the system paths For include statements of the form #include<xyz>, the compiler searches only system paths This option is global.
Include files (-include)	Use this option to specify the include file search path.

3.2.6.3 Optimization

Use this panel to configure the ARM C compiler optimization options.

The following table lists the options in the **Optimization** panel.

Table 3-14. Optimization options

Option	Description
Language standard	Select the programming language or standard to which the compiler should conform. • ISO C90 (-ansi) - Select this option to compile code written in ANSI standard C. The compiler does not enforce strict standards. For example, your code can contain some minor extensions, such as C++ style comments (//), and $ characters in identifiers. • ISO C99 (-std=c99) - Select this option to instruct the compiler to enforce stricter adherence to the ANSI/ISO standard. • Compiler Default (ISO C90 with GNU extensions) - Select this option to enforce adherence to ISO C90 with GNU extensions. • ISO C99 with GNU Extensions (-std=gnu99)
Other optimization flags	Specify additional command line options; type in custom optimization flags that are not otherwise available in the UI.

3.2.6.4 Warnings

Use this panel to configure the ARM C compiler warnings options.

The following table lists the options in the **Warnings** panel.

Table 3-15. Warnings options

Option	Description
Warn if a global function has no prototype (-Wmissing-prototype)	Warn if a global function has no prototype.

Table continues on the next page...

Table 3-15. Warnings options (continued)

Option	Description
Warn if a function has no arg type (-Wstrict-prototypes)	Warn if a function is declared or defined without specifying the argument types.
Warn if a wrong cast (-Wbad-function-cast)	Warn whenever a function call is cast to a non-matching type.
Other warning flags	Specify additional command line options; type in custom warning flags that are not otherwise available in the UI.

3.2.6.5 Miscellaneous

Use this panel to configure the ARM C compiler miscellaneous options.

The following table lists the options in the **Miscellaneous** panel.

Table 3-16. Miscellaneous options

Option	Description
Generates assembler listing (-Wa, -adhlns="$@.lst")	Enables the assembler to create a listing file as it compiles assembly language into object code.
Save temporary files (--save-temps Use with caution!)	Store the usual “temporary” intermediate files permanently.
Verbose (-v)	Check this option if you want the IDE to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the -v command-line option. By default this checkbox is clear. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages.
Other compiler flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.2.7 Cross ARM C Linker

Use this panel to configure the ARM C linker options.

The following table lists the options in the **Cross ARM C Linker** panel.

Table 3-17. Cross ARM C Linker options

Option	Description
Command	Shows the location of the linker executable file. Default: `${cross_prefix}${cross_c}${cross_suffix}`

Table continues on the next page...

Table 3-17. Cross ARM C Linker options (continued)

Option	Description
All Options	Shows the actual command line the assembler will be called with. Default: `-T "${ProjDirPath}"/Linker_Files/aarch64elf.x -nostartfiles -nodefaultlibs -L"C:\Users\b14174\workspace-15\FirstProjectTest" -Wl,-Map,"FirstProjectTest.map"`
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: `${COMMAND} ${cross_toolchain_flags} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

3.2.7.1 General

Use this panel to configure the ARM C linker general options.

The following table lists the options in the General panel.

Table 3-18. General options

Option	Description
Script files (-T)	This option passes the -T argument to the linker file
Do not use standard start files (-nostartfiles)	This option passes the -nostartfiles argument to the linker file. It does not allow the use of the standard start files.
Do not use default libraries (-nodefaultlibs)	This option passes the -nodefaultlibs argument to the linker file. It does not allow the use of the default libraries.
No startup or default libs (-nostdlib)	This option passes the -nostdlib argument to the linker file. It does not allow the use of startup or default libs.
Remove unused sections (-Xlinker --gc-sections)	This option passes the -Xlinker --gc-sections argument to the linker file. It removes the unused sections.
Print removed sections (-Xlinker --print-gc-sections)	This option passes the -Xlinker --print-gc-sections argument to the linker file. It ptints the removed sections.
Omit all symbol information (-s)	This option passes the -s argument to the linker file. This option omits all symbol information.

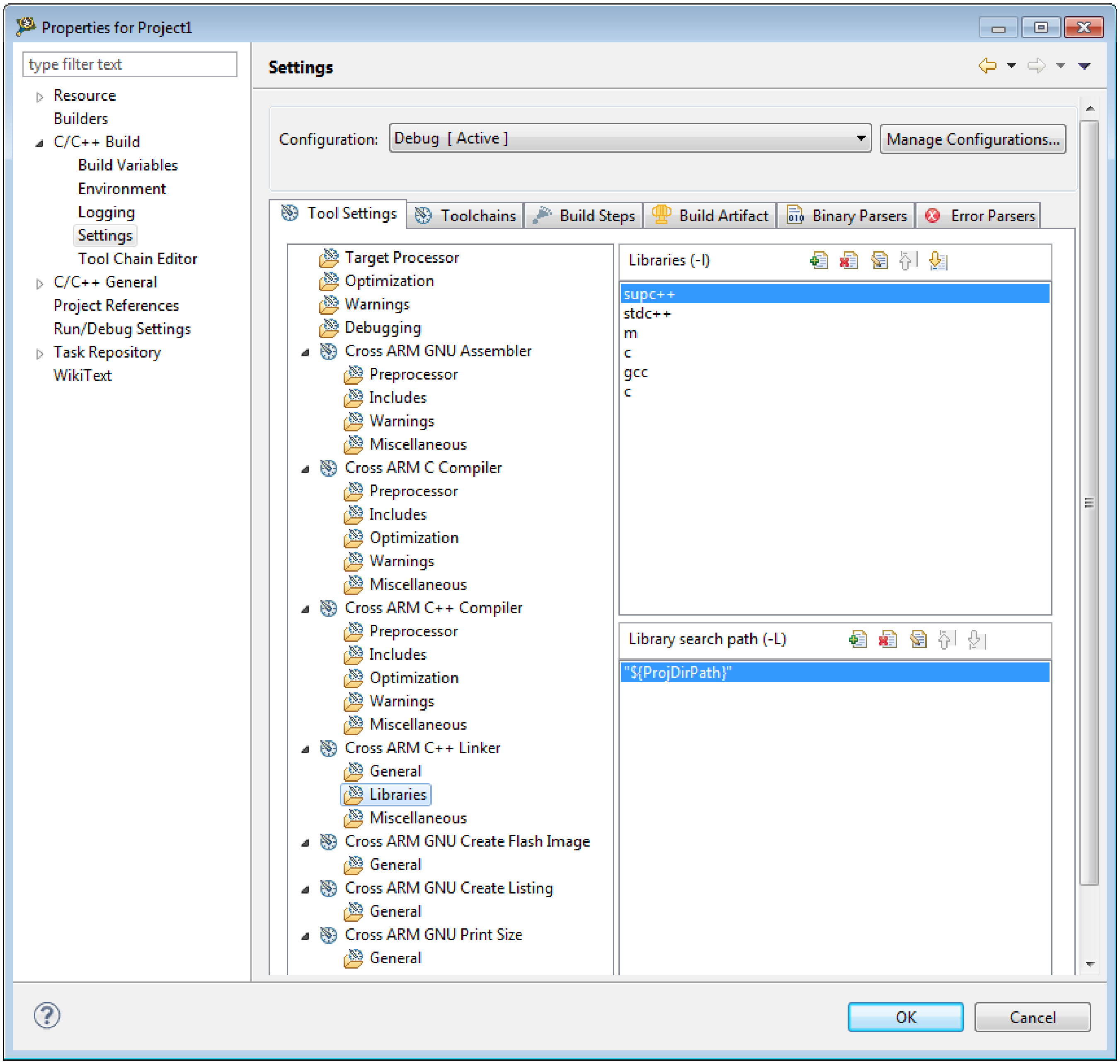
3.2.7.2 Libraries

Use this panel to configure the ARM C linker libraries options.

The following table lists the options in the Libraries panel.

Table 3-19. Libraries options

Option	Description
Libraries (-l)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search #include files in several different ways. You can also set the search order as follows: For include statements of the form #include"xyz", the compiler first searches user paths, then the system paths. For include statements of the form #include<xyz>, the compiler searches only system paths. This option is global.
Library search path (-L)	Use this option to specify the include library search path.



Configuration: Debug [Active]

Libraries (-l)

```
supc++
stdc++
m
c
gcc
c
```

Library search path (-L)

```
"${ProjDirPath}"
```

Figure 3-2. Libraries panel

3.2.7.3 Miscellaneous

Use this panel to configure the ARM C linker miscellaneous options.

The following table lists the options in the **Miscellaneous** panel.

Table 3-20. Miscellaneous options

Option	Description
Linker flags	This option specifies the flags to be passed with the linker file.
Other objects	This option lists paths that the VSPA linker searches for objects. The linker searches the paths in the order shown in this list.
Generate Map	This option specifies the map filename. Default: $ {BuildArtifactFileBaseName}.map
Cross Reference (-Xlinker --cref)	Check this option to instruct the linker to list cross-reference information on symbols. This includes where the symbols were defined and where they were used, both inside and outside macros.
Print link map (-Xlinker --printf-map)	Check this option to instruct the linker to print the map file.
Verbose (-v)	Check this option to show verbose information, including hex dump of program segments in applications; default setting
Other linker flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

3.2.8 Cross ARM GNU Create Flash Image

Use this panel to configure the Cross ARM GNU create flash image options.

The following table lists the options in the **Cross ARM GNU Create Flash Image** panel.

Table 3-21. Cross ARM GNU Create Flash Image options

Option	Description
Command	Shows the location of the executable file. Default: `${cross_prefix}${cross_objcopy}${cross_suffix}`
All Options	Shows the actual command line the assembler will be called with. Default: `"FirstProjectTest.elf" -O ihex`
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

3.2.8.1 General

Use this panel to configure the Cross ARM GNU create flash image general options.

The following table lists the options in the General panel.

Table 3-22. General options

Option	Description
Output file format	Defines the object file format.
Section: -j .text	Select to define section: -j .text.
Section: -j .data	Select to define section: -j .data.
Other sections (-j)	Add other sections.
Other flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.2.9 Cross ARM GNU Create Listing

Use this panel to configure the Cross ARM GNU create listing options.

The following table lists the options in the **Cross ARM GNU Create Listing** panel.

Table 3-23. Cross ARM GNU Create Listing options

Option	Description
Command	Shows the location of the executable file. Default: `${cross_prefix}${cross_objdump}${cross_suffix}}`
All Options	Shows the actual command line the assembler will be called with. Default: `"FirstProjectTest.elf" --source --all-headers --demangle --line-numbers --wide`
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

3.2.9.1 General

Use this panel to configure the Cross ARM GNU create listing general options.

The following table lists the options in the General panel.

Table 3-24. General options

Option	Description
Display source	Check to display source.
Display all headers	Check to display headers in the listing file; disassembler writes listing headers, titles, and subtitles to the listing file
Demangle names	Check to demangle names.

Table continues on the next page...

Table 3-24. General options (continued)

Option	Description
Display debugging info	Check to display debugging information.
Disassemble	Check to disassembles all section content and sends the output to a file. This command is global and case-sensitive.
Display file headers	Check to display the contents of the overall file header.
Display line numbers	Check to display the line numbers.
Display relocation info	Check to displays the relocation entries in the file.
Display symbols	Check to display the symbols.
Wide line	Check to display wide lines.
Other flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

3.2.10 Cross ARM GNU Print Size

Use this panel to configure the Cross ARM GNU print size options.

The following table lists the options in the **Cross ARM GNU Print Size** panel.

Table 3-25. Cross ARM GNU Print Size options

Option	Description
Command	Shows the location of the executable file. Default: `$${cross_prefix}${cross_size}${cross_suffix}`
All Options	Shows the actual command line the assembler will be called with. Default: `--format=berkeley "FirstProjectTest.elf"`
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: `${COMMAND} ${INPUTS} ${FLAGS}}`

3.2.10.1 General

Use this panel to configure the Cross ARM GNU print size options.

The following table lists the options in the General panel.

Table 3-26. General options

Option	Description
Size format	Select size format: Berkeley or SysV
Hex	Select to choose Hex.

Table continues on the next page...

Table 3-26. General options (continued)

Option	Description
Show totals	Select to show totals.
Other flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

Chapter 4
Preparing target

This chapter lists how to prepare for debug various target types:

- Preparing hardware targets
- Preparing simulator target

4.1 Preparing hardware targets

Please refer to the Getting Started Guide for a description on how to prepare the supported hardware targets.

4.2 Preparing simulator target

This topic explains how to configure and start simulator.

To configure and start the simulator, perform these steps:

4.2.1 Configuration

This topic explains how to configure simulator.

1. If you're running the CodeWarrior software on a Linux machine, note that the simulator is already unpacked under Common/CCSSim folder, skip steps 2-4 .

2. If you're running the CodeWarrior software on a Windows machine and you have installed the Simulator package on a remote Linux64 machine during the installation of CodeWarrior software, skip steps 3-4.

3. Get the simulator from the CodeWarrior layout:

```
<Layout>/Common/CCSSim/LS_SIM_RELEASE_0_x_0_00xxx.tgz
```

4. Move the file to the Linux x86_64 machine and untar it.

5. For information about licensing the simulator, see section "Licensing" in the *Layerscape Simulator User Guide*.

4.2.2 Use cases

This section lists the simulator use cases.

- Bare metal debug
- U-boot debug, Linux kernel debug, Linux application debug

4.2.2.1 Bare metal debug

To perform bare metal debug:

1. If you're running the CodeWarrior software on a Windows machine, navigate to the linux64 folder inside the unpacked archive of the simulator you have set up on the Linux x86_64 machine

2. If you're running the CodeWarrior software on a Linux machine, navigate to the Common/CCSSim folder inside the CodeWarrior installation folder.

3. For details about the simulator start-up scripts available for debugging, see section "Layerscape architecture flavors and simulator start-up scripts" in the *Layerscape Simulator User Guide*.

4. For bare metal debug on LS2085A, run the following simulator start-up script:

```
./start_sim_bare_metal
```

4.2.2.2 U-Boot debug, Linux kernel debug, Linux application debug

This topic explains steps to perform U-Boot, Linux kernel, and Linux application debug.

To perform U-Boot, Linux kernel, and Linux application debug:

1. On top of the simulator start-up scripts, there is a package consisting of a set of SDK binary images (U-Boot, Linux kernel) and a start-up script called `run-sim.sh`, which loads all the mentioned images and begins execution on the primary GPP core. For details, see "Using ls2-sim-support scripts (run-sim.sh) and CodeWarrior" section in the *Layerscape Simulator User Guide*.

2. If you have your custom SDK images, copy them in the images folder from `ls2-sim` support package.
3. In a console, navigate to the `ls2-sim-support` folder.
4. Set the `LS2_SIM_BASE_DIR` environment variable to point to the location of the simulator scripts.
5. For U-Boot debug and Linux kernel debug, run:

   ```
   ./run-sim.sh -g
   ```
6. For Linux application debug, run

   ```
   ./run-sim.sh
   ```

   Wait until the Linux kernel is booted and the Linux login prompt appears.
7. If you need complete details about the `run-sim.sh` parameters, run:

   ```
    ./run-sim.sh -h
   ```

Chapter 5
Configuring Target

The Target Connection Configuration (TCC) feature lets you configure the probe and the target hardware.

TCC eases out the configuration process due to the auto- discovery capabilities and live validation of the configuration. TCC lets you use one configuration for multiple projects by setting it as the active configuration (configure once debug all projects), but if more than one configuration is required, you can add as many configuration as necessary. TCC can be used as an RCP application for eclipse allowing the user to benefit from the full capabilities either way.

This chapter lists:

- Target Connection configurator overview
- Configuration types
- Operations with configurations
- Configure the target configuration using Target Connection Configurator
- Generating GDB script from a configuration
- Debugger server connection
- Logging Configuration

5.1 Target Connection configurator overview

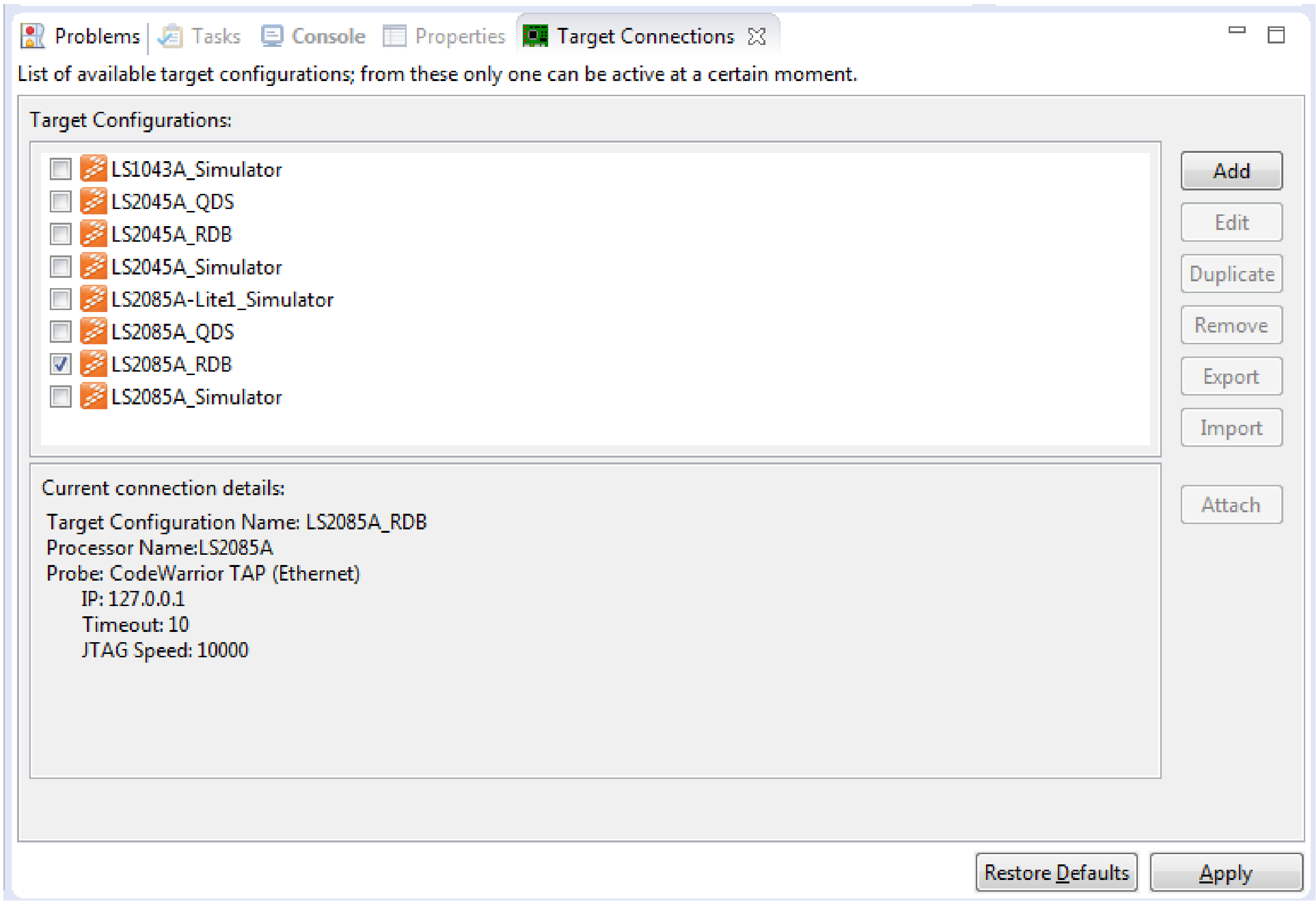
You can view all existing configuration, manage configurations, and set the active configuration using the Target Connection manager.

To access the Target Connection manager (using eclipse), select **Window > Preferences > Target Connection Configuration**. You will be able to see the Target Connection manager in the right panel of the Preferences window.

Besides the possibility to configure the target connection through the preferences, you can access the same capabilities available in the Target Connection View.

To access the Target Connections view, select **Window > Show View > Other > Target Connections**.

The view lists a brief information about the current connection.



List of available target configurations; from these only one can be active at a certain moment.

Target Configurations:

- □ LS1043A_Simulator
- □ LS2045A_QDS
- □ LS2045A_RDB
- □ LS2045A_Simulator
- □ LS2085A-Lite1_Simulator
- □ LS2085A_QDS
- ☑ LS2085A_RDB
- □ LS2085A_Simulator

Current connection details:

```
Target Configuration Name: LS2085A_RDB
Processor Name:LS2085A
Probe: CodeWarrior TAP (Ethernet)
      IP: 127.0.0.1
      Timeout: 10
      JTAG Speed: 10000
```

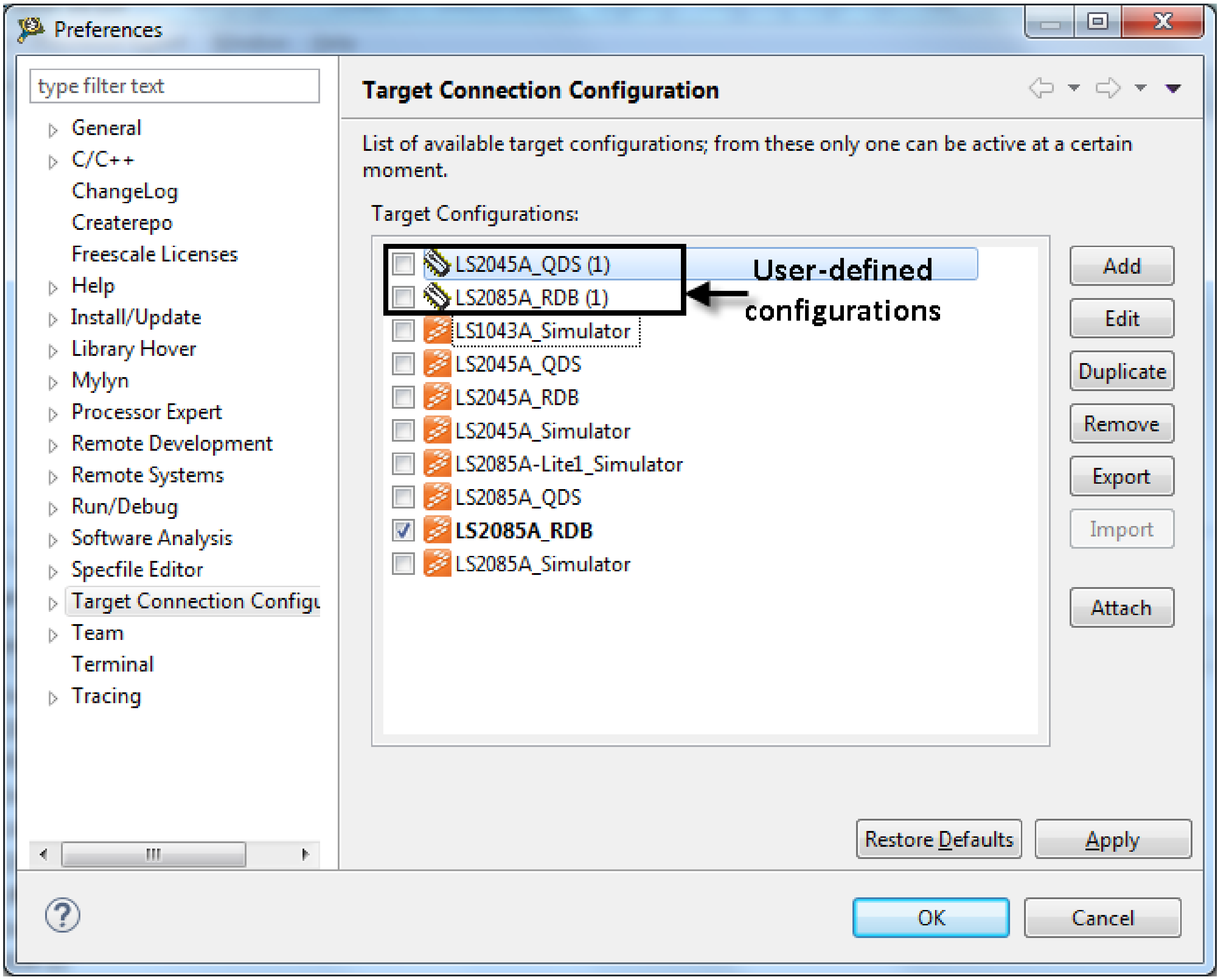
Figure 5-1. Target Connection view

5.2 Configuration types

There are two type of target connection configuration: user-defined and pre-defined.

The pre-defined configurations are marked with orange icons with Freescale logo.Unlike, user-defined configuration, pre-defined configurations cannot be removed. Also, the user doesn’t have access to the pre-defined configuration file; therefore the pre-defined configurations cannot be imported or exported.

However, the pre-defined configuration can be duplicated and saved under a different name.



List of available target configurations; from these only one can be active at a certain moment.

Target Configurations:

- □ LS2045A_QDS (1)
- □ LS2085A_RDB (1)
- □ LS1043A_Simulator
- □ LS2045A_QDS
- □ LS2045A_RDB
- □ LS2045A_Simulator
- □ LS2085A-Lite1_Simulator
- □ LS2085A_QDS
- ☑ **LS2085A_RDB**
- □ LS2085A_Simulator

Figure 5-2. Configuration types

5.3 Operations with configurations

This topic explains the target connection configuration options.

The table below lists the target connection configuration options that you can use to manage configurations, view all existing configuration, and set the active configuration.

Table 5-1. Target Connections Configurations options

Option	Description
Add	Use to create a new configuration.
Edit	Use to edit the selected configuration. Modify the configuration then click **OK** to save the changes.

Table continues on the next page...

Table 5-1. Target Connections Configurations options (continued)

Option	Description
Duplicate	Use to duplicate an existing configuration. You can edit the duplicated configuration.
Remove	Use to remove an existing configuration. Select the configuration you want to delete, and click **Remove**.
Export	Use to export a configuration to the workspace. Select the configuration you want to export. Click **Export**. Select the location in the workspace where you want to export the configuration and click **OK** to finish.
Import	Use to import a configuration from the workspace. Select the configuration you want to import to the internal configuration folder. Click **Import**.
Set Active configuration	Check the checkbox next to the configuration to set it as Active Configuration.
View details about a configuration	TCC panel lets you determine whether a configuration is pre-defined or user-defined by using different color icons; Orange for pre-defined and Green for user-defined. Also, if a configuration is not complete and cannot be used for debug, TCC panel marks it as *(Incomplete)*.

5.4 Configure the target configuration using Target Connection Configurator

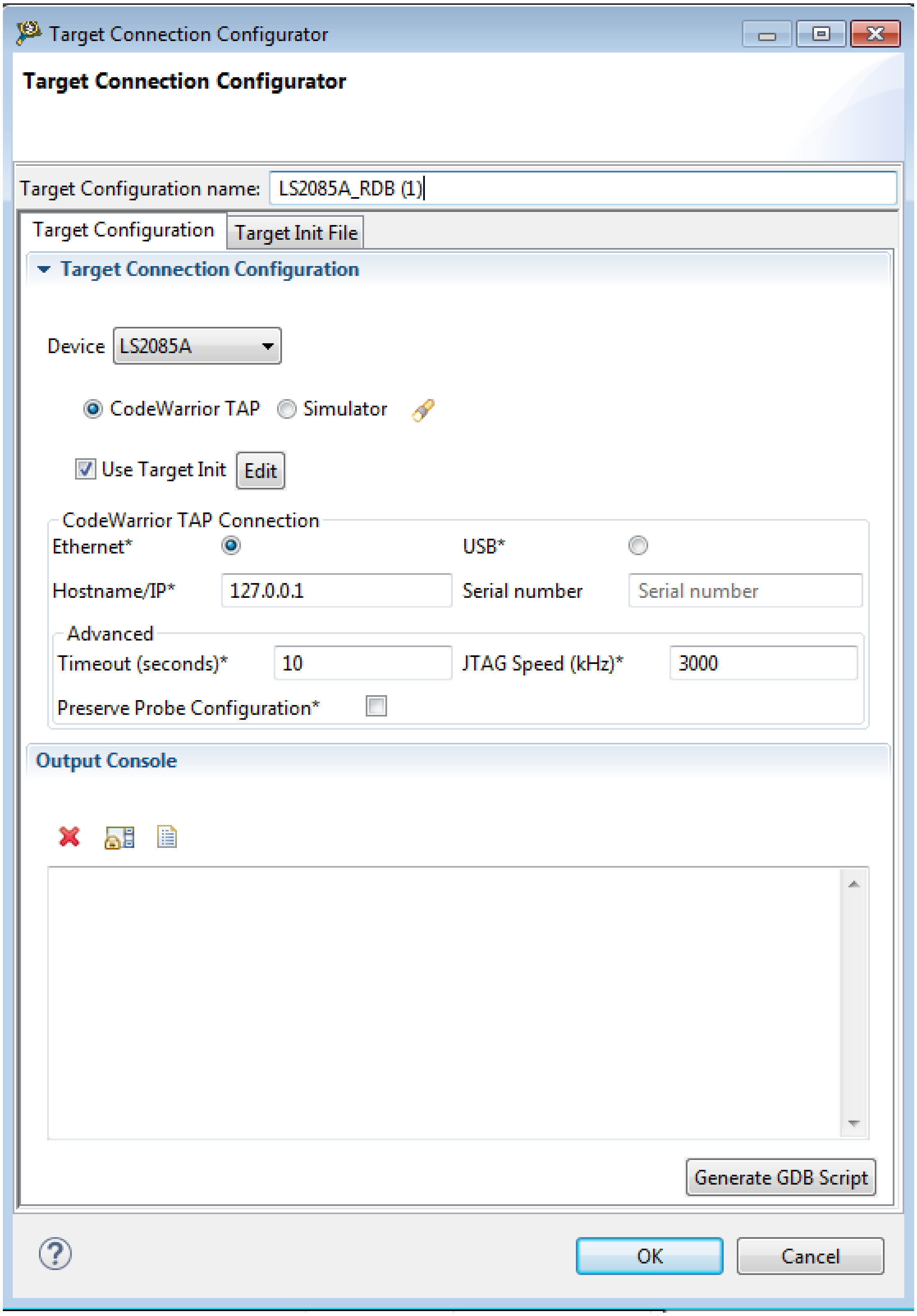
To configure the target configuration in Target Connection Configurator, you need to select the debugged processor and the probe.

1. Choose the debugged core from the launch configuration file.
2. In order to connect to the target, select a connection type, such as simulator or hardware. And configure the probe options, such as IP, serial number for USB connection.
   The available probes depend on the selected processor. For example, since there is CWTAP support for LS2085A-Lite1 and LS2085A, CWTAP probe is supported additional to Simulator. In case you select CWTAP, you could use probe discovery capability.

   NOTE

   in this release, the list of detected CWTAPs also includes the probes connected to other processors in addition to the one selected.

   a. Click the **Search for HW probes** button to automatically discover the probes connected to the local machine or network for SoCs that support CWTAP (for SoCs that support only simulators the button will be disabled).



Target Connection Configurator
Target Connection Configurator

Target Configuration name: LS2085A_RDB (1)

Target Configuration
Target Init File
Target Connection Configuration

Device LS2085A

● CodeWarrior TAP ○ Simulator

☑ Use Target Init

CodeWarrior TAP Connection

Ethernet* ● USB* ○

Hostname/IP* 127.0.0.1 Serial number Serial number

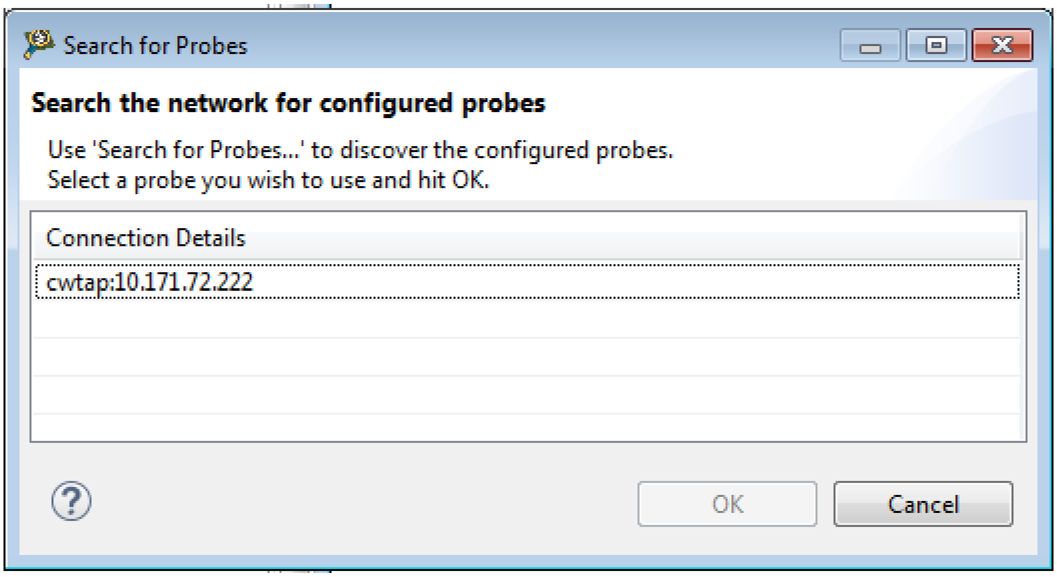
Advanced

Timeout (seconds)* 10 JTAG Speed (kHz)* 3000

Preserve Probe Configuration* □

Output Console
Generate GDB Script
OK
Cancel

Figure 5-3. Target Connection Configurator



Search for Probes

Search the network for configured probes

Use 'Search for Probes...' to discover the configured probes.
Select a probe you wish to use and hit OK.

Connection Details
cwtap:10.171.72.222

Figure 5-4. Search for probes

b. When the user selects one of the discovered probes, the target configuration will use the selected CWTAP and the selected probe attributes will be updated accordingly.

3. Select **Preserve Probe Configuration** to make all CWTAP configurations disappear. In this case, you will have to specify only CCS server used to access the CWTAP.
4. Check the **Use Target Init** checkbox to load/edit the gdb file and launch the target init GDB script.
5. You can load/save and edit the gdb script in the **Target Init File** tab page.

   The script is loaded by GDB and it is run automatically before launching a debug session. Initialization script is embedded in the TCC configuration.

6. Click **Apply** to save and set the new target connection as an active configuration. To set an active configuration, select the check box corresponding to the target connection to be set as active configuration. The active configuration acts as source for the target connection data necessary to start a debug session.

NOTE

In the **Target Connection** view, the active configuration name is displayed in bold.

5.5 Generating GDB script from a configuration

TCC configures the target by sending a set of commands to the GDB server.

These commands can be exported and viewed as a .gdb script. To export the .gdb script:

1. Configure the target configuration using Target Connection Configurator.
2. Click **Generate GDB script**.

   The GDB script with the parameters configured in the target connection dialog are outputted in the **Output Console** view. Export the GDB script to the required file using **Export log into file**.

   GDB script can be used as it is when starting a debug session from the (GDB) console mode.

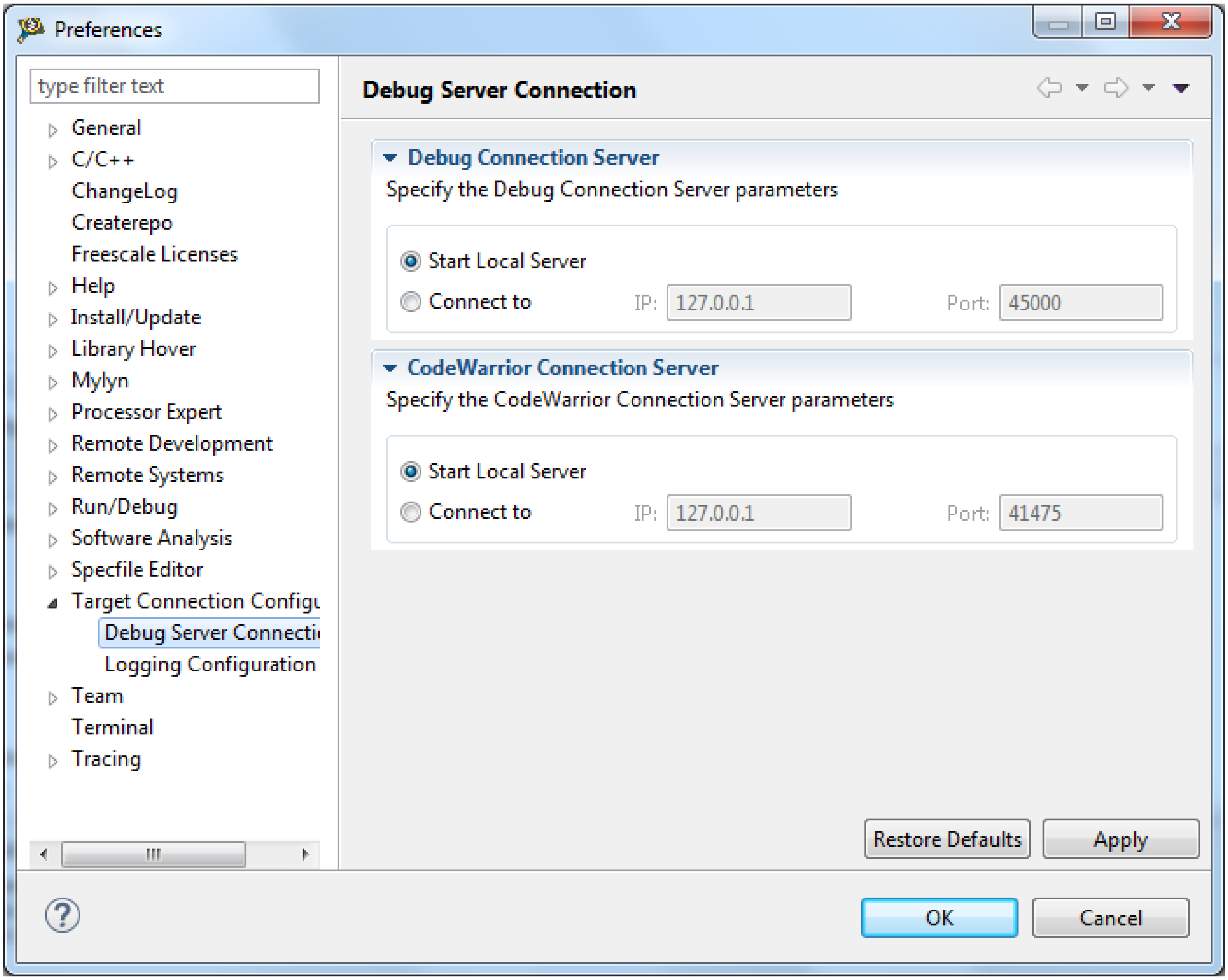
5.6 Debugger server connection

Each target connection configuration allows the user to select the type of connection to use with GTA: a local server or a remote connection to an already set up GTA server.

- **Debug Server Connection**
  - **Start local server**: Automatically starts the GTA first time when a certain command is issued to GTA. The GTA will be stopped when the user chooses to use a remote GTA or the CodeWarrior software is closed.
  - **Connect to**: User can specify the server address and IP of an already running debug server.

    In the **CodeWarrior Connection Server** section the user can specify the connection server parameters.

- **CodeWarrior Connection Server**
  - **Start Local Server**: GTA starts and configures the connection server.
  - **Connect to**: The connection server is already started/configured and the GTA can use it.



Debug Server Connection

Debug Connection Server

Specify the Debug Connection Server parameters

● Start Local Server

○ Connect to IP: 127.0.0.1 Port: 45000

CodeWarrior Connection Server

Specify the CodeWarrior Connection Server parameters

● Start Local Server

○ Connect to IP: 127.0.0.1 Port: 41475

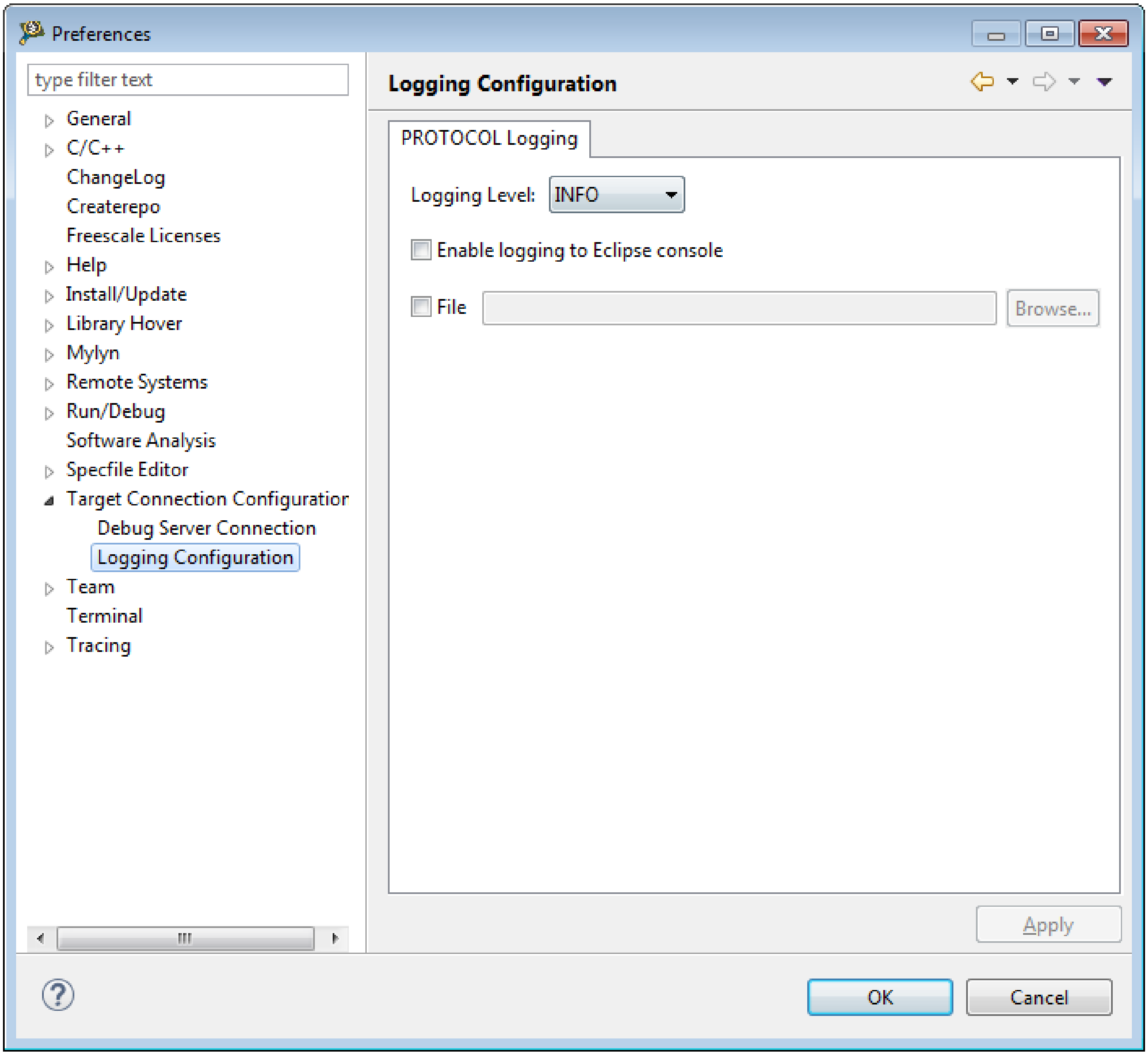
Figure 5-5. Debug Server Connection

5.7 Logging Configuration

The **Logging Configuration** preference panel can be used to enable the protocol logging (ccs).

Using this panel, the user can configure the logging level and choose the destination for the collected info:

- an Eclipse console, PROTOCOL Logging Console. The console is visible only when **Enable logging to Eclipse console** is selected
- a file



Logging Level: INFO

□ Enable logging to Eclipse console

□ File

Figure 5-6. Logging Configuration panel

The INFO level for logging adds more information to the output, for example register IDs, memory addresses, memory spaces. But it does not output the contents for register values, memory, and JTAG chain expansion (for get_config_chain() command).

For details about monitor log commands, refer Logging.

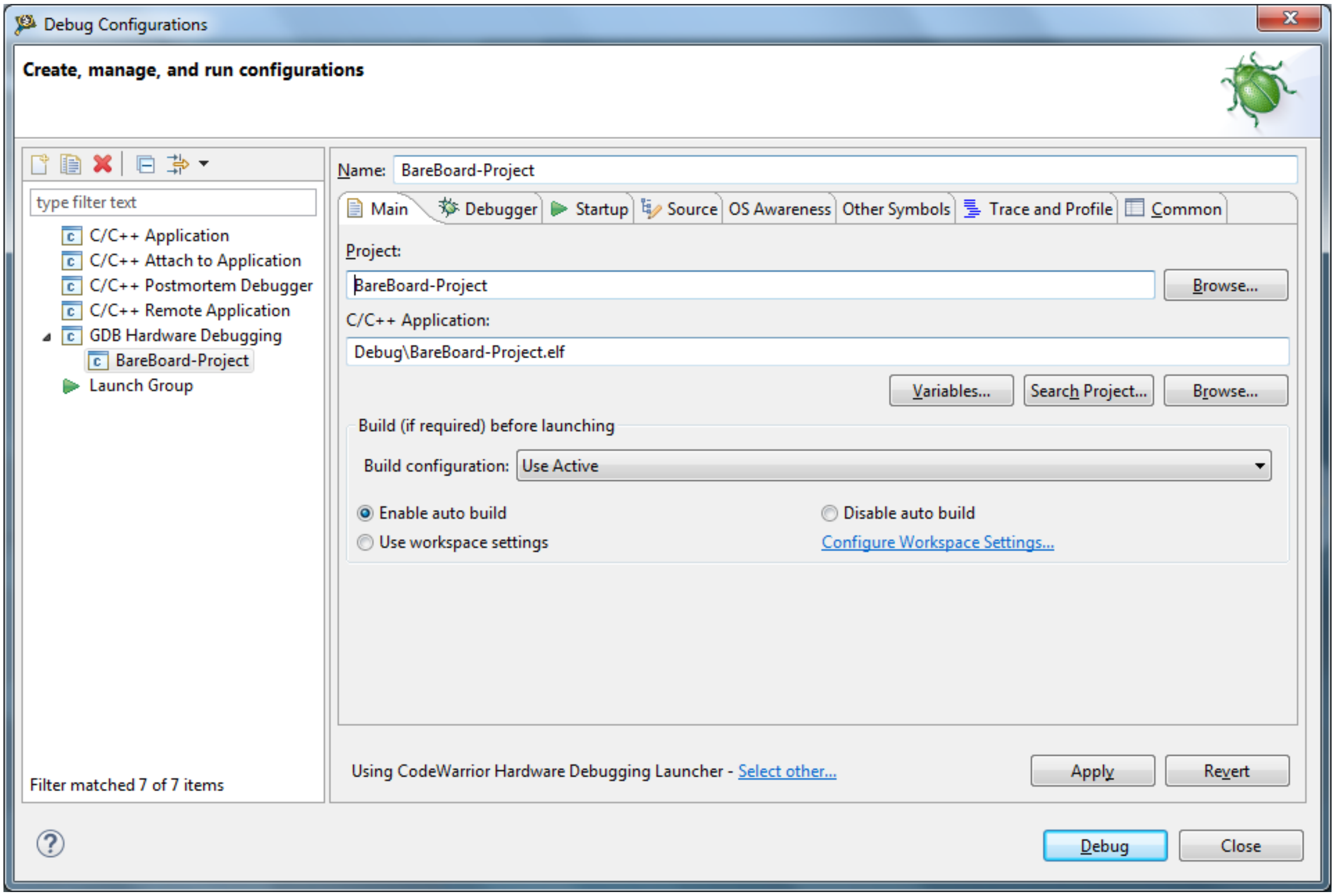
Chapter 6
FSL Debugger References

This chapter lists:

- Customizing debug configuration
- Registers features
- OS awareness
- Linux kernel awareness
- Launch a hardware GDB debug session where no configuration is available
- Memory tools GDB extensions
- Monitor commands

6.1 Customizing debug configuration

You can use the **Debug Configurations** dialog to customize various aspects of a debug configuration.



Name: BareBoard-Project

Project: BareBoard-Project

C/C++ Application: Debug\BareBoard-Project.elf

Build configuration: Use Active

● Enable auto build
○ Disable auto build
○ Use workspace settings

Filter matched 7 of 7 items

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 6-1. Debug Configurations dialog

NOTE

The CodeWarrior debugger shares some pages, such as Connection and Download. The settings that you specify in these pages also apply to the selected debugger.

To modify a debug configuration:

1. Click **Run > Debug Configurations** in the CodeWarrior IDE.

   The **Debug Configurations** dialog appears.
2. Make the required changes, and click the **Apply** button to save the pending changes.
3. To undo the pending changes, click the **Revert** button.

   The IDE restores the last set of saved settings to all pages of the Debug Configurations dialog. The **Revert** button appears disabled until you make new pending changes.

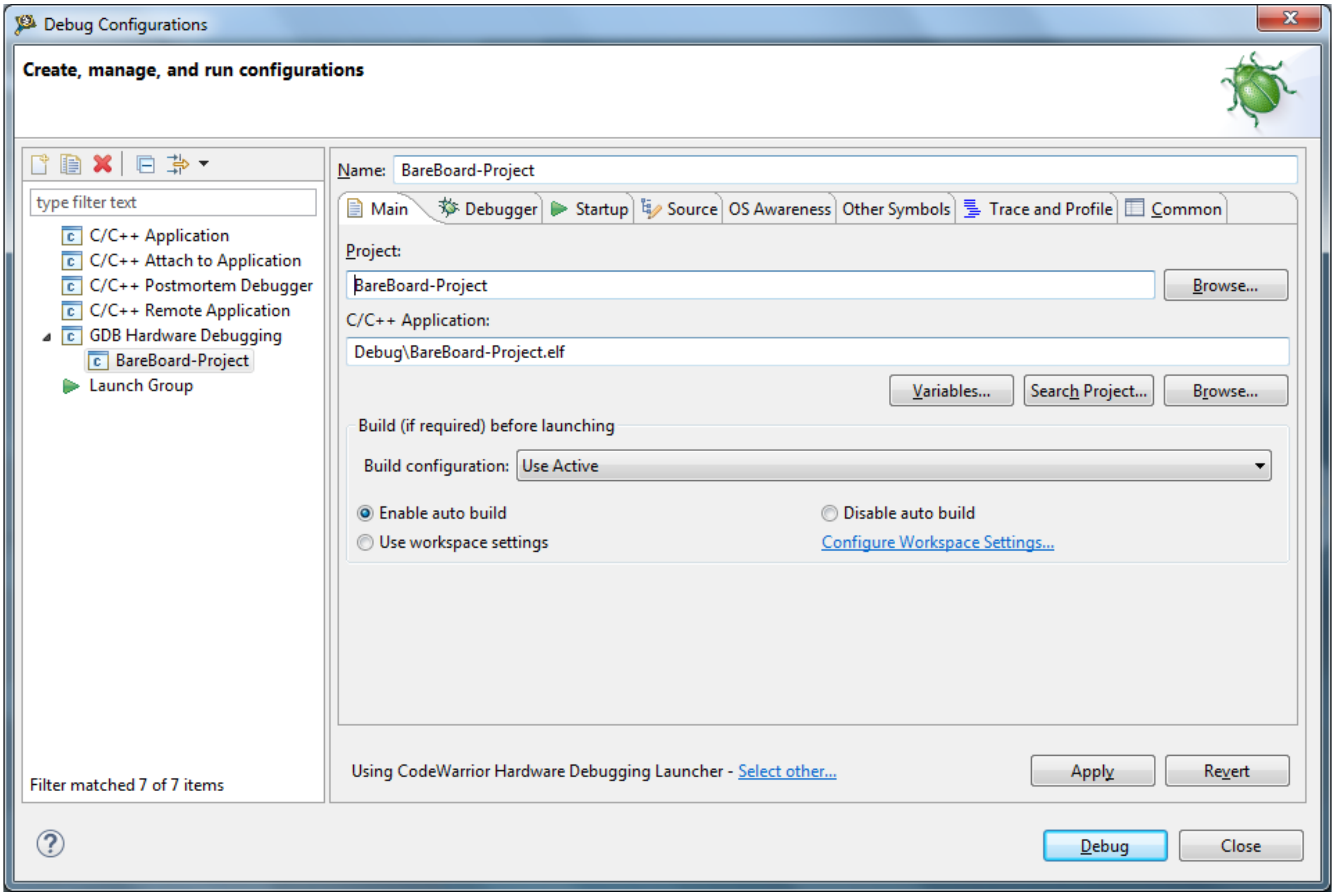
4. A debug configuration can be saved within the project by setting its location relative to the project loaded in the current workspace. For this, click the Common tab, and in the **Shared file** option, select a project directory where you want to save the debug configuration. Now, you can import the project into another workspace without loosing the debug configuration file.
5. Click the **Close** button to close the **Debug Configurations** dialog.

The tabs in the **Debug Configurations** dialog box are:

- Main
- Debugger
- Startup
- Source
- OS Awareness
- Other Symbols
- Common
- Trace and Profile

6.1.1 Main

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



Debug Configurations

Create, manage, and run configurations

Name: BareBoard-Project

Project: BareBoard-Project

C/C++ Application: Debug\BareBoard-Project.elf

Build (if required) before launching

Build configuration: Use Active

● Enable auto build ○ Disable auto build

○ Use workspace settings

Filter matched 7 of 7 items

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 6-2. Main tab

The Main tab options are explained in the following table.

Table 6-1. Main tab options

Option	Description
C/C++ Application	Specifies the name of the C or C++ application.
Variables	Click to open the Select build variable dialog box and select the build variables to be associated with the program. **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.
Search Project	Click to open the **Program Selection** dialog box and select a binary.
Browse	Click **Browse** to select a different C/C++ application.
Project	Specifies the project to associate with the selected debug launch configuration. Click **Browse** to select a different project.
Build (if required) before launching	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.
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.

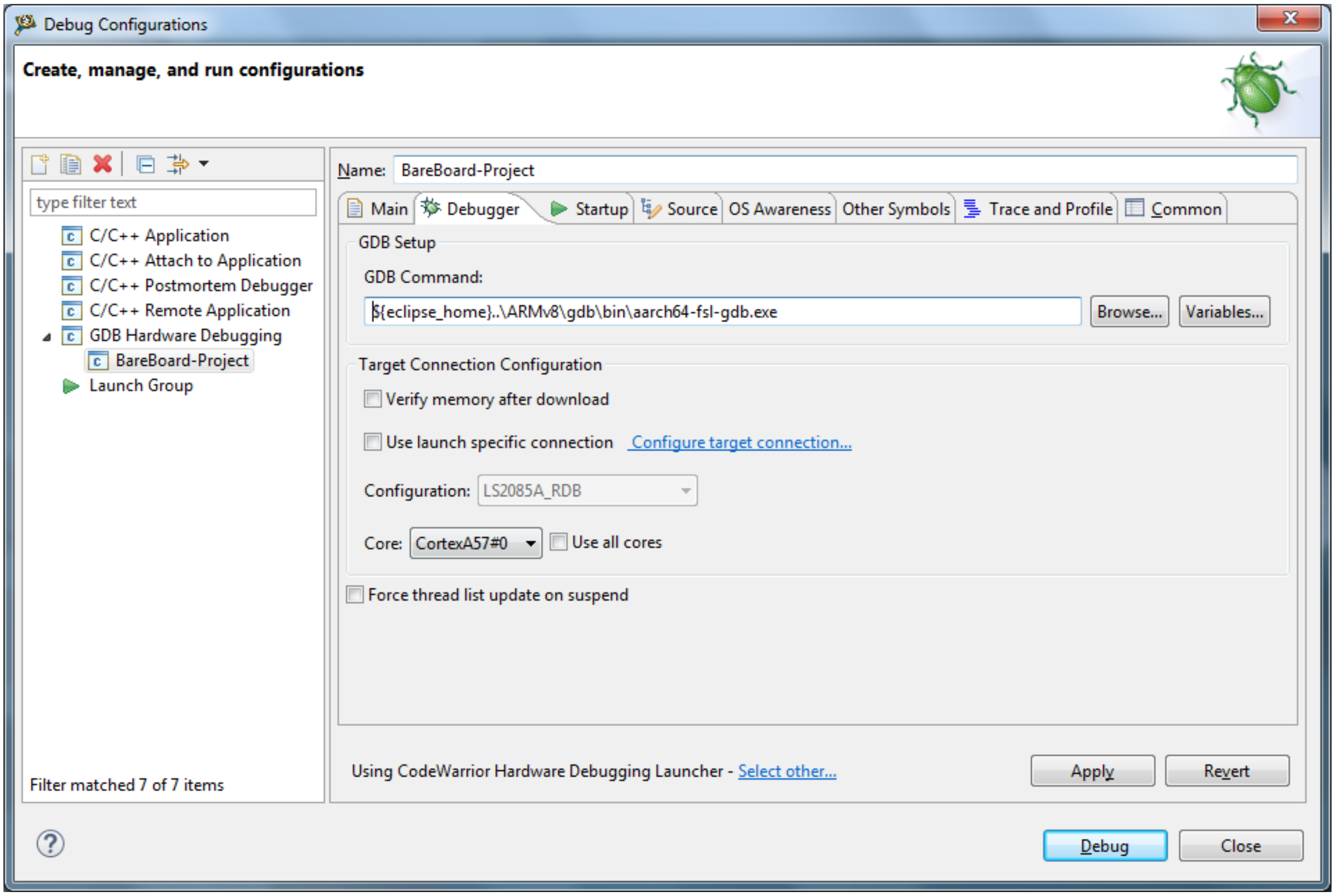
Table continues on the next page...

Table 6-1. Main tab options (continued)

Option	Description
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 Active (default)	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.

6.1.2 Debugger

Use this page to select a debugger to use when debugging an application.



Debug Configurations

Create, manage, and run configurations

Name: BareBoard-Project

GDB Setup

GDB Command: ${eclipse_home}..\ARMv8\gdb\bin\aarch64-fsl-gdb.exe

Target Connection Configuration

□ Verify memory after download

□ Use launch specific connection Configure target connection...

Configuration: LS2085A_RDB

Core: CortexA57#0 □ Use all cores

□ Force thread list update on suspend

Using CodeWarrior Hardware Debugging Launcher - Select other...

Filter matched 7 of 7 items

Figure 6-3. Debugger tab

The Debugger tab options are explained in the following table.

Table 6-2. Debugger tab options

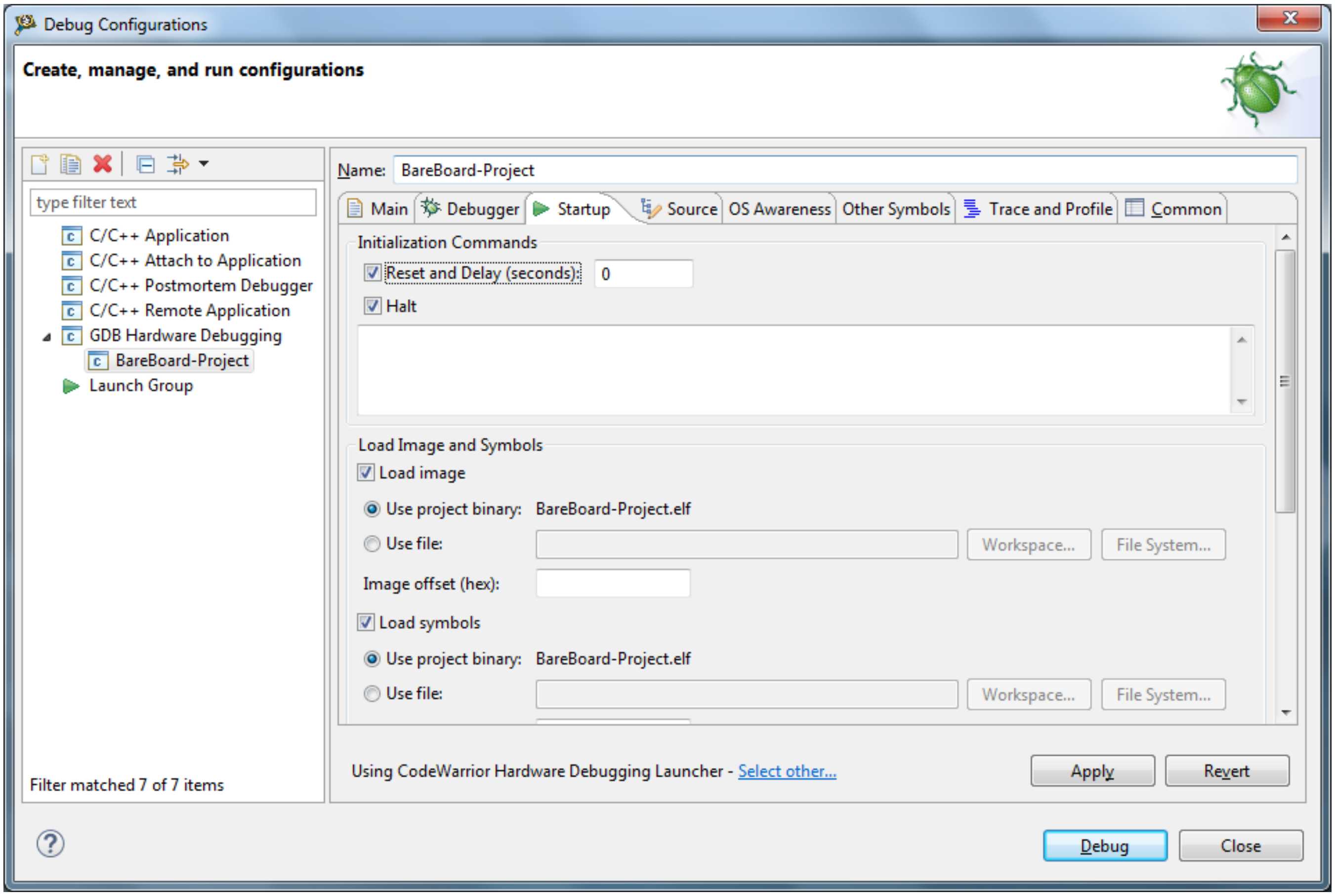
Option	Description
GDB Setup	
GDB Command	Specifies the GDB command. For example: ${eclipse_home}..\ARMv8\gdb\bin\aarch64-fsl-gdb.exe.
Browse	Click to navigate.
Variables	Click to select variables.
Target Connection Configuration	

Verify memory download	If selected, download validation is performed after binary is downloaded to target. The console displays the validation result in an output similar to the one presented below.
	`Section .note.gnu.build-id, range 0x400000 -- 0x400024: matched.`
	`Section .text, range 0x400040 -- 0x400568: matched.`
	`Section .rodata, range 0x400568 -- 0x400578: matched.`
	`Section .data, range 0x410578 -- 0x410cd0: matched.`
	If checkbox is deselected, validation is not performed and the above output is not displayed.
	In GDB command line, a user can execute the `compare-sections` command after the executable is loaded to target (using the `load` command), and same output will be displayed. A typical GDB session with download validation is presented in the example below.
	`target extended-remote host:port`
	`mon ctx set current :ccs:LS2085A:A57#0`
	`attach 1`
	`load elf_file`
	`file elf_file`
	`compare-sections`

Use launch specific connection	Select to specify the target connection configuration in this launch. This will override the configuration specified globally in **Window->Preferences** dialog.
Configuration	Enabled when Use launch specific connection is checked. Use to select the required configuration.
Core	Select the core to debug.
Use all cores	Select if your application uses all cores (SMP).
Force thread list update on suspend	Click if you want to force thread list update on suspend.

6.1.3 Startup

Use this page to specify the startup options and values to use when an application runs.



Name: BareBoard-Project

Initialization Commands

☑ Reset and Delay (seconds): 0

☑ Halt

Load Image and Symbols

☑ Load image

● Use project binary: BareBoard-Project.elf

○ Use file:

Image offset (hex):

☑ Load symbols

● Use project binary: BareBoard-Project.elf

○ Use file:

Using CodeWarrior Hardware Debugging Launcher - Select other...

Filter matched 7 of 7 items

Figure 6-4. Startup tab

The following table list the **Startup** tab options.

Table 6-3. Startup tab options

Option	Description
Reset and Delay (seconds)	Select to reset the target at startup and delay the initialization for the specified amount of seconds
Halt	Select to halt the target at startup
Load image	Select to specify that an image should be loaded to the target
Use project binary	Select to load the binary of the current project
Use file	Select to load a different file
Workspace	Click to select a file to load from the workspace
File System	Click to select a file to load from the file system
Image offset (hex)	Specify the offset on the target from where to load the image
Load symbols	Select to specify that symbols should be loaded in the debugger
Use project binary	Select to load symbols from the binary of the current project
Use file	Select to load symbols from a different file
Workspace	Click to select a file with symbols to load from the workspace
File System	Click to select a file with symbols to load from the file system

Table continues on the next page...

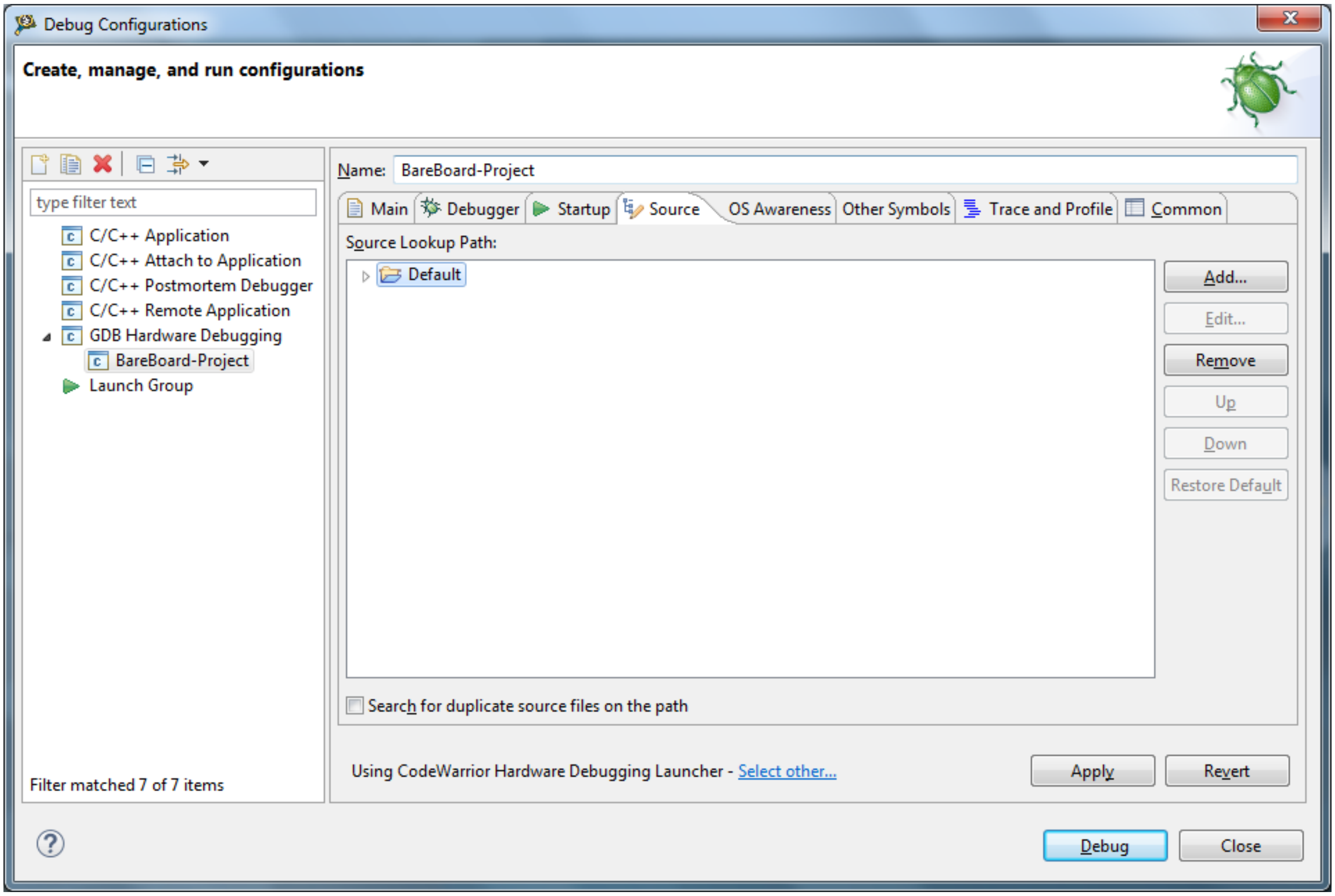
Table 6-3. Startup tab options (continued)

Option	Description
Symbol offset (hex)	Specify an offset for the symbols
Set program counter at (hex)	Select to set the PC at startup to a specified value
Set breakpoint at	Select to set a breakpoint at a specified location
Resume	Select to indicate the execution should resume
Run commands	Specify commands to be run in the debugger after loading image / symbols

6.1.4 Source

Use this page 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.



Debug Configurations

Create, manage, and run configurations

Name: BareBoard-Project

Source Lookup Path:

Default

□ Search for duplicate source files on the path

Using CodeWarrior Hardware Debugging Launcher - Select other...

Filter matched 7 of 7 items

Figure 6-5. Source tab

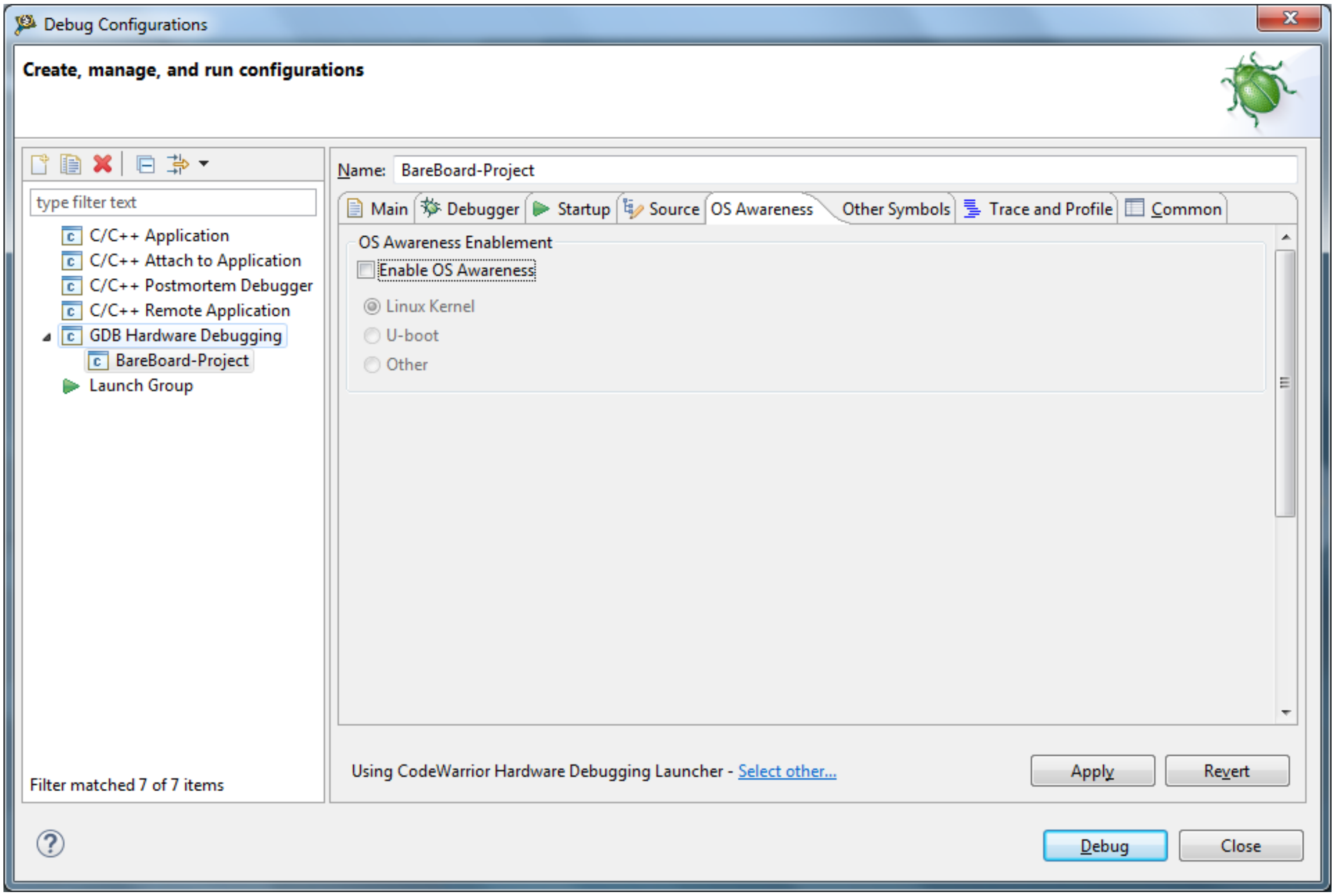
The Source tab options are explained in the following table.

Table 6-4. 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.

6.1.5 OS Awareness

Use this page to specify whether the OS Awareness should be enabled.



Name: BareBoard-Project

☐ Enable OS Awareness

● Linux Kernel

○ U-boot

○ Other

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 6-6. OS Awareness tab

The following table list the **OS Awareness** tab options.

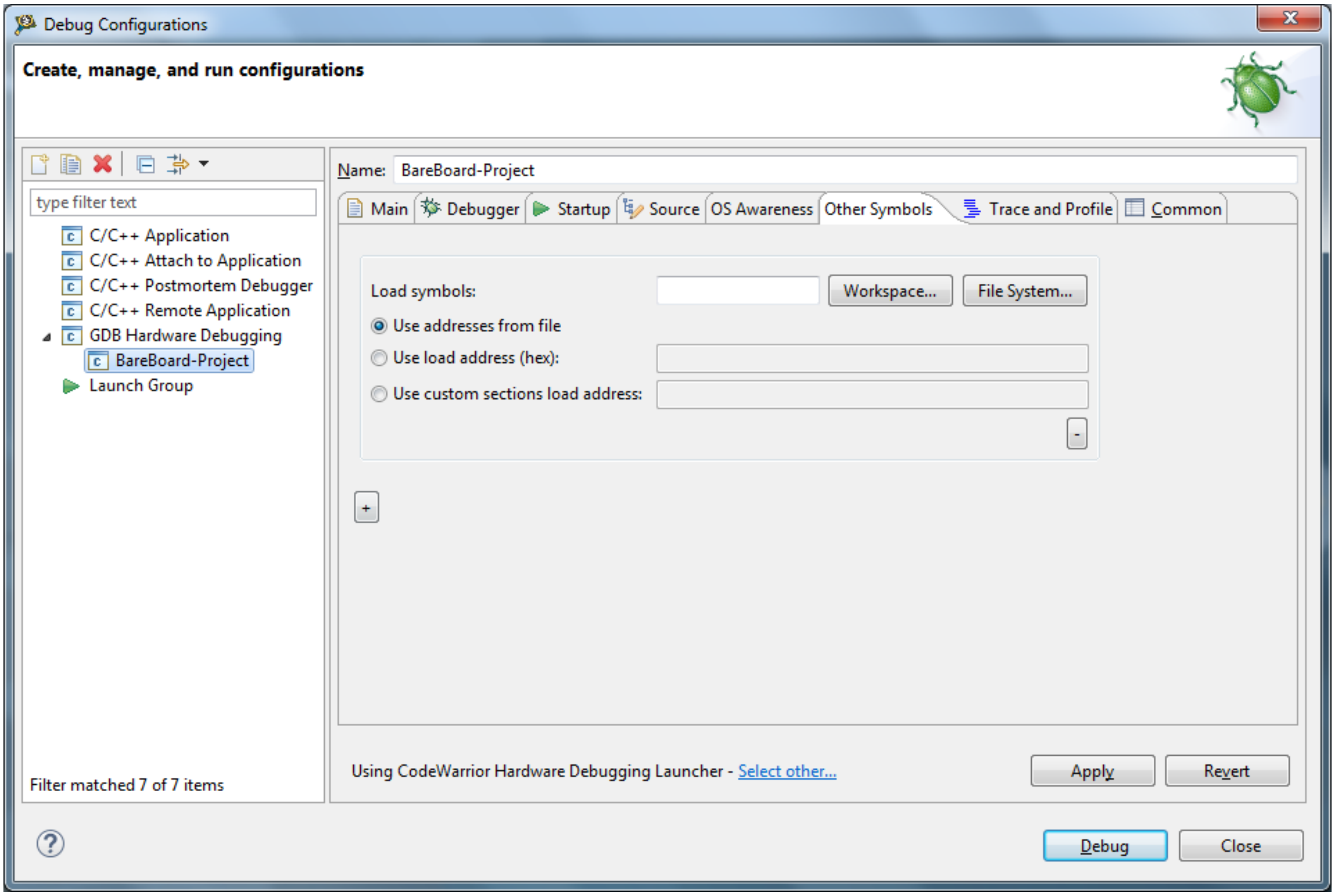
Table 6-5. OS Awareness tab options

Option	Description
Enable OS Awareness	Select to enable OS Awareness (and activate the other tab options).
Linux Kernel	Select to enable OS awareness for Linux Kernel.
U-boot	Select to enable OS awareness for U-Boot.
Other	Select to enable user-defined types of OS awareness.
Use CodeWarrior script for Linux Kernel Awareness	Select to enable OS Awareness for Linux Kernel using CodeWarrior specific script.
Use script	Select to specify a custom script to enable OS Awareness.
Workspace	Click to select a custom script from the workspace.
File System	Click to select a custom script from the file system.
Suspend target when module insert or removal is detected	Select to suspend target when module insert or removal is detected.
Automatically load configured symbolic file at module init detection	Select to automatically load symbolic files.
Auto-load module sybolic files list	Lists automatically loaded symbolic files.

6.1.6 Other Symbols

Use this page to specify other symbols settings.

The **Other Symbols** tab allows reading additional symbol table information from one or more `elf` files given by the user.



Name: BareBoard-Project

Load symbols:

● Use addresses from file

○ Use load address (hex):

○ Use custom sections load address:

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 6-7. Other Symbols tab

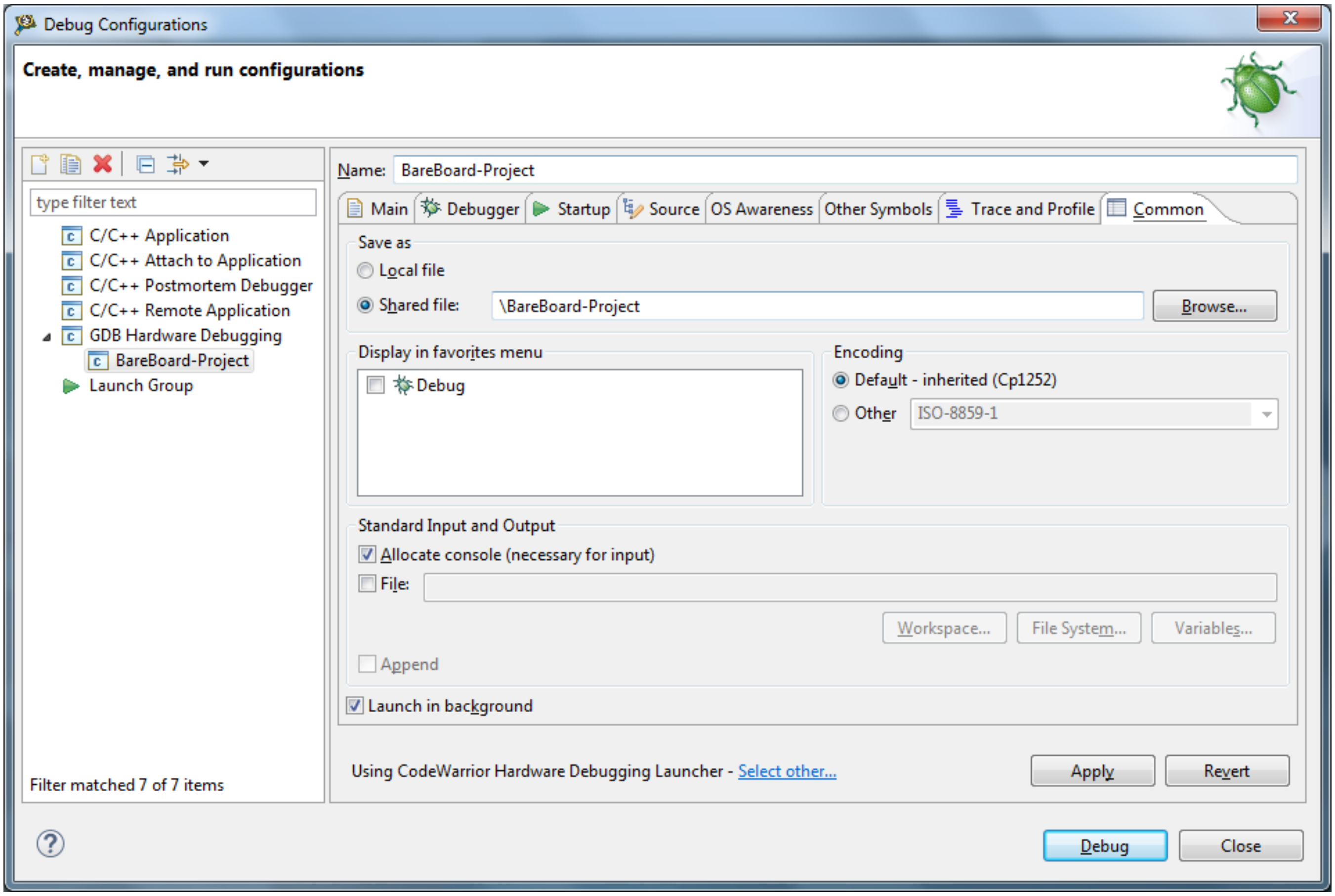
The symbol table information is read by using the `add-symbols` command; this command is similar to the GDB `add-symbol-file` command. However, unlike the `add-symbol-file` command, the `add-symbols` doesn't require the user to provide the load address for the file. The symbols from the `elf` file are loaded using the compile-time addresses for all loadable sections in case this address is not given as a parameter.

If an address is given as a parameter, then the `add-symbols` command loads symbols for all loadable sections based on the specified memory load address. Similar to the GDB command `add-symbol-file`, the `add-symbols` may load symbols for only specific sections at the given load addresses. In order to add symbols from more than one `elf` file, you only need to add a new **Load Symbols** group specifying the new `elf` file and the load options. To remove an `elf` file, press the **Remove** button corresponding to the **Load Symbols** group you want to eliminate.

Option	**Description**
Load Symbols	Choose the `elf` ffile you want to use from either the file system or the workspace.
Use addresses from file	Select to load the symbols from the `elf` file using compile-time addresses for all loadable sections.
Use load address (hex)	Select to load symbols for all loadable sections based on the specified memory load address. The input from the user is a hex address, without the `0x` prefix and it represents the load address in target memory (address of first loadable section).
Use custom sections load address	Select to load symbols for explicitly provided sections at the specified load addresses. Here the user must specify the section and the load address. Example: `-s .text 0x80000000`

6.1.7 Common

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



Name: BareBoard-Project

Save as

○ Local file

● Shared file: \BareBoard-Project

Display in favorites menu

□ Debug

Encoding

● Default - inherited (Cp1252)

○ Other ISO-8859-1

Standard Input and Output

☑ Allocate console (necessary for input)

□ File:

□ Append

☑ Launch in background

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 6-8. Common tab

The following table lists and explains the **Common** tab options.

Table 6-6. Common tab options

Option	Description
Save as	
Local file	Select to save the launch configuration locally.
Shared file	Select to specifies 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	Check to add the configuration name to Run or Debug menus for easy selection.
Encoding	Select an encoding scheme to use for console output.
Standard Input and Output	
Allocate Console (necessary for input)	Select to assign a console view to receive the output.
File	Specify the file name to save output.
Browse Workspace	Specifies the path of, or browse to, a workspace to store the output file.
Browse File System	Specifies the path of, or browse to, a file system directory to store the output file.

Table continues on the next page...

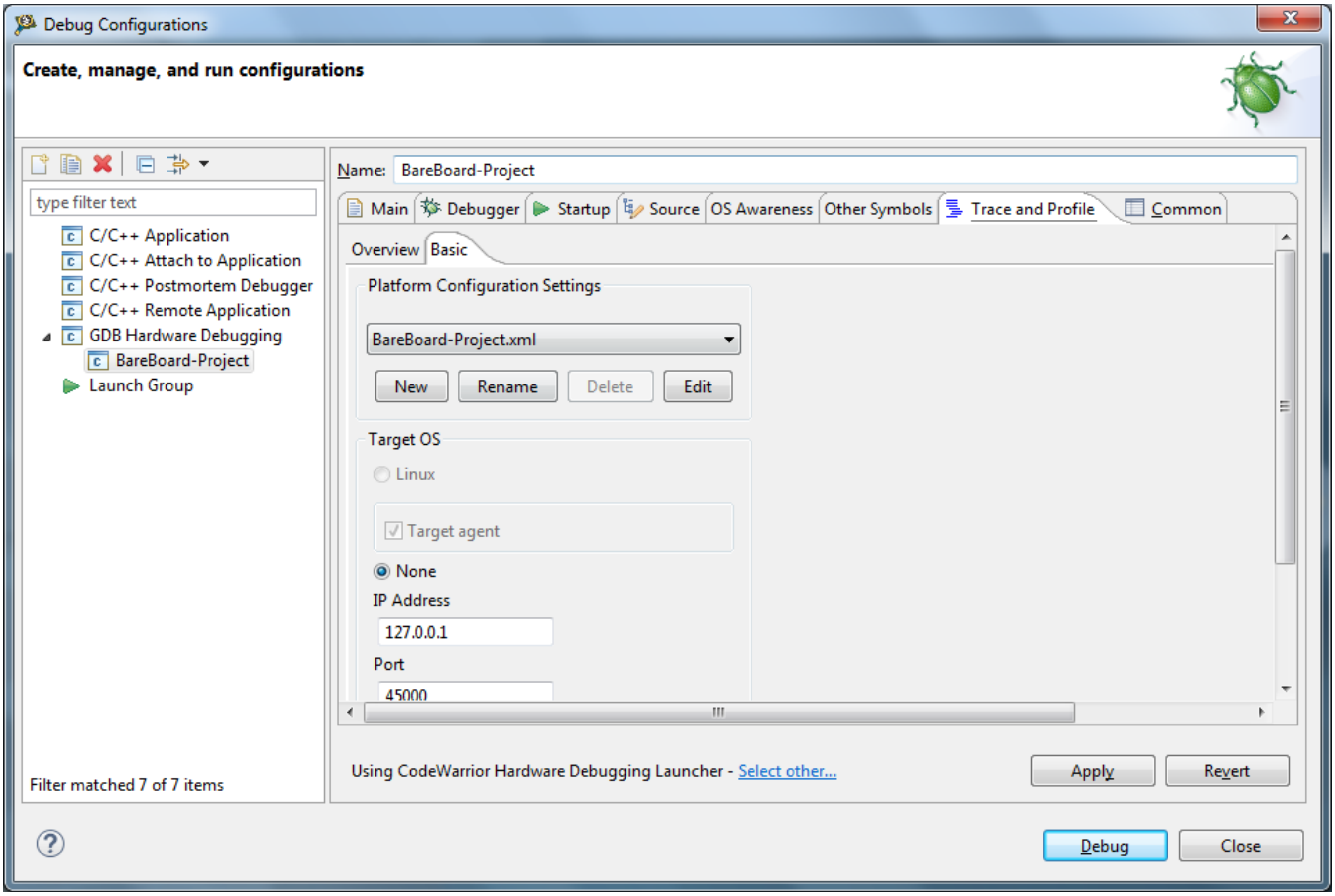
Table 6-6. Common tab options (continued)

Option	Description
Variables	Select variables by name to include in the output file.
Append	Check to append output. Uncheck to recreate file each time.
Port	Check to redirect standard output (`stdout`, `stderr`) of a process being debugged to a user specified socket. **Note:** You can also use the `redirect` 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	Check to launch configuration in background mode.

6.1.8 Trace and Profile

Use this page to specify trace and profile settings.

For any new project, go to **Debug configuration**, select a launch configuration from **GDB Hardware Debugging** from the left panel and select the **Trace and Profile** tab from the right panel.



- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
  - BareBoard-Project
- Launch Group

Filter matched 7 of 7 items

Name: BareBoard-Project

Platform Configuration Settings

BareBoard-Project.xml

Target OS

○ Linux

☑ Target agent

● None

IP Address

127.0.0.1

Port

45000

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 6-9. Trace and Profile tab

6.2 Registers features

This topic explains Peripherals view and GDB customer register commands.

This section lists:

- Peripherals view
- GDB custom register commands

6.2.1 Peripherals view

The Peripherals view lists information about the processor system and platform ip-blocks organized in the form of register groups and memory mapped register groups.

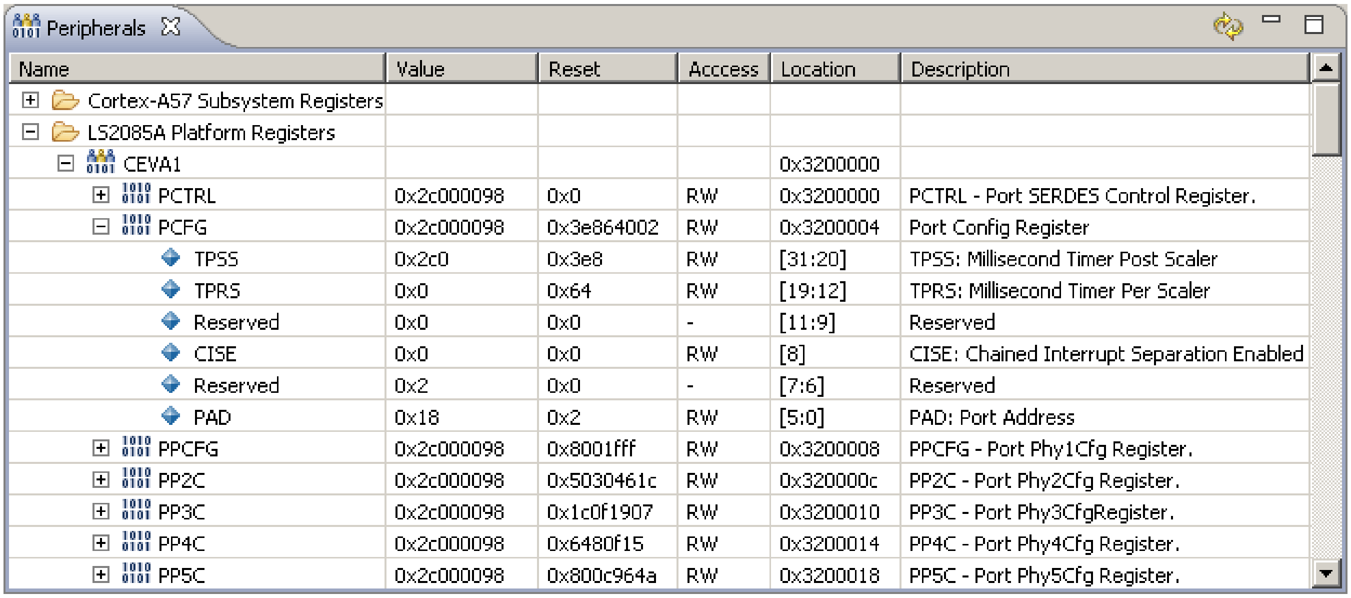
The registers are displayed in a tree view with three columns:

- Name – the name of the register or group
- Value – the value from of the register read from target

- Location – the address of the register or the address of the first register for groups (applicable only for platform groups - MMR).
- Access – the access mode: R=read-only, RW=read-write, W=write-only
- Reset – the register reset value
- Description – the register description

Register and bit field values can be modified on target by clicking in the value cell and entering a new value.

The view is automatically opened when a debug session is started and it is populated with registers when the target is first suspended at program entry point. The view can also be opened manually from the menu: **Window > ShowView> Other > Peripherals** or by using the shortcut: **Alt+Shift+Q, R**.



Name	Value	Reset	Acccess	Location	Description
⊞ Cortex-A57 Subsystem Registers					
⊟ LS2085A Platform Registers					
⊟ CEVA1				0x3200000	
⊞ PCTRL	0x2c000098	0x0	RW	0x3200000	PCTRL - Port SERDES Control Register.
⊟ PCFG	0x2c000098	0x3e864002	RW	0x3200004	Port Config Register
TPSS	0x2c0	0x3e8	RW	[31:20]	TPSS: Millisecond Timer Post Scaler
TPRS	0x0	0x64	RW	[19:12]	TPRS: Millisecond Timer Per Scaler
Reserved	0x0	0x0	-	[11:9]	Reserved
CISE	0x0	0x0	RW	[8]	CISE: Chained Interrupt Separation Enabled
Reserved	0x2	0x0	-	[7:6]	Reserved
PAD	0x18	0x2	RW	[5:0]	PAD: Port Address
⊞ PPCFG	0x2c000098	0x8001fff	RW	0x3200008	PPCFG - Port Phy1Cfg Register.
⊞ PP2C	0x2c000098	0x5030461c	RW	0x320000c	PP2C - Port Phy2Cfg Register.
⊞ PP3C	0x2c000098	0x1c0f1907	RW	0x3200010	PP3C - Port Phy3CfgRegister.
⊞ PP4C	0x2c000098	0x6480f15	RW	0x3200014	PP4C - Port Phy4Cfg Register.
⊞ PP5C	0x2c000098	0x800c964a	RW	0x3200018	PP5C - Port Phy5Cfg Register.

6.2.2 GDB custom register commands

There are several GDB commands for manipulating system and platform registers.

The following commands are currently implemented:

- info reg-groups
- readreg
- writereg

To see a detailed description of each command enter "help <cmd>" in the GDB console.

The commands are querying into an SQLite DB associated with the target that is currently debugged in order to fetch register information based on its name.

Usage examples:

info reg-groups [GROUP_NAME]

readreg GROUP_NAME.REG_NAME

writereg GROUP_NAME.REG_NAME

6.3 OS awareness

OS awareness support in the CodeWarrior software is a group of features that simplify and improve the user experience while debugging the OS-specific projects.

The OS awareness features are enabled from the **OS Awareness** tab in the **Debug Configurations** dialog.

Currently, predefined support exists for debugging Linux kernel and U-Boot projects. When importing an ELF image for a Linux kernel or U-Boot project using the **CodeWarrior Debug Projects** wizard, the image type is auto-detected based on the symbol table and the configuration of the options in the **OS Awareness** tab. The user can manually change the options in the **OS Awareness** tab at any time. Advanced users can also use custom scripts to add the OS awareness support for their specific projects.

6.3.1 Linux kernel awareness

This topic explains how to enable Linux kernel awareness.

To enable Linux kernel awareness, select the checkboxes **Enable OS Awareness**, **Linux Kernel** , and **Use CodeWarrior script for Linux Kernel Awareness** in the **OS Awareness** tab.

For details on how to create a Linux kernel project and start a debug session, see Linux kernel debug.

6.3.1.1 List Linux kernel information

Linux kernel awareness allow users to see relevant Linux kernel operating system information.

- Build time and kernel version

- Kernel module list
- Kernel thread list

The Linux kernel information is available in the command line and in the Eclipse view.

6.3.1.1.1 GDB commands

Once a debug session is started and debug is suspended, go to the gdb console and run `<gdb_command>`. The following GDB commands are available:

- `- ka-show-info`: Prints Linux kernel general information

  ```
  gdb) ka-show-info
  Build Time = #7 Mon Mar 31 11:44:09 EEST 2014
  Linux Version = 3.12.0+
  ```
- `- ka-show-thread-list`: Prints the kernel threads

  ```
  (gdb) ka-show-thread-list
  ```

Name	Pid	State	Address	Core
Swapper	0	running	0xffffffc0004de430	0
init	1	interruptible	0xffffffc079c50000	0
kthreadd	2	interruptible	0xffffffc079c50880	0

6.3.1.1.2 Eclipse view

When Linux kernel awareness is enabled from the OS Awareness tab, the OS Resources view displays information about:

- Linux system information
- modules list
- kernel thread information

6.3.1.2 Linux kernel debug

Linux kernel module debugging is enabled by default when kernel awareness extensions are enabled.

The following gdb commands are implemented for Linux kernel module debug.

6.3.1.2.1 GDB commands

GDB commands:

- - ka-show-module-list :

  Description: Prints Linux kernel modules

  (gdb) ka-show-module-list

Name	Address
krng	0xffffffbffc010000
rng	0xffffffbffc00c000

- ka-module-load:

  Description: Loads symbolics file for a kernel module.

  The commands has the following parameters:
  - (required) the kernel module symbolics file
  - (optional) the module name, necessary when the symbolics file name and the kernel module name are different

  Example:

```
(gdb) ka-module-load /data/ARM_DEVEL/linux/ls2-linux/crypto/krng.o
              Symbol file /data/ARM_DEVEL/linux/ls2-linux/crypto/krng.o loaded
sucessfully
```

- ka-module-unload:

  Description: Unloads symbolics file for a kernel module.

  The commands has one required parameter: the module name

  Example:

```
(gdb) ka-module-unload rng
       Symbol file /data/ARM_DEVEL/linux/ls2-linux/crypto/rng.o unloaded sucessfully
```

- ka-module-files:

  Description:Shows the loaded symbolics file for a kernel modules.

  The command has an optional argument (integer > 0) representing the maximum number of files

  Example:

```
(gdb) ka-module-files
     Name           Loaded file
     rng            /data/ARM_DEVEL/linux/ls2-linux/crypto/rng.o
     krng           /data/ARM_DEVEL/linux/ls2-linux/crypto/krng.o
```

- ka-module-config-suspend:

  Description: Configures module detect suspend action:

  The command has one optional argument (boolean):
  - True: suspend target when module insert/removal is detected
  - False: do not suspend target when module insert/removal is detected

  If no parameter is passed, the command returns the configuration value

  Example:

  ```
  (gdb) ka-module-config-suspend True
  (gdb) ka-module-config-suspend True
  ```

- ka-module-config-auto-load:

  Description: Configures module detect auto-load action:

  The command has one optional argument (boolean):
  - True: automatically load configured symbolic files at module init detection
  - False: no not automatically load module symbolics at module init detection

  If no parameter is passed, the command returns the configuration value.

  Example:

  ```
  (gdb) ka-module-config-auto-load True
  (gdb) ka-module-config-auto-load True
  ```

- ka-module-config-map-load:

  Description: Adds the module symbolics file in the module configuration map.

  If the auto-load is enabled, this symbolics file is automatically loaded when the corresponding module is inserted.

  The commands has the following parameters:
  - (required) the kernel module symbolics file
  - (optional) the module name, necessary when the symbolics file name and the kernel module name are different

  Example:

  ```
  (gdb) ka-module-config-map-load /data/linux/crypto/krng.o
  ```

- ka-module-config-map-unload:

  Unloads symbolics file from the module configuration map. The commands has one required parameter: - the module name

  Example:

```
(gdb) ka-module-config-map-unload krng
```

- ka-module-config-show:

  Description:Shows the module configuration parameters. The command has an optional argument (integer > 0) representing the maximum number of files shown from the configuration map

  Example:

  ```
  (gdb) ka-module-config-show
  Name                    Loaded file
  rng                     /data/linux/crypto/rng.o
  krng                    /data/linux/crypto/krng.o
  ```

6.3.1.3 Linux kernel image version verification

When Kernel awareness is enabled, CodeWarrior performs a Linux Kernel image version verification to validate that the binary image on the target (uImage) matches the ELF symbolic file (vmlinux) in the debugger.

When access to target version is available (after the u-boot copies the Linux kernel image into DDRAM), the debugger performs the version verification. In case of mismatch, the debugger prints the following message in the gdb console: `Warning: Kernel image running on the target is different than the vmlinux image in debugger.`

In addition, the user can trigger at any time the version verification in the following way:

- From CLI using the `ka-show-info` commands. For example:

  ```
  (gdb) ka-show-info

  Build Time = #2 SMP PREEMPT Thu Nov 13 10:09:26 EET 2014
  Linux Version = 3.16.0-Layerscape2-SDK+gec37efe
  Target version check : ELF image version matches target image version
  ```

  In case of version mismatch, the Target version check message is `Warning: Kernel image running on the target is different than the vmlinux image in debugger`. If the access to target version is not available yet, the Target version check message is `not available yet`. The user should check again after the u-boot copies the Linux kernel image into the DDRAM.

- From Eclipse, OS Resources window, select **System Info**. The same information as for the CLI command is shown here.

6.3.2 U-Boot awareness

This topic explains how to enable U-Boot awareness.

The U-Boot awareness features enhance and improve the usability for U-Boot debugging by simplifying the debugging interface. The U-Boot awareness feature provides:

- a single debug session for all U-Boot booting phases that allows user to debug from the first instruction after reset to relocation in DDRAM
- U-Boot command line prompt for booting the Linux kernel

With U-Boot awareness, the debugger automatically detects each U-Boot stage using debugger eventpoints and performs specific actions, such as setting the relocation offset for DDRAM.

To enable the U-Boot awareness features, select the checkboxes **Enable OS Awareness**, **U-boot**, and **Use CodeWarrior script for U-boot Awareness** in the **OS Awareness** tab. For details on how to create a U-Boot project and how to start a debug session, see U-boot Debug.

6.3.2.1 List U-Boot information

When U-Boot awareness is enabled from the **OS Awareness** tab, the **OS Resources** view displays information about:

- U-Boot version, configuration, and build time
- Memory, that is RAM size, RAM top, relocation address, and relocation offset. However, this information is displayed only when the data is available after relocation.

6.3.2.2 U-Boot image version verification

For U-Boot, CodeWarrior performs the same kind of checking as for Linux kernel image. In the same way, the mismatch warning is shown in the gdb console when the U-Boot version is available and the user can check the version at any time from Eclipse, OS Resource window, selecting “Version”.

6.4 Launch a hardware GDB debug session where no configuration is available

This topic explains how to launch a hardware GDB debug session.

Before you proceed, ensure that you have an ARMv8 project in your workspace, which is compiled, and the binary elf file is available.

To launch the debug session, you need to:

1. Create a debug configuration
2. Configure the target configuration using Target Connection Configurator

6.4.1 Create a debug configuration

To create a debug configuration:

1. Select the ARMv8 project in the **Project Explorer** view.
2. Select **Debug > Debug Configurations**. The **Debug Configurations** dialog appears.
3. Right-click **GDB Hardware Debugging** and select **New**.
4. Select the **Main** tab.
5. Make sure that the text box under the **C/C++ Application** option specifies the elf file path of the project you want to use. For example, `Debug/<project name>.elf`
6. Select the **Debugger** tab.
7. In the text box under the **GDB Command** option set the path to gdb. For example, `${eclipse_home}..\ARMv8\gdb\bin\aarch64-fsl-gdb.exe`
8. Click the **Debug** button.

NOTE

For details about configuring target connection, refer Configure the target configuration using Target Connection Configurator

6.5 Memory tools GDB extensions

This topic explains memory tools GDB extensions.

In GDB console the following commands are available:

Table 6-7. Memory tools GDB extensions

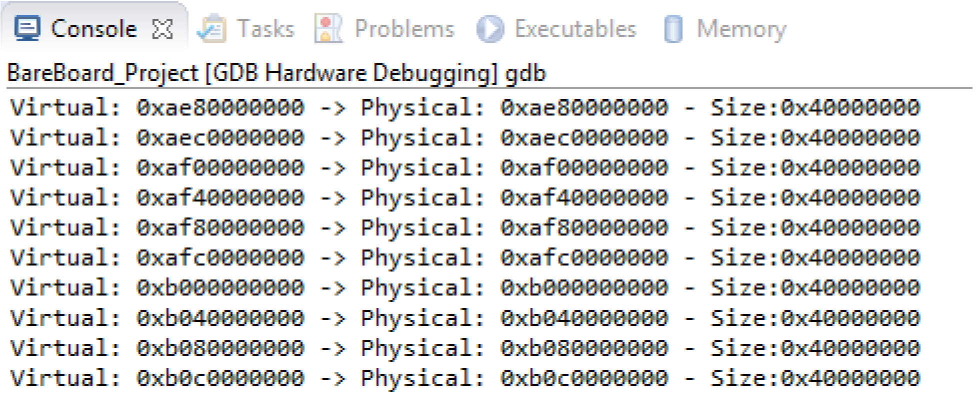
Command	Syntax	Description
Memory fill	lld-mbf start_addr end_addr value	Fill the memory from start_addr to end_addr with data. Example: lld-mbf 0x0 0x40 0xEE will fill 0x0..0x3f with bytes of value 0xEE

Table continues on the next page...

Table 6-7. Memory tools GDB extensions (continued)

Command	Syntax	Description
Memory compare	lld-mbc addr1 addr2 addr3	The memory in the range addr1-addr2 is compared word by word to memory starting from addr3.
Memory modify	lld-mbm addr value	Modify memory at specified address with the specified value. Example: lld-mbm 0x10 0xff - Will change the word at 0x10 to 0x000000ff
Memory Management Unit view	mmu [-h] [-el {0,1,2,3}] [-t <virtual address>]	Dump platform MMU state in a user readable format. **Optional arguments:** • -h, help: Shows this help message and exit. • -el {0,1,2,3}: Specifies the exception level for which to read the translations; if not specified, current exception level will be used. • -t <virtual address>, --translate <virtual address>: Specifies the virtual address to be translated. **NOTE:** In order to see an output, you must be either in debug with a bare board project or attached to a Linux session. **Example:** • Issuing `mmu` command without any parameters will list all the MMU valid entries for the current exception level. • Issuing `mmu -el 3` command will list all the valid MMU entries for the EL3 exception level. • Issuing `mmu -t 0x2000000` will translate the virtual address 0x2000000 to the corresponding physical address using MMU state for the current exception level.

The figure below shows the output of the `mmu` command.



BareBoard_Project [GDB Hardware Debugging] gdb

```
Virtual: 0xae80000000 -> Physical: 0xae80000000 - Size:0x40000000
Virtual: 0xaec0000000 -> Physical: 0xaec0000000 - Size:0x40000000
Virtual: 0xaf00000000 -> Physical: 0xaf00000000 - Size:0x40000000
Virtual: 0xaf40000000 -> Physical: 0xaf40000000 - Size:0x40000000
Virtual: 0xaf80000000 -> Physical: 0xaf80000000 - Size:0x40000000
Virtual: 0xafc0000000 -> Physical: 0xafc0000000 - Size:0x40000000
Virtual: 0xb000000000 -> Physical: 0xb000000000 - Size:0x40000000
Virtual: 0xb040000000 -> Physical: 0xb040000000 - Size:0x40000000
Virtual: 0xb080000000 -> Physical: 0xb080000000 - Size:0x40000000
Virtual: 0xb0c0000000 -> Physical: 0xb0c0000000 - Size:0x40000000
```

Figure 6-10. MMU view

For details about other GDB debug commands that can be run in GDB console from console view, refer the GDB documentation available at: https://sourceware.org/gdb/current/onlinedocs/gdb/

NOTE

Note that Freescale Semiconductor, Inc does not own GDB documentation, and is mentioned solely for reference purpose.

6.6 Monitor commands

This topic explains monitor commands.

The following table lists the available monitor commands.

Command	Syntax	Description
Display contexts tree	mon ctx id <ctx-id> list	Displays the debug contexts tree having as root the specified context. The context has the format: <connection>:<soc>: <core#no>
Set current context	mon ctx set current ctx_id	Set the context for the debug session. This should be set after target extended-remote and before attach. For a single core application the context should look like: <connection>:<soc>:<core#no>. For a multicore application (SMP) the context should be: <connection>:<soc>
	mon ctx get current	Show the current context
	mon ctx id <ctx-id> info	List all properties of the specified context
	mon ctx id <ctx-id> set <prop-name> <value>	Set a property for the specified context
Memory access	monitor mem read [context] address access_size space count	Read memory from address using the provided access size, memory space (see list-ms sub-command) and count. If present, the context can be a core context; otherwise the current context is used. The result is displayed as a hexadecimal encoded byte stream. Example: monitor mem read :ccs:LS2085A:A57#0 0x89ab1234 4 virtual 1
Memory access	monitor mem write [context] address access_size space data	Write memory to address using the provided access size and memory space (virtual, physical -see list-ms sub-command). If present, the context can be a core context; otherwise the current context is used. The data is presented as a series of hexadecimal byte values. Example: monitor mem write :ccs:LS2085A:A57#0 0x89ab1234 1 virtual 1234
Memory spaces	monitor mem list-ms (context)	Lists the available memory spaces. If present, the context can be core contex; otherwise the current context is used.
Reset	monitor reset debug	Performs reset and keeps cores in debug mode.

Table continues on the next page...

Autodiscovery - probes [1]	monitor discover probes [utap|etap|gtap|cwtap]	Discover reachable probes of requested type or all if the type is missing
Autodiscovery - JTAG IDCODEs [1]	monitor discover idcodes <probe>	Discover the JTAG IDCODEs of the devices connected to the specified probe. For example, Eg: monitor discover idcodes cwtap:fsl022dab 0x0A01E01D
Autodiscovery - SoCs[1]	monitor discover socs <probe>	Discover the possible SoCs connected to the specified probe. For example, monitor discover socs cwtap:fsl022dab LS2085A-Lite1

[1] Auto-discovery commands are not supported while using simulators.

6.7 I/O support

Librarian I/O model is divided into 2 modes.

Librarian I/O model is divided into 2 modes:

- UART_C_Static_Lib_Bare: `printf` support through UART port.
- simrdimon: I/O operations through debugger console.

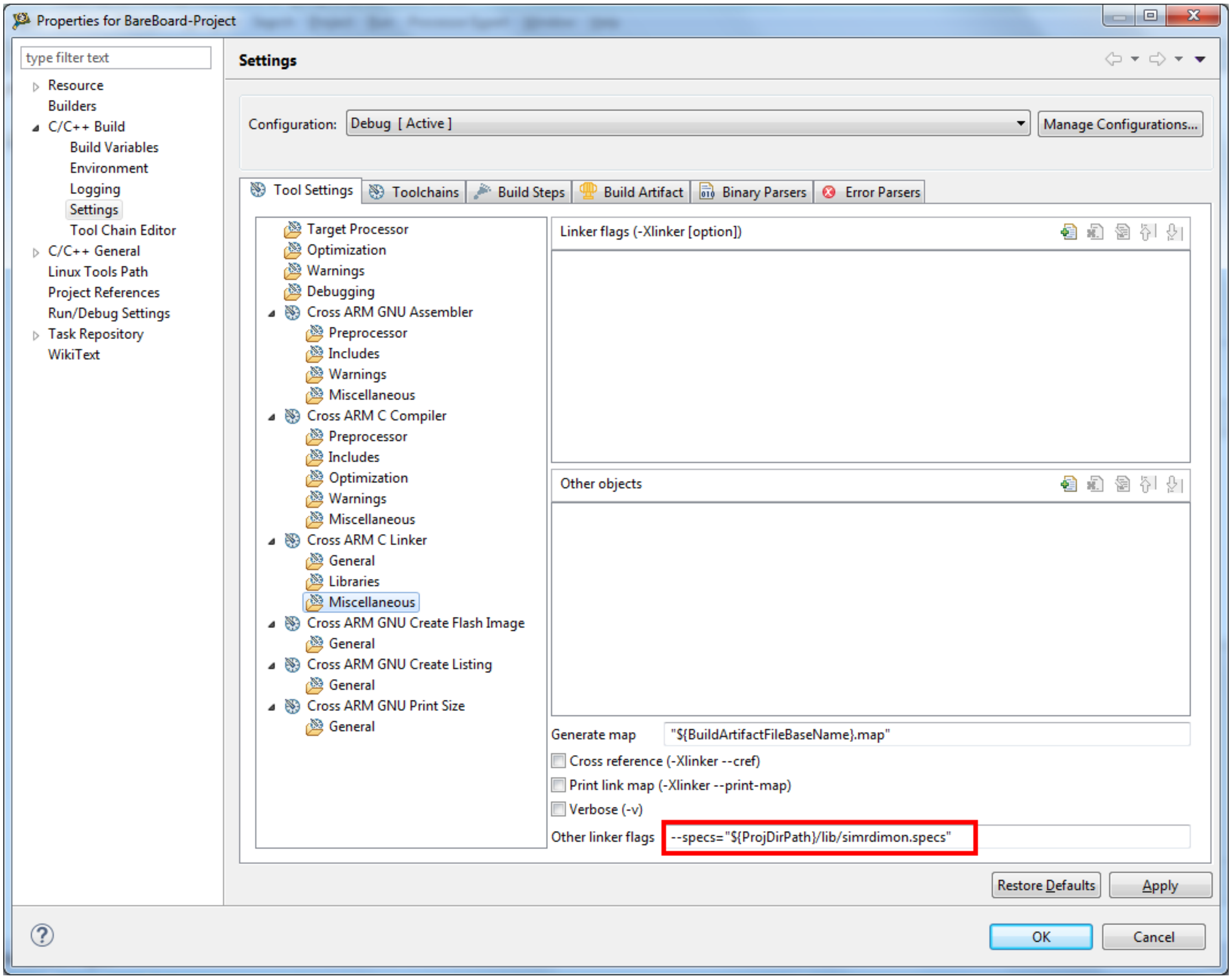
NOTE

> These libraries are compiled by using the highest optimize level for speed (-O3) and no debug data (no DWARF information). The user can recompile these libraries to change the compiler options and use the new libraries in their projects. Projects for these library are located at *{CW_ARMv8}\ARMv8\CodeWarrior_Examples*

There are two examples in ARMv8 stationary wizard:

- C (HelloWorld_C_Base)
- C++ (HelloWorld_CPP_Bare)

The default I/O mode is debugger console; in other words the simrdimon library is used. The user can verify the status by looking at the **Other linker flag** text box, which contains `--specs="${ProjDirPath}/lib/simrdimon.specs"`. Navigate to **Cross ARM C** (or C++) **Linker** > **Miscellaneous** from the left pane under **Tool Settings** tab, to see **Other linker flag** text box.



Configuration: Debug [Active]

Generate map "${BuildArtifactFileBaseName}.map"

□ Cross reference (-Xlinker --cref)

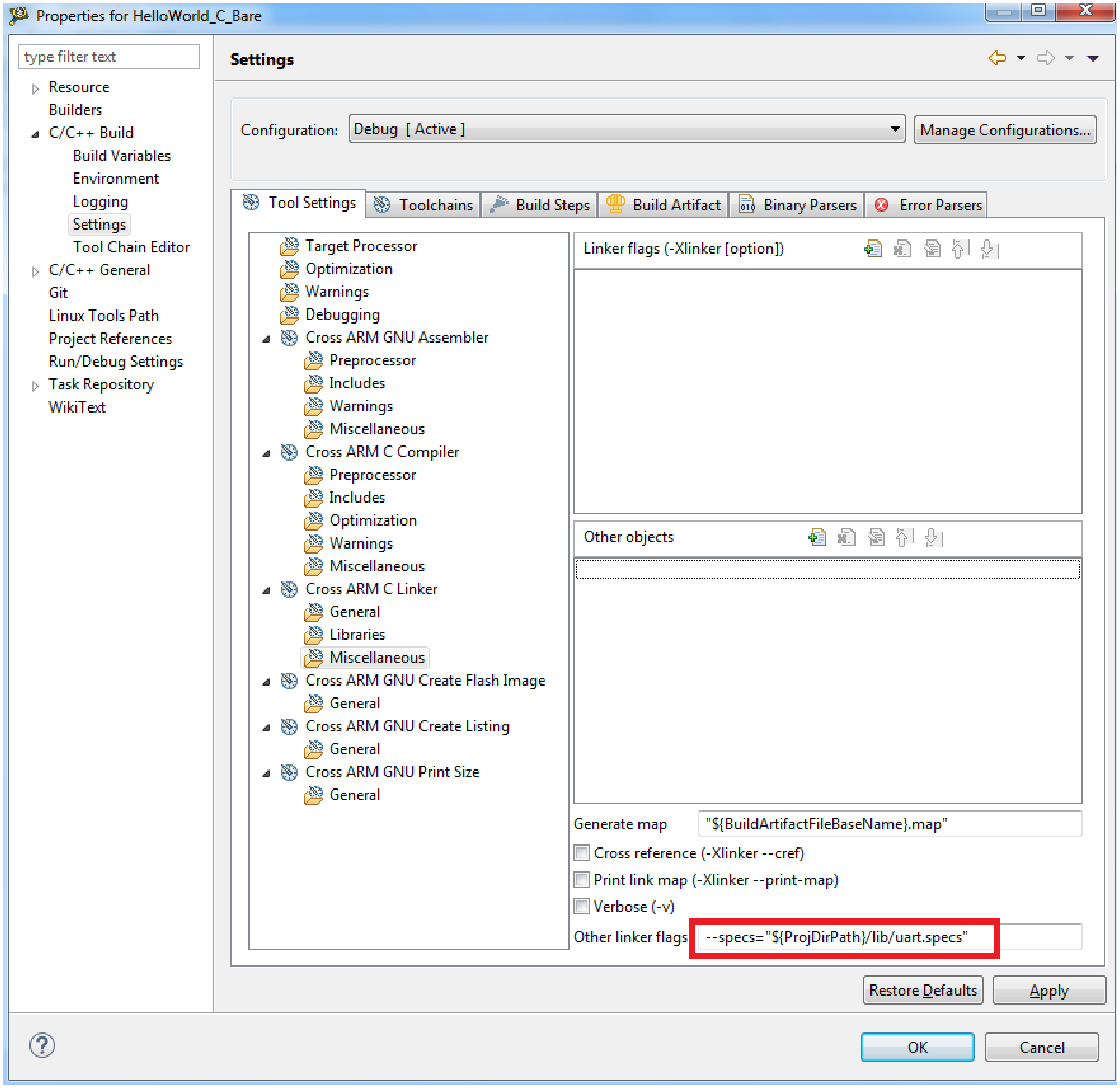
□ Print link map (-Xlinker --print-map)

□ Verbose (-v)

Other linker flags --specs="${ProjDirPath}/lib/simrdimon.specs"

Figure 6-11. Properties dialog - simrdimon.specs

The user can switch to the I/O UART model by changing the file spec for UART model. The user should replace the *simrdimon.specs* with *uart.specs* in the **Other linker flags** text box from **Cross ARM C (or C++) Linker-- > Miscellaneous**.



Configuration: Debug [Active]

Generate map "${BuildArtifactFileBaseName}.map"

□ Cross reference (-Xlinker --cref)

□ Print link map (-Xlinker --print-map)

□ Verbose (-v)

Other linker flags --specs="${ProjDirPath}/lib/uart.specs"

Figure 6-12. Properties dialog - uart.specs

6.7.1 Configuring the UART library and simulator

The simulator is using tio_console for UART redirection. Please refer to the Simulator documentation for further information.

By default, just the start_sim_ uboot script displays a separate tio_console window. In case of using the start_sim_bare_metal simulator, you can specify -use_tio_console argument on the command line to enable the tio_console.

The UART library needs to match the simulator configuration regarding the actual duart port in use, and the backward compatibility support:

- By default the simulator is using “duart1_1” configuration, which means the second port of the duart1 controller (configured in scripts/console.py). The UART library project is configured accordingly to include duartB.c file in the compilation. In case the simulator configuration is changed, the library needs to be recompiled to include the appropriate file in the compilation process (A = duart1_0, B = duart1_1, C = duart2_0, D = duart2_1).
- The simulator uses a SIM_BACKWARD_COMPATIBILITY parameter to force conformance to different architecture specs versions. The DUART register offsets have been changed between the two specs, so the UART library needs to match the simulator. In the duart_config.h file, BACKWARD_COMPATIBILITY needs to be defined to match SIM_BACKWARD_COMPATIBILITY parameter in the simulator. By default the start_sim_uboot simulator used SIM_BACKWARD_COMPATIBILITY=1, while start_sim_bare_metal simulator uses SIM_BACKWARD_COMPATIBILITY=0.

If changes are needed in the UART library project, the recompiled library needs to be added to the wizard-generated projects (HelloWorld_C_Bare or HelloWorld_CPP_Bare).

Chapter 7
Flash programmer

Flash programming is done using python script.

```
{CW Install Dir}\CW_ARMv8\ARMv8\gdb_extensions\flash\cwflash.py
```

7.1 Configuring flash programmer

To configure the flash programmer, open the `cwflash.py` script in an editor and modify the connection parameters in accordance to your setup.

- BOARD_TYPE – supported options are “QDS” and “RDB” for the corresponding board types.
- FLASH_TYPE – supported options are “nor” and “nand”. Please take into account that some device types may not be supported for the selected board.

  First two options should be sufficient for most of the use cases (CodeWarrior TAP connected through USB to GDB host machine). However, if additional configuration is required, please update the next parameters too.

- CWTAP_CONN - If empty, it assumes that CWTAP uses an USB connection. For Ethernet connection please set the IP address. For example: CWTAP_CONN = "192.168.0.1".
- SOC_NAME – name of the SoC. For example: LS2085A.
- JTAG_SPEED – JTAG frequency used by debugger to communicate with the target.
- CCS_PORT, CCS_IP – IP and port of the CCS instance. If a local connection is used (CCS_IP = "127.0.0.1"), debugger will automatically start a CCS instance if none is available on that port.
- GTA_IP, GTA_PORT – IP and port of the GTA (GDB server). This option should be modified if you want to have multiple sessions at the same time.
- GDB_TIMEOUT - Number of seconds to wait for the remote target responses.

7.2 Starting flash programmer

This topic explains steps to start the flash programmer.

To start the flash programmer, perform the following steps:

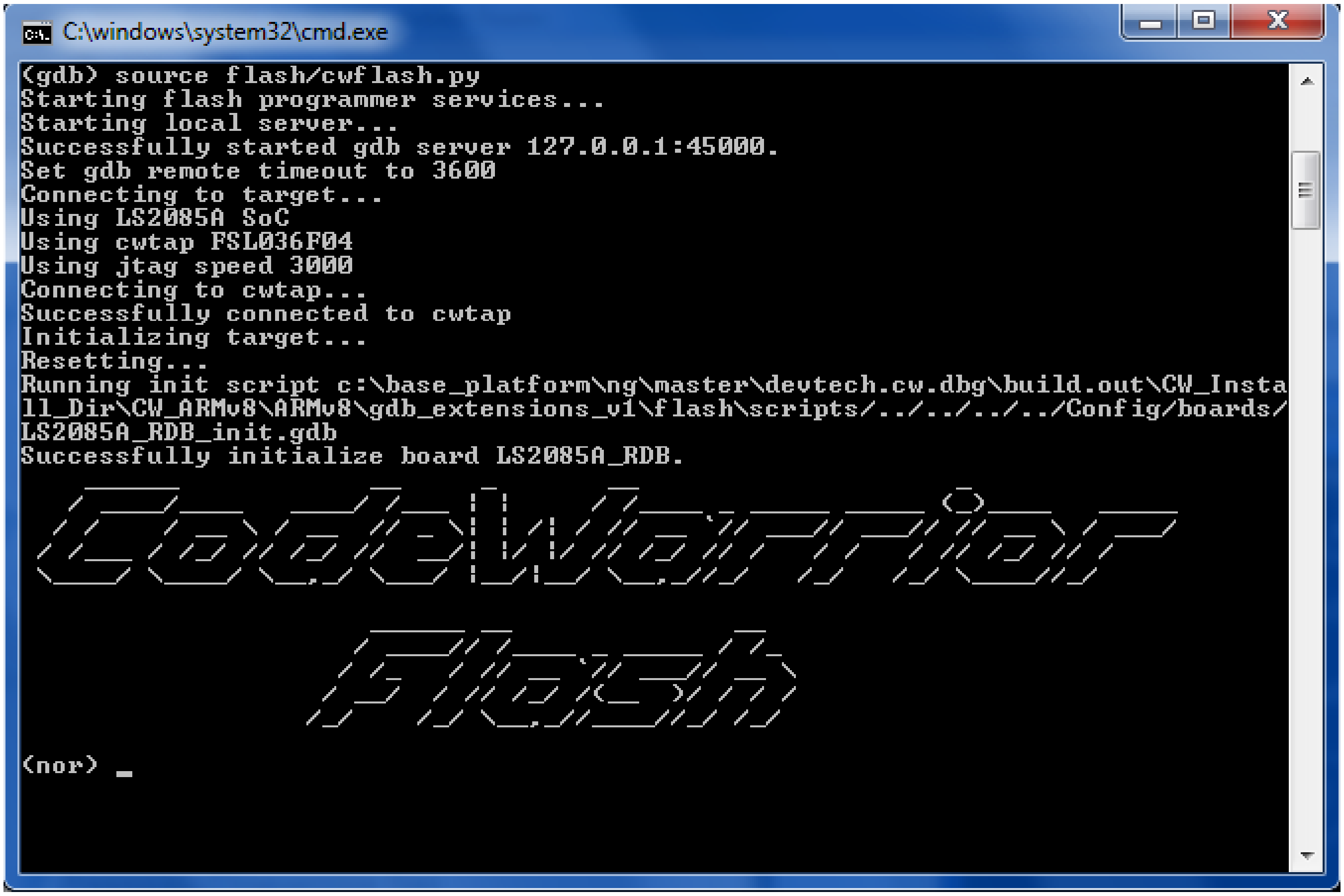
1. Open a terminal and switch to the following location:

   ```
   {CW Install Dir}\CW_ARMv8\ARMv8\gdb\bin
   ```

2. Start GDB from this location:
   - Windows: Run `aarch64-fsl-gdb.bat`
   - Linux: Run `./aarch64-fsl-gdb`
3. Execute `cwflash.py` script.

   ```
   source ../../gdb_extensions/flash/cwflash.py
   ```

   If the connection is successful, the output is shown as follows:



```
(gdb) source flash/cwflash.py
Starting flash programmer services...
Starting local server...
Successfully started gdb server 127.0.0.1:45000.
Set gdb remote timeout to 3600
Connecting to target...
Using LS2085A SoC
Using cwtap FSL036F04
Using jtag speed 3000
Connecting to cwtap...
Successfully connected to cwtap
Initializing target...
Resetting...
Running init script c:\base_platform\ng\master\devtech.cw.dbg\build.out\CW_Insta
ll_Dir\CW_ARMv8\ARMv8\gdb_extensions_v1\flash\scripts/../../../../Config/boards/
LS2085A_RDB_init.gdb
Successfully initialize board LS2085A_RDB.
    ______            __     _       __                    _
   / ____/____   ____/ /___ | |     / /____ _ _____ _____(_)____   _____
  / /    / __ \ / __  // _ \| | /| / // __ `// ___// ___// // __ \ / ___/
 / /___ / /_/ // /_/ //  __/| |/ |/ // /_/ // /   / /   / // /_/ // /
 \____/ \____/ \__,_/ \___/ |__/|__/ \__,_//_/   /_/   /_/ \____//_/


                      ______ __              __
                     / ____// /____ _ _____ / /_
                    / /_   / // __ `// ___// __ \
                   / __/  / // /_/ /(__  )/ / / /
                  /_/    /_/ \__,_//____//_/ /_/

(nor) _
```

Figure 7-1. Output

7.3 Using flash programmer

This section explains the operations supported by flash programmer.

- Erase flash memory
- Write binary file in flash memory
- Dump flash memory content into binary file

7.3.1 Erase flash memory

This topic explains command to erase an area of flash device.

To erase an area of the flash device, use the following command:

```
fl_erase offset size
```

where:

- <offset>: Specifies the offset inside the device.
- <size>: Specifies the size of the area that will be erased.

For example:

```
fl_ erase 0x40000 0x100
```

Type `fl_erase -h` for command help.

7.3.2 Write binary file in flash memory

This topic explains command to write binary file in the flash memory.

To write binary file in the flash memory, use the following command:

```
fl_write offset data [size] [--erase]
```

where:

- <offset>: Specifies the offset inside the device. If offset is not specified, it is assumed to be 0.
- <data>: Specifies the path to the file to be written in the flash or a hex sequence.
- <size>: Specifies how much data from the file should be written. If file size is not specified, entire file is written.
- <erase>: Specifies if erase should be performed first.

For example:

```
fl_write 0x40000 u-boot.bin --erase
```

Type `fl_write -h` for command help.

NOTE

The path to binary file must not contain spaces.

7.3.3 Dump flash memory content into binary file

This topic explains command to dump the contents of the flash memory into a binary file.

To dump the contents of the flash memory into a binary file, use the following command:

```
fl_dump offset size [-f FILE]
```

where:

- <offset>: Specifies the offset inside the device.
- <size>: Specifies the size of data to be read.
- <-f>: Specifies the path to the file where the data will be saved.

For example:

```
fl_dump 0x40000 0x20000 -f dump.bin
```

Type `fl_dump -h` for command help.

NOTE

The path to binary file must not contain spaces.

7.4 Switch current device used for flash programming

This topic explains command to switch current device used for flash programming.

To switch the current device used for flash programming, use the following command:

```
fl_current flash_type
```

For example:

```
fl_current nor
```

NOTE

If the command succeeds, the output appears as shown in -.

Chapter 8
Use Cases

This chapter lists:

- U-Boot debug
- Linux application debug
- Linux kernel debug

8.1 U-Boot debug

This topic describes the steps required to perform a U-Boot debug using CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.

This topic lists the steps to:

- Build the U-Boot sources and the auxiliary tools.
- Perform U-Boot debug in CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.

8.1.1 U-Boot setup

This topic explains U-Boot build.

For details on U-Boot build, refer SDK Documentation.

8.1.2 Create an ARMv8 project for U-Boot debug

This topic explains steps to create an ARMv8 bare metal project for U-Boot debug.

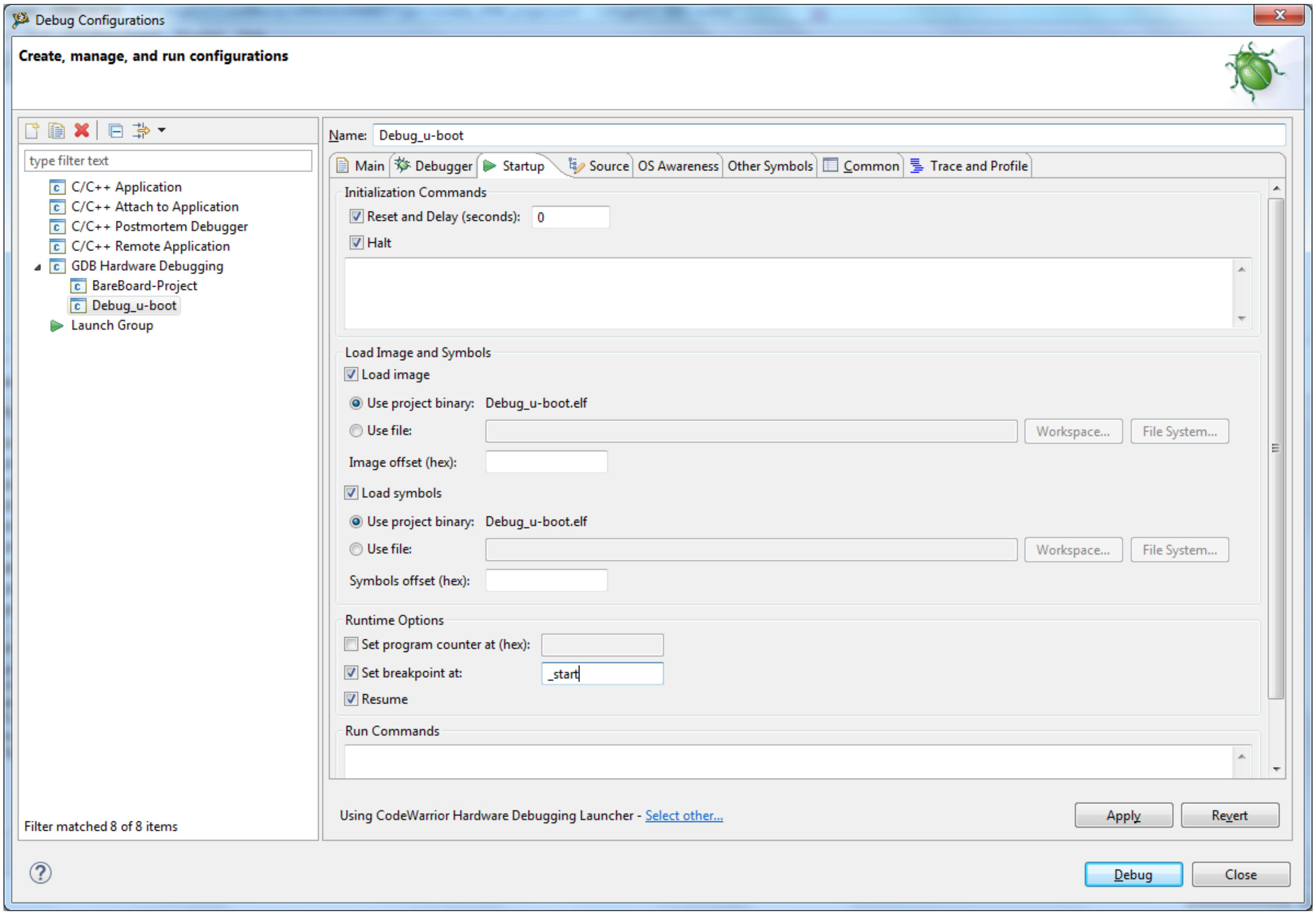
To create an ARMv8 bare metal project for U-Boot debug, perform these steps:

1. Open CodeWarrior for ARMv8.
2. Import a U-Boot image as described in CodeWarrior ELF Importer wizard.

3. Select **Run > Debug Configurations** to open the Debug Configurations dialog.
4. Click on the **Startup** tab.
   a. Set breakpoint at: _start.
   b. Select the **Resume** checkbox.

NOTE

Step (b) should be done only if nothing is running yet on the target board, or in case you have just started the target board, but have not started U-Boot. However, in case you simply attach it to a running U-Boot session the above step should be skipped. PC will reflect the current PC while U-Boot is running.



Name: Debug_u-boot

Initialization Commands

☑ Reset and Delay (seconds): 0

☑ Halt

Load Image and Symbols

☑ Load image

● Use project binary: Debug_u-boot.elf

○ Use file:

Image offset (hex):

☑ Load symbols

● Use project binary: Debug_u-boot.elf

○ Use file:

Symbols offset (hex):

Runtime Options

☐ Set program counter at (hex):

☑ Set breakpoint at: _start

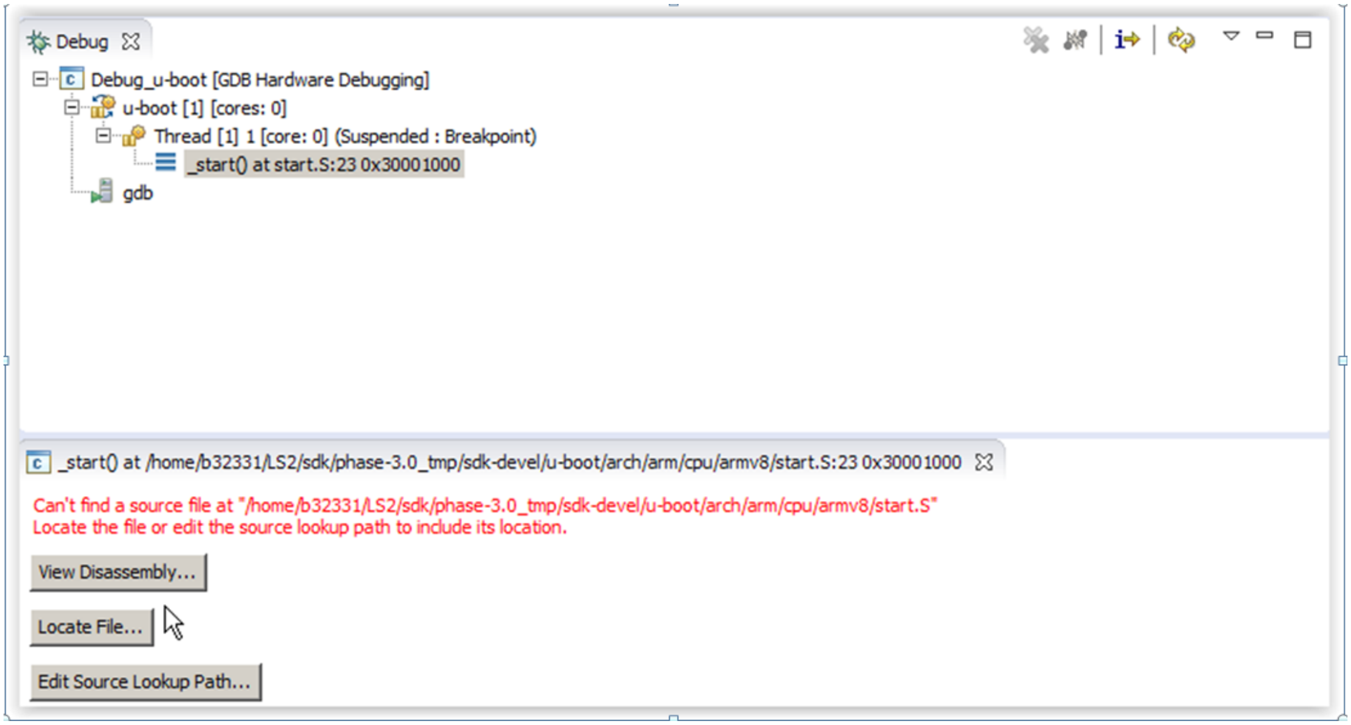
☑ Resume

Run Commands

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 8-1. Startup tab

5. Set up the target connection configuration, as explained in Configuring Target.
6. Click the **Debug** button to initiate the debug session. The debugger should stop at _start symbol.



```
Debug_u-boot [GDB Hardware Debugging]
  u-boot [1] [cores: 0]
    Thread [1] 1 [core: 0] (Suspended : Breakpoint)
      _start() at start.S:23 0x30001000
  gdb
```

_start() at /home/b32331/LS2/sdk/phase-3.0_tmp/sdk-devel/u-boot/arch/arm/cpu/armv8/start.S:23 0x30001000

Can't find a source file at "/home/b32331/LS2/sdk/phase-3.0_tmp/sdk-devel/u-boot/arch/arm/cpu/armv8/start.S"
Locate the file or edit the source lookup path to include its location.

Figure 8-2. Debugger stops at _start symbol

8.1.3 U-Boot debug support

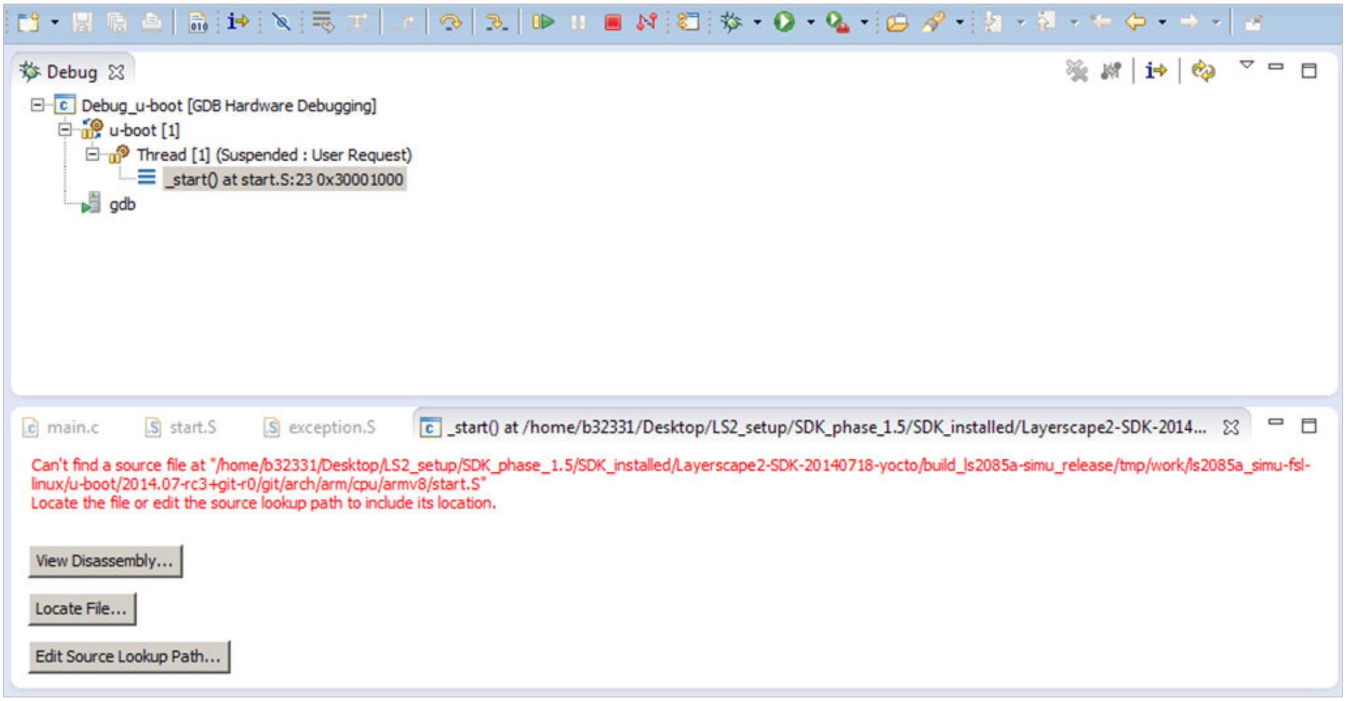
This section explains steps to perform U-Boot debug in CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.

8.1.3.1 Setting the source path mapping

This topic explains steps to load symbols and set source path mapping.

To load symbols and set source path mapping, perform these steps:

1. Locate the file suggested by the debugger.


Debug
main.c
start.S
exception.S
_start() at /home/b32331/Desktop/LS2_setup/SDK_phase_1.5/SDK_installed/Layerscape2-SDK-2014...
View Disassembly...
Locate File...
Edit Source Lookup Path...

```
Debug_u-boot [GDB Hardware Debugging]
  u-boot [1]
    Thread [1] (Suspended : User Request)
      _start() at start.S:23 0x30001000
  gdb
```

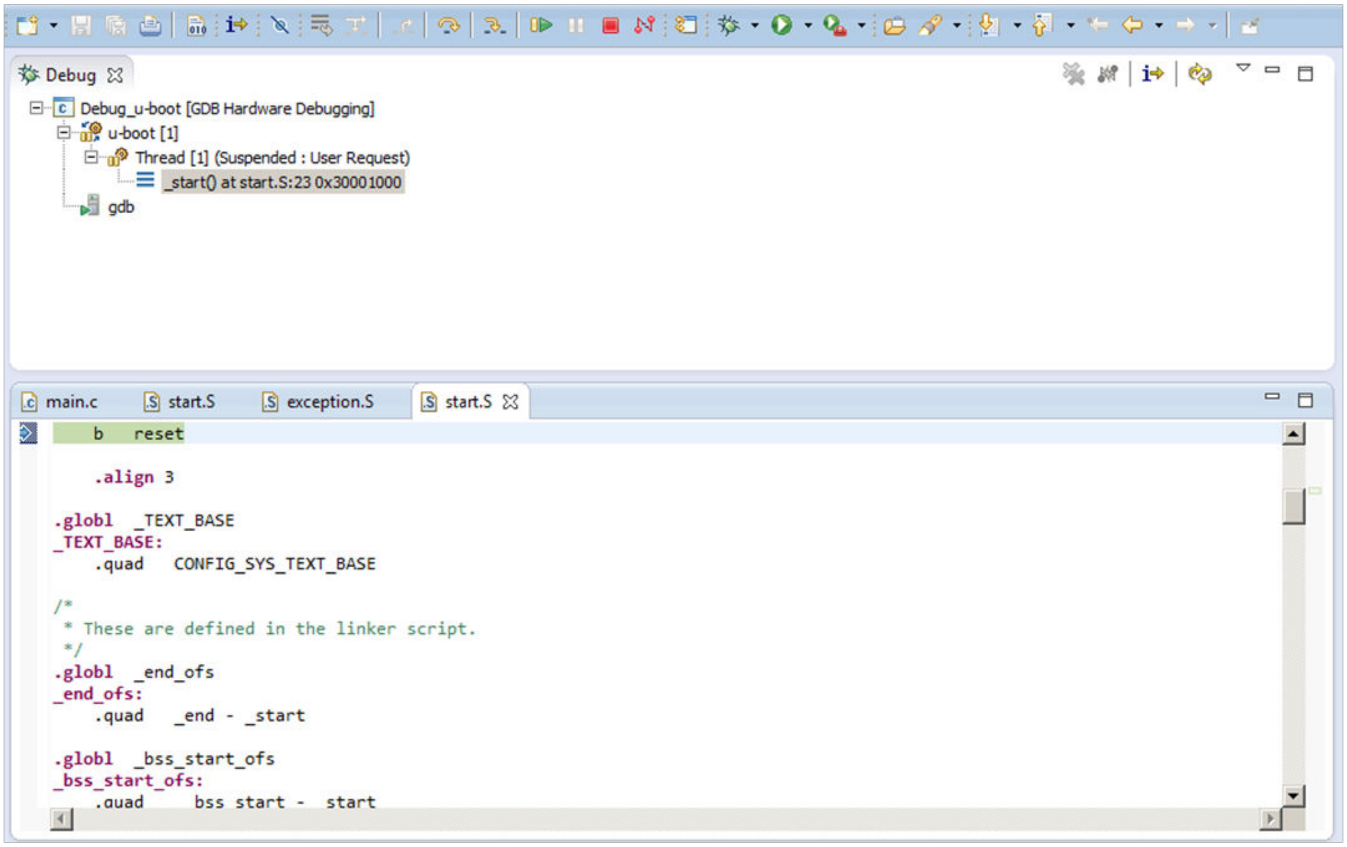
```
Can't find a source file at "/home/b32331/Desktop/LS2_setup/SDK_phase_1.5/SDK_installed/Layerscape2-SDK-20140718-yocto/build_ls2085a-simu_release/tmp/work/ls2085a_simu-fsl-linux/u-boot/2014.07-rc3+git-r0/git/arch/arm/cpu/armv8/start.S"
Locate the file or edit the source lookup path to include its location.
```

Figure 8-3. Locate source

2. The stack and the source views appears as in the following figure.

NOTE

You can add a static map entry using the Edit Source Lookup Path button to avoid locating file using the Locate File button, whenever a new file is requested.



```
Debug_u-boot [GDB Hardware Debugging]
  u-boot [1]
    Thread [1] (Suspended : User Request)
      _start() at start.S:23 0x30001000
  gdb
```

```
    b   reset

    .align 3

.globl  _TEXT_BASE
_TEXT_BASE:
    .quad   CONFIG_SYS_TEXT_BASE

/*
 * These are defined in the linker script.
 */
.globl  _end_ofs
_end_ofs:
    .quad   _end - _start

.globl  _bss_start_ofs
_bss_start_ofs:
    .quad    _bss_start - _start
```

Figure 8-4. Stack and sources

3. Click the **Resume** button. Alternatively, press the F8 key.

NOTE

If everything is setup correctly and the target is simulator, clicking the Resume button (F8), will show the next U-Boot log and the build time in the tio_console from the Linux machine. The log will be available within 4-5 seconds after clicking the button.

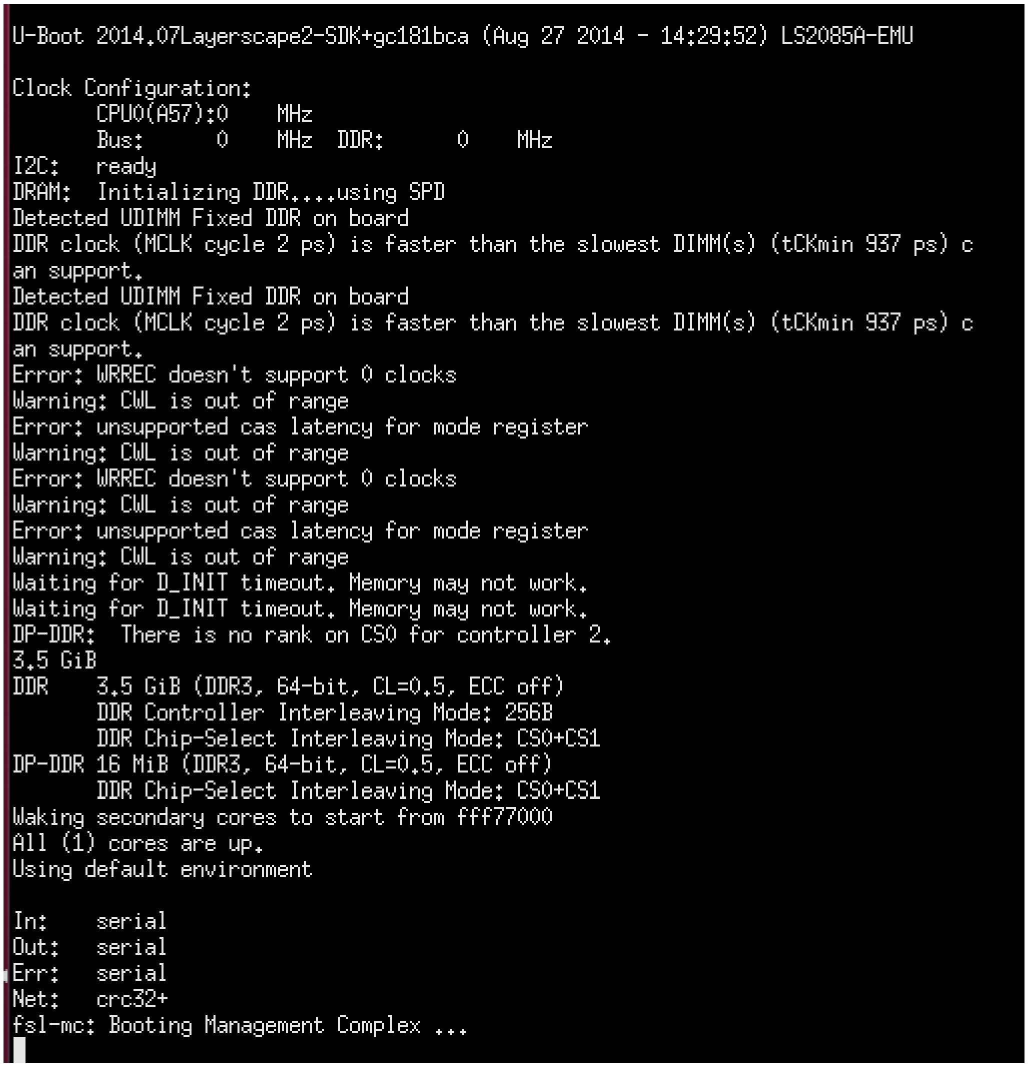
4. If you want to start the U-Boot debug again, close/terminate the actual connection. If you are using the simulator target, stop the simulator consoles, restart the simulator consoles, and debug again.

NOTE

Currently, the restart/reset features are not supported by simulator.

NOTE

If you want to attach to the same U-Boot session, disconnect the CodeWarrior software and reconnect again. You will not need to set the PC and the path mapping is correct.



```
U-Boot 2014.07Layerscape2-SDK+gc181bca (Aug 27 2014 - 14:29:52) LS2085A-EMU

Clock Configuration:
       CPU0(A57):0    MHz
       Bus:      0    MHz  DDR:      0    MHz
I2C:   ready
DRAM:  Initializing DDR....using SPD
Detected UDIMM Fixed DDR on board
DDR clock (MCLK cycle 2 ps) is faster than the slowest DIMM(s) (tCKmin 937 ps) c
an support.
Detected UDIMM Fixed DDR on board
DDR clock (MCLK cycle 2 ps) is faster than the slowest DIMM(s) (tCKmin 937 ps) c
an support.
Error: WRREC doesn't support 0 clocks
Warning: CWL is out of range
Error: unsupported cas latency for mode register
Warning: CWL is out of range
Error: WRREC doesn't support 0 clocks
Warning: CWL is out of range
Error: unsupported cas latency for mode register
Warning: CWL is out of range
Waiting for D_INIT timeout. Memory may not work.
Waiting for D_INIT timeout. Memory may not work.
DP-DDR:  There is no rank on CS0 for controller 2.
3.5 GiB
DDR    3.5 GiB (DDR3, 64-bit, CL=0.5, ECC off)
       DDR Controller Interleaving Mode: 256B
       DDR Chip-Select Interleaving Mode: CS0+CS1
DP-DDR 16 MiB (DDR3, 64-bit, CL=0.5, ECC off)
       DDR Chip-Select Interleaving Mode: CS0+CS1
Waking secondary cores to start from fff77000
All (1) cores are up.
Using default environment

In:    serial
Out:   serial
Err:   serial
Net:   crc32+
fsl-mc: Booting Management Complex ...
```

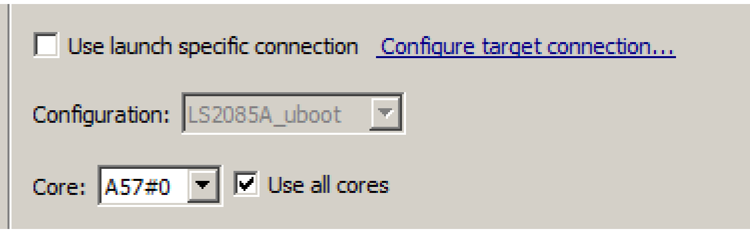
Figure 8-5. tio_console

8.1.3.2 Debug capabilities

This topic explains steps to bring-up the U-Boot.

1. The multicore debug is also supported if you want to inspect the secondary cores after release in the last stage.
   a. Select the **Use all cores** checkbox in the **Debugger** tab.

Figure 8-6. Debugger tab



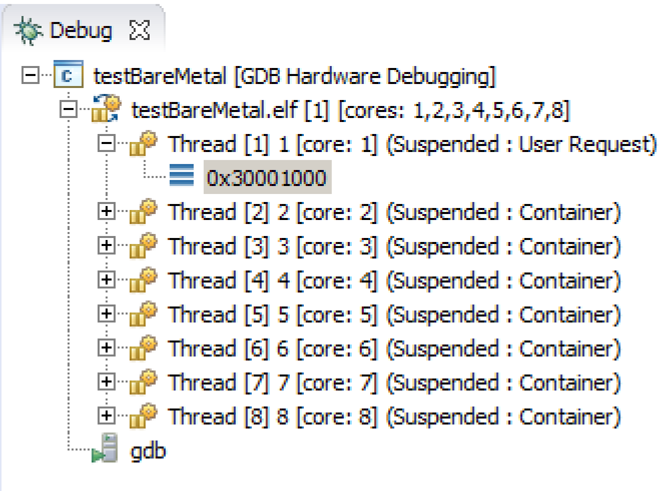
☐ Use launch specific connection Configure target connection...

Configuration: LS2085A_uboot

Core: A57#0 ☑ Use all cores

b. When teh debugging startes, you can see stack/registers for every core. Note that the run control is per SoC and not per core.

Figure 8-7. Debug view



```
Debug
testBareMetal [GDB Hardware Debugging]
  testBareMetal.elf [1] [cores: 1,2,3,4,5,6,7,8]
    Thread [1] 1 [core: 1] (Suspended : User Request)
      0x30001000
    Thread [2] 2 [core: 2] (Suspended : Container)
    Thread [3] 3 [core: 3] (Suspended : Container)
    Thread [4] 4 [core: 4] (Suspended : Container)
    Thread [5] 5 [core: 5] (Suspended : Container)
    Thread [6] 6 [core: 6] (Suspended : Container)
    Thread [7] 7 [core: 7] (Suspended : Container)
    Thread [8] 8 [core: 8] (Suspended : Container)
  gdb
```

2. Double-click a line to inspect breakpoints. You can inspect these using:
   - **Breakpoints** view
   - `info breakpoints` command from the GDB shell.
3. You can perform the step operations till the U-Boot boots up.

8.2 Linux application debug

This document describes the steps required to perform Linux Application Debug using CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA. This document lists the steps to:

- Build the Linux sources and auxiliary tools
- Networking support
- Perform Linux application debug in CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA

8.2.1 Linux setup

For details on Linux setup, refer SDK Documentation.

8.2.2 Network setup after booting the Linux on simulator

This section is only needed for Linux application debug when running on the simulator:

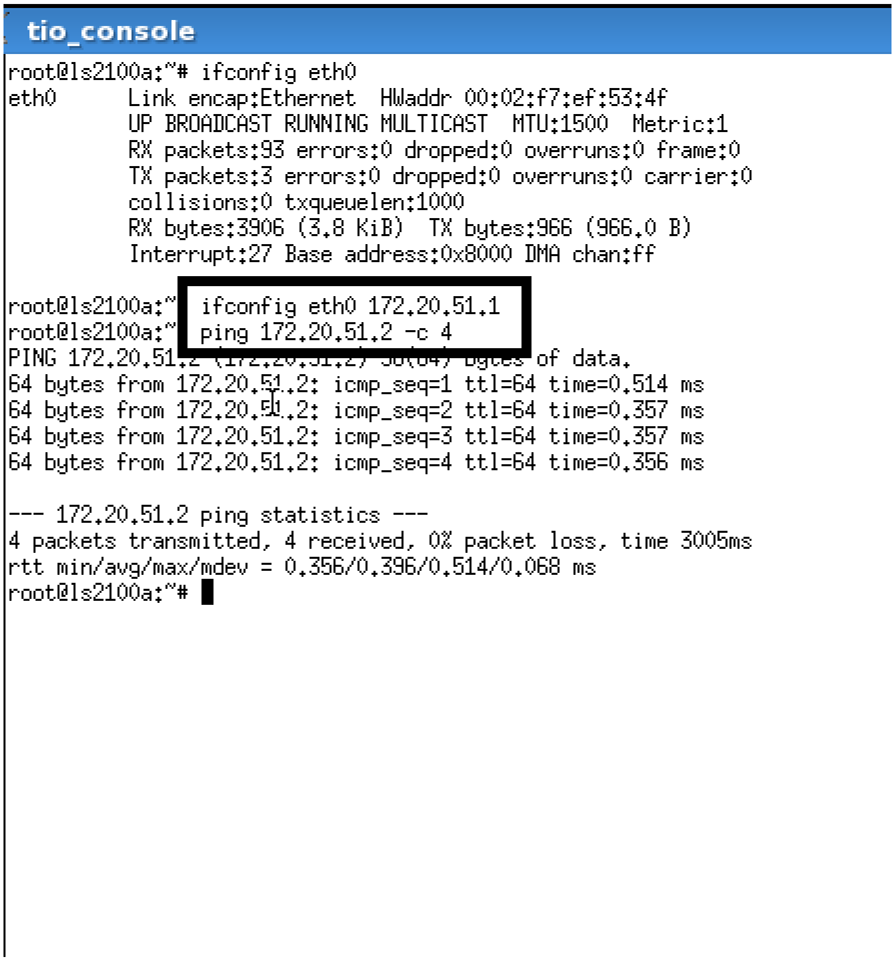
1. After Linux is booted on the simulator target, on the linux host PC, create a virtual interface used to communicate with the simulator, using next command:

```
<path_to_ls2-sim-support/scripts>./tuntap_if_configure.sh create ARM1 5A:F2:FE:A4:93:48
172.20.51.2
```

2. After above command, you should make a bridge between the virtual local interface ARM1 and WRIOP0 mac #1, using next command:

```
<path_to_ls2-sim-support/scripts>./start_tio_bridge.sh -m w0_m4 -n ARM1
```

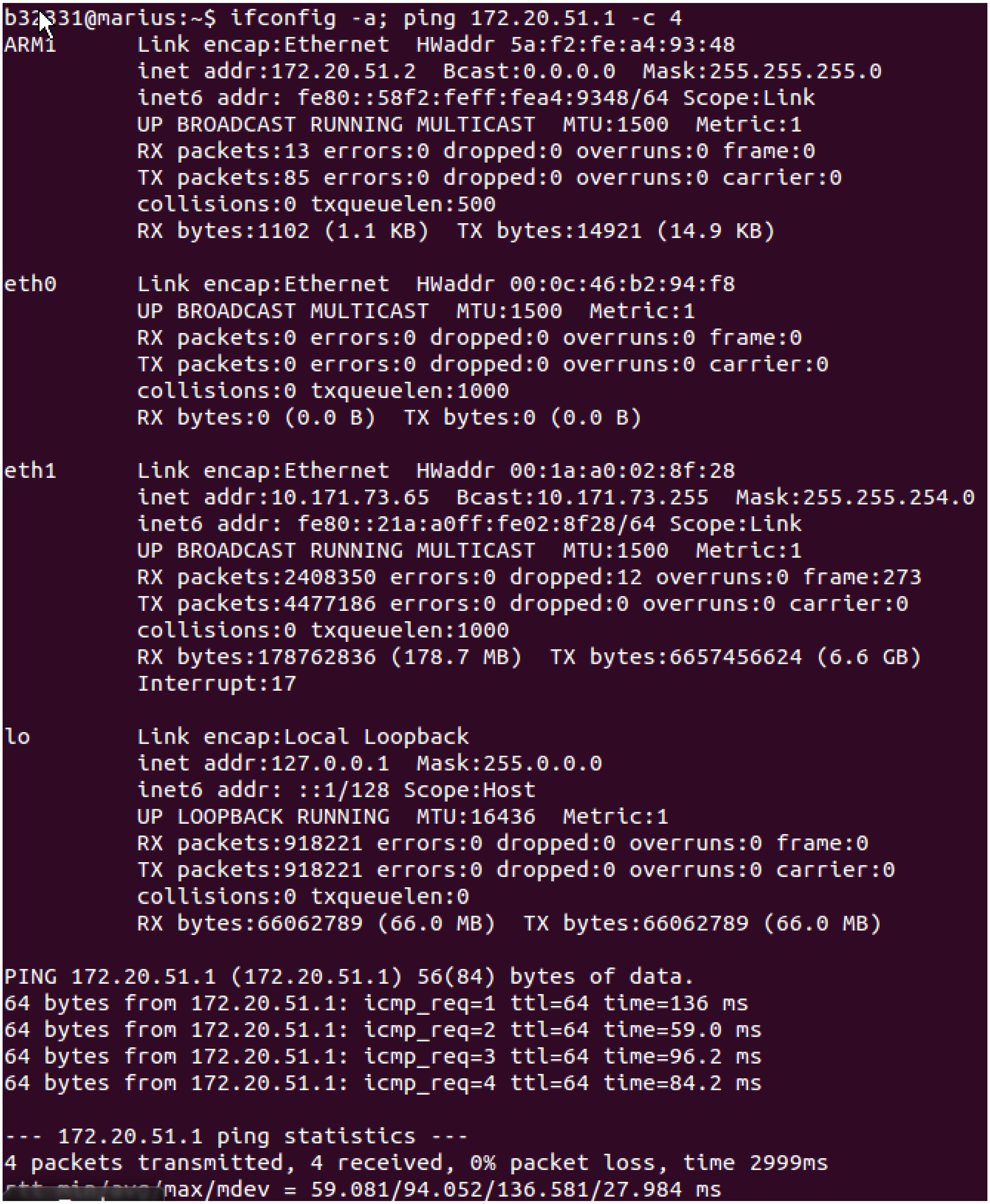
3. On the Linux booted on the simulator target, run **ifconfig** command to provide an IP address, as shown below. Also you can test the connectivity between eth0 (embedded linux) to ARM1 (virtual NIC on the Linux Host PC) using **ping**



```
root@ls2100a:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:02:f7:ef:53:4f
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:93 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3906 (3.8 KiB)  TX bytes:966 (966.0 B)
          Interrupt:27 Base address:0x8000 DMA chan:ff

root@ls2100a:~# ifconfig eth0 172.20.51.1
root@ls2100a:~# ping 172.20.51.2 -c 4
PING 172.20.51.2 (172.20.51.2) 56(84) bytes of data.
64 bytes from 172.20.51.2: icmp_seq=1 ttl=64 time=0.514 ms
64 bytes from 172.20.51.2: icmp_seq=2 ttl=64 time=0.357 ms
64 bytes from 172.20.51.2: icmp_seq=3 ttl=64 time=0.357 ms
64 bytes from 172.20.51.2: icmp_seq=4 ttl=64 time=0.356 ms

--- 172.20.51.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 0.356/0.396/0.514/0.068 ms
root@ls2100a:~# █
```

Figure 8-8. Run ifconfig command

4. You can also run a PING test from ARM1 (virtual NIC on the Linux Host PC) to eth0 (embedded linux) as shown below:



```
b32331@marius:~$ ifconfig -a; ping 172.20.51.1 -c 4
ARM1      Link encap:Ethernet  HWaddr 5a:f2:fe:a4:93:48
          inet addr:172.20.51.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::58f2:feff:fea4:9348/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:1102 (1.1 KB)  TX bytes:14921 (14.9 KB)

eth0      Link encap:Ethernet  HWaddr 00:0c:46:b2:94:f8
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet  HWaddr 00:1a:a0:02:8f:28
          inet addr:10.171.73.65  Bcast:10.171.73.255  Mask:255.255.254.0
          inet6 addr: fe80::21a:a0ff:fe02:8f28/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2408350 errors:0 dropped:12 overruns:0 frame:273
          TX packets:4477186 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:178762836 (178.7 MB)  TX bytes:6657456624 (6.6 GB)
          Interrupt:17

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:918221 errors:0 dropped:0 overruns:0 frame:0
          TX packets:918221 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:66062789 (66.0 MB)  TX bytes:66062789 (66.0 MB)

PING 172.20.51.1 (172.20.51.1) 56(84) bytes of data.
64 bytes from 172.20.51.1: icmp_req=1 ttl=64 time=136 ms
64 bytes from 172.20.51.1: icmp_req=2 ttl=64 time=59.0 ms
64 bytes from 172.20.51.1: icmp_req=3 ttl=64 time=96.2 ms
64 bytes from 172.20.51.1: icmp_req=4 ttl=64 time=84.2 ms

--- 172.20.51.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 59.081/94.052/136.581/27.984 ms
```

Figure 8-9. PING test from ARM1 (virtual NIC on the Linux Host PC) to eth0

NOTE

If the TCP connection between ARM1 (virtual NIC on the Linux Host PC) and eth0 (embedded linux) is slow when the ping command is executed (e.g. the time is more than 200 ms), you need to increase the timeout limit in GDB to wait for the remote target to respond. The default value is 2 seconds. You can add the command in the .gdbinit file. To set the timeout limit at 10 seconds, the command is: set remotetimeout 10

5. You need to create SSH tunnels to access the internal IP addresses of the embedded Linux of the simulator from a different machine. This step is required only when CodeWarrior and the simulator are running on different machines.
   a. Create SSH tunnel for port where gdbserver is running on the embedded Linux using next command:

```
ssh -L <Extenal IP Linux Host>:<PORT for gdbserver, e.g. 1234>:<IP of the Embedded
Linux in Simulator>:<port for gdbserver, e.g. 1234> root@<IP address of the
Embedded Linux in
simulator>e.g.: sudo ssh -L 10.171.73.65:1234:172.20.51.1:1234  root@172.20.51.1
```

   b. Create SSH tunnel for port where SSH server is running on the embedded linux using next command:

```
ssh -L <Extenal IP Linux Host>:<PORT for ssh server forward, e.g. 81>:<IP of the
Embedded Linux in Simulator>:<default port for ssh server - 22> root@<IP address of
the Embedded
Linux in simulator>e.g.: sudo ssh -L 10.171.73.65:81:172.20.51.1:22 root@172.20.51.1
```

NOTE

If you are not running the tunneling commands for the first time, you may receive a warning that host identification has changed. For removing that warning and to be able to make the tunneling, you need to remove the key linux target from known_hosts using this command:

ssh-keygen -f "/root/.ssh/known_hosts" -R 172.20.51.1

6. On the embedded Linux, run the next command: `touch ~/.hushlogin`.

8.2.3 Debugging simple Linux application

This topic explains how to create a simple Linux application project, update RSE connection, enable full debug support, and debug the Lniux application project.

- Creating simple Linux application project

- Updating RSE connection
- Using sysroot
- Debugging Linux application project

8.2.3.1 Creating simple Linux application project

This topic explains steps to create a ARMv8 Linux application project.

To create a ARMv8 Linux application project:

1. Open CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.
2. Select **File > New > ARMv8 Stationary > Linux Application Debug > Hello World C Project**.
3. Specify a project name.
4. Click **Finish**.
5. Select the newly created Linux application project in the **Project Explorer** view.
6. Select **Project > Build project**.

8.2.3.2 Updating RSE connection

This topic explains how to change the settings in a default RSE connection.

The IP/hostname and the SCP port of the Linux target must be set to the correct values. For example, if your target is the simulator and the CodeWarrior software is running on a different machine (refer Network setup after booting the Linux on simulator), the IP Connection and SCP port must be changed accordingly to the values configured in step 3 of Network setup after booting the Linux on simulator. To change the default values perform the following steps:

1. Select **Windows > Show View > Other**.

   The **Show View** dialog appears.

2. Navigate to **Remote Systems > Remote Systems**.

   The **Remote Systems** view appears.

3. Right-click the default Remote System Explorer (RSE) connection, *ScpConnection*.
4. Select **Properties > Host**.

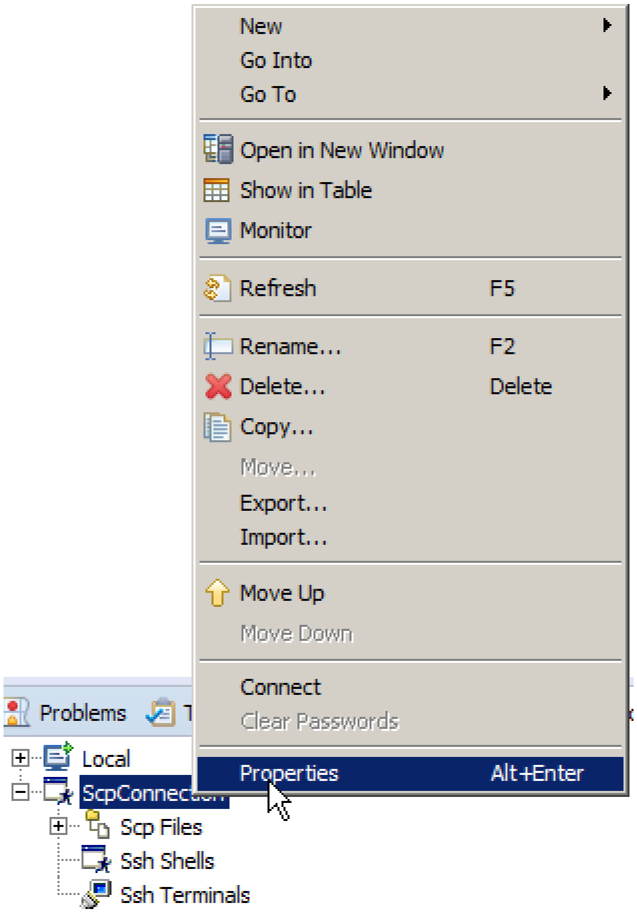
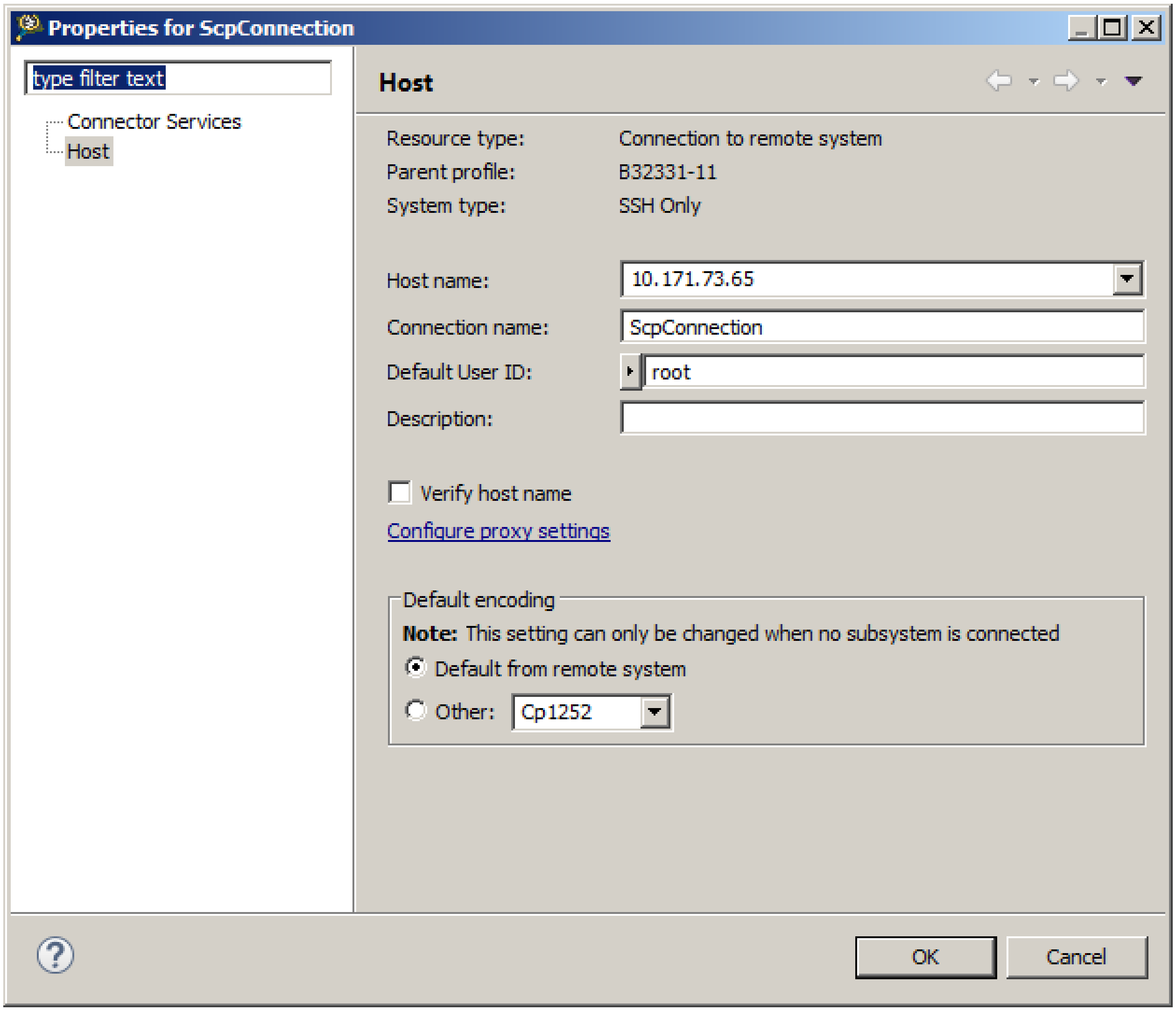


Figure 8-10. ScpConnection properties

The **Properties for ScpConnection** dialog appears.

5. Specify the IP of the Linux target in the **Host name** text box, and click **OK**.



Properties for ScpConnection

type filter text

Host

Resource type: Connection to remote system

Parent profile: B32331-11

System type: SSH Only

Host name: 10.171.73.65

Connection name: ScpConnection

Default User ID: root

Description:

☐ Verify host name

Configure proxy settings

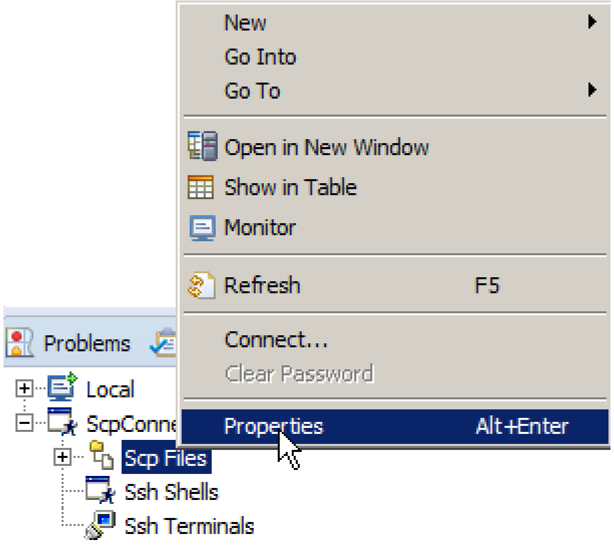
Default encoding

Note: This setting can only be changed when no subsystem is connected

● Default from remote system

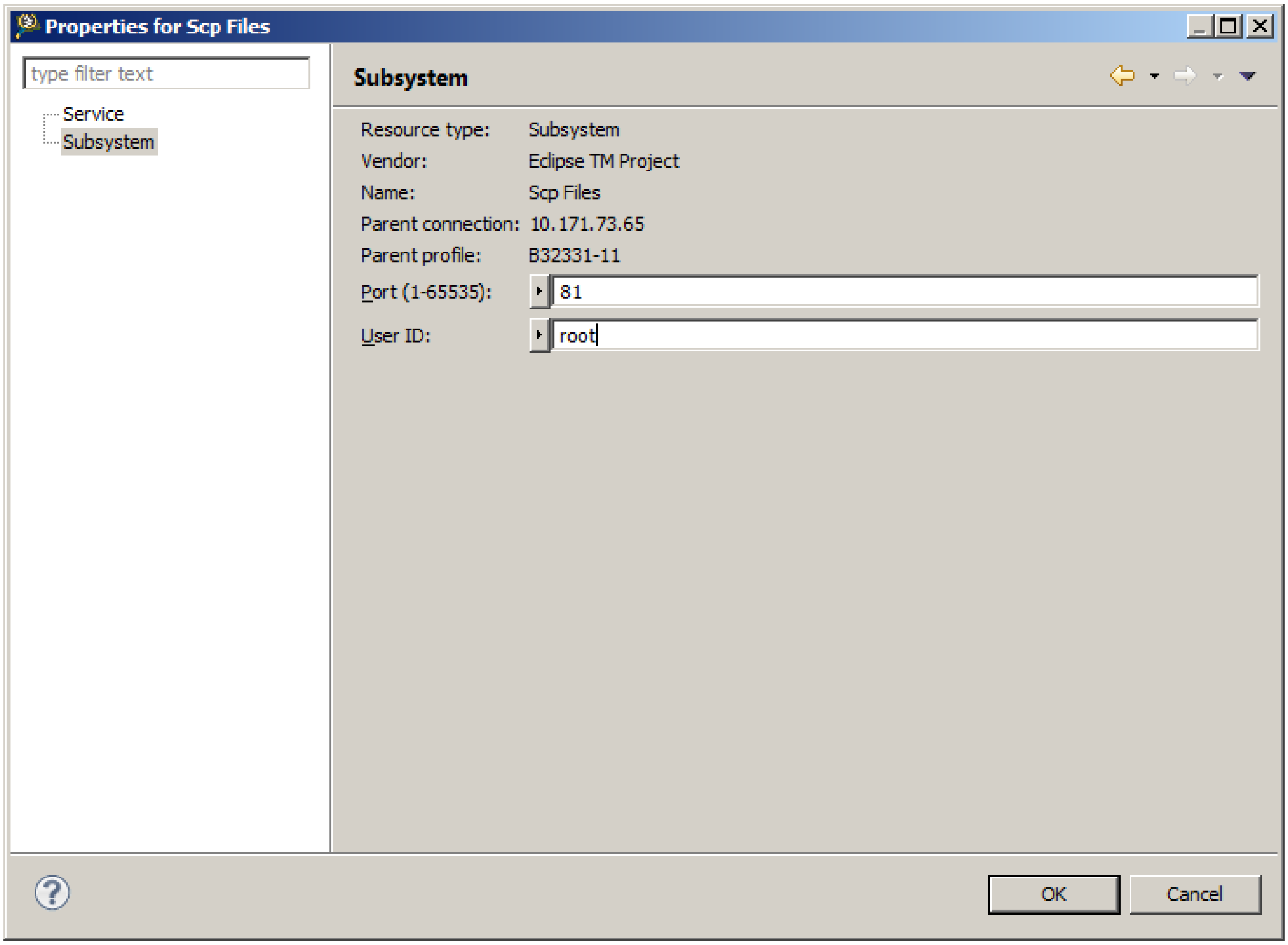
○ Other: Cp1252

6. Right-click **Scp Files** in the **Remote Systems** view, and select **Properties**.



The **Properties for Scp Files** dialog appears.

7. Select **Subsystem**.
8. Specify the port number. For example, 81 instead of 22.
9. Specify User ID as root.



Resource type: Subsystem
Vendor: Eclipse TM Project
Name: Scp Files
Parent connection: 10.171.73.65
Parent profile: B32331-11
Port (1-65535): 81
User ID: root

Figure 8-11. Select UserID

10. Click **OK**.

NOTE

For a full debug support, it is recommended to perform the steps in section, Using sysroot.

8.2.3.3 Using sysroot

This section is required only if you want to enable full debug support (inside target system libraries) for the Linux application project.

NOTE

> Before you proceed, ensure that you have completed all the steps in Updating RSE connection.

To enable full debug support for a Linux application project, perform these steps:

1. GDB should be configured to use the target system libraries.
    a. On the host PC, create a folder `rootfs` and a sub-directory `lib`.
    b. Copy the following libraries: `libc`, `ld`, `libphtread` in the `rootfs/lib/` folder. You can find these libraries at `${CW_Layout}\Cross_Tools\gcc-linaro-aarch64-linux-gnu-4.8.3\aarch64-linux-gnu`. Use the full library name as you see it on target, for example `libpthread.so.0`, `ld-linux-aarch64.so.1`, `libc.so.6`.
    c. Create a `*.gdbinit` file on the file system. For example, `test.gdbinit`
    d. Add following content in the `.gdbinit` file:

    ```
    set sysroot <host_path_to_rootfs>
    ```

    For example, `set sysroot C:\Users\u12345\Desktop\rootfs`

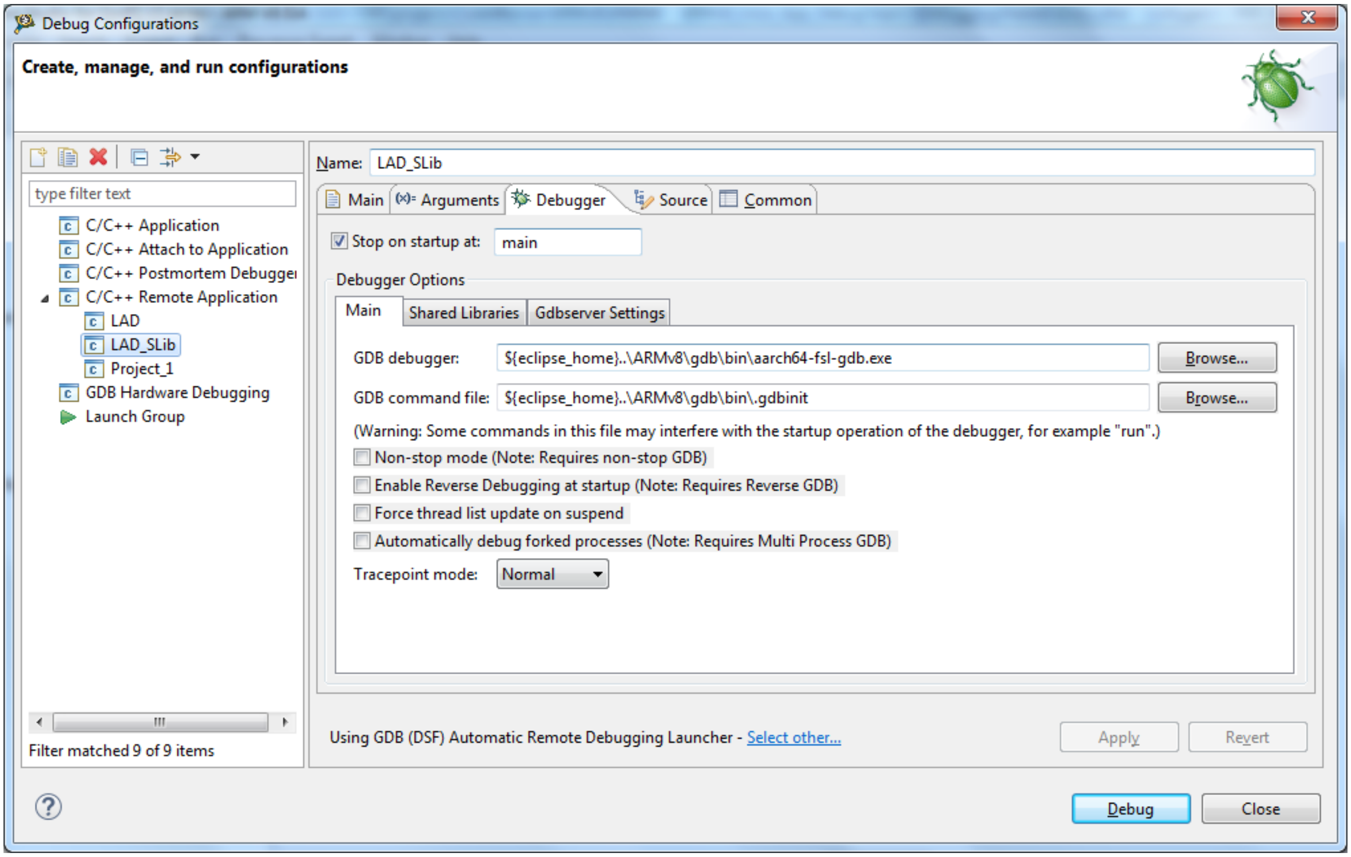
    NOTE

    > If you are running the CodeWarrior software on the same Linux machine where you have compiled the yocto, you can directly set up in the gdbinit file the sysroot from yocto:
    >
    > ```
    > set sysroot /home/u12345/Desktop/LS2_setup/SDK_phase_2.0/
    > Layerscape2-SDK-20140829-yocto/build_ls2085a-
    > simu_release/tmp/sysroots/ls2085a-simu
    > ```

2. Add missing settings in launch configuration file.
    a. Right-click the project and select **Debug As > Debug Configurations**.

    The **Debug Configurations** dialog appears.

    b. Expand **C/C++ Remote Application**, select the launch configuration for the Linux application project you want to debug.
    c. Click the **Main** sub tab in the **Debugger** tab.
    d. Browse to `*.gdbinit` path in **GDB command file** field.



Name: LAD_SLib

☑ Stop on startup at: main

Debugger Options

GDB debugger: ${eclipse_home}..\ARMv8\gdb\bin\aarch64-fsl-gdb.exe

GDB command file: ${eclipse_home}..\ARMv8\gdb\bin\.gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

☐ Non-stop mode (Note: Requires non-stop GDB)

☐ Enable Reverse Debugging at startup (Note: Requires Reverse GDB)

☐ Force thread list update on suspend

☐ Automatically debug forked processes (Note: Requires Multi Process GDB)

Tracepoint mode: Normal

Filter matched 9 of 9 items

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

Figure 8-12. Debugger tab - Main

e. Click **Apply**.

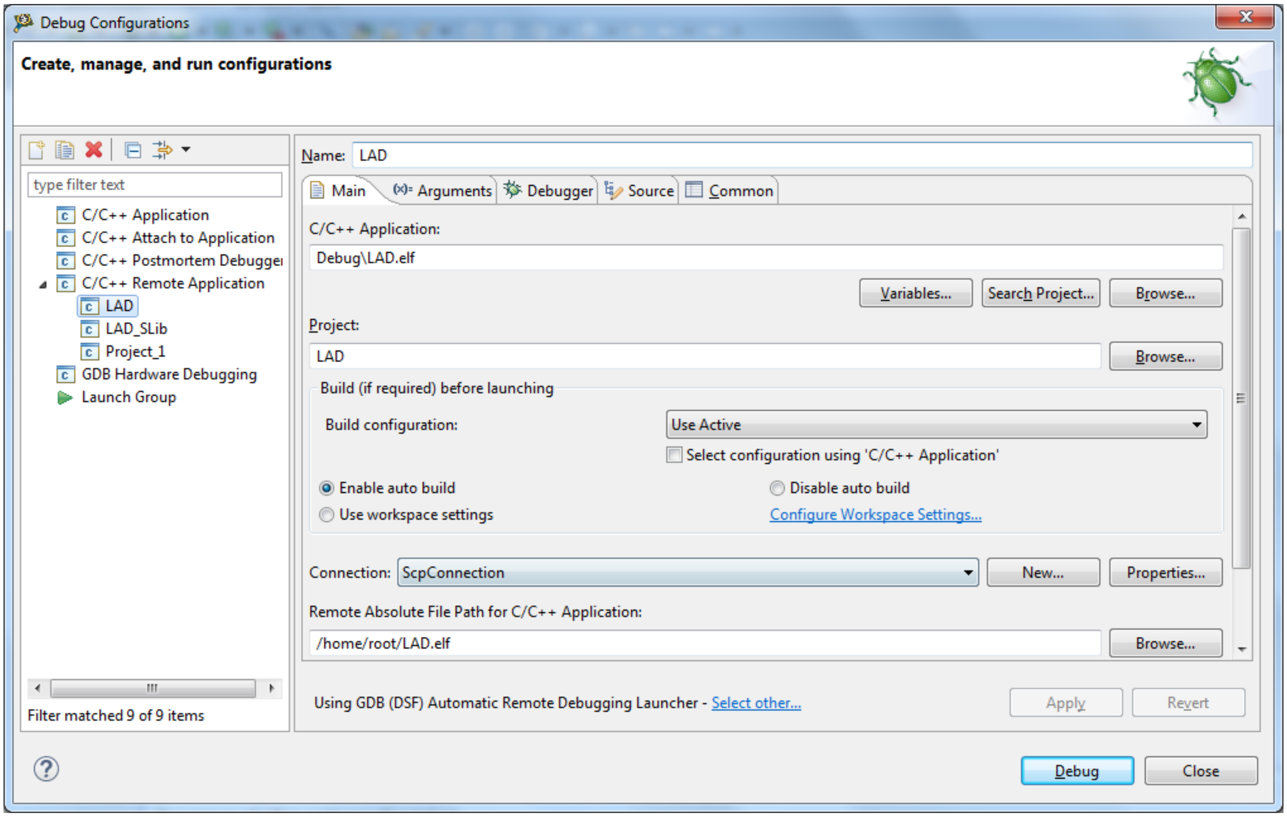
8.2.3.4 Debugging Linux application project

This topic explains steps to debug a Linux application project.

To debug a Linux application project:

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

2. In the **Debug Configuration** dialog, expand **C/C++ Remote Application** and select the launch configuration for the Linux application project you want to debug.



Debug Configurations

Create, manage, and run configurations

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
  - LAD
  - LAD_SLib
  - Project_1
- GDB Hardware Debugging
- Launch Group

Filter matched 9 of 9 items

Name: LAD

C/C++ Application: Debug\LAD.elf

Project: LAD

Build (if required) before launching

Build configuration: Use Active

□ Select configuration using 'C/C++ Application'

● Enable auto build ○ Disable auto build

○ Use workspace settings Configure Workspace Settings...

Connection: ScpConnection

Remote Absolute File Path for C/C++ Application: /home/root/LAD.elf

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

3. Click **Debug**.

8.2.4 Debugging a Linux application using a shared library

This topic explains:

- Creating Linux shared library project
- Updating RSE connection
- Using sysroot
- Debugging Linux shared library project

8.2.4.1 Creating Linux shared library project

To create an ARMv8 Linux application project using a shared library, perform these steps:

1. Open CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.
2. Select **File > New > ARMv8 Stationary > Linux Application Debug > Hello World C Shared Library Project**.

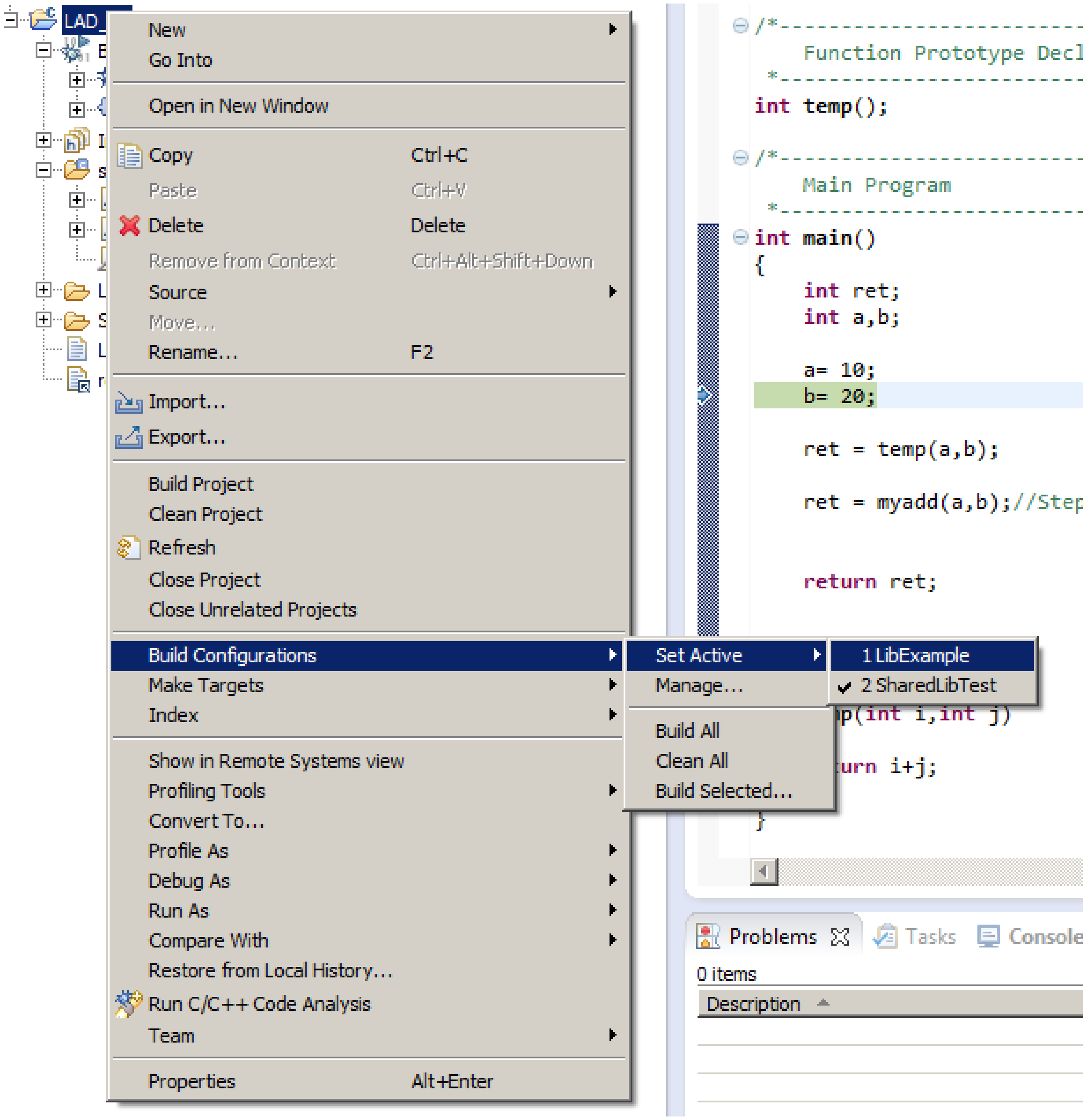
3. Provide a project name.
4. Click **Finish**.
5. Select the project node in the **Project Explorer** view.
6. Build all configurations:

   The project has two build configurations:

   - LibExample - Builds the shared library
   - SharedLibTest (the active configuration) - Uses the shared library

   a. Right-click the project and select **Build Configurations > Set Active > LibExample**.

   b. Build project. The `lib<project_name>.so` library is created.



```
/*------------------------
    Function Prototype Decl
 *------------------------
int temp();

/*------------------------
    Main Program
 *------------------------
int main()
{
    int ret;
    int a,b;

    a= 10;
    b= 20;

    ret = temp(a,b);

    ret = myadd(a,b);//Step

    return ret;

p(int i,int j)

urn i+j;

}
```

Figure 8-13. Build configurations

   c. Select the project, right-click and select **Build Configurations > Set Active > SharedLibTest**.
   d. Build project. The `<project_name>.elf` library is created.

8.2.4.2 Updating RSE connection

Refer to the steps in Updating RSE connection.

8.2.4.3 Updating launch configuration for Linux application using shared library

This topic explains steps to set the launch configuration for a Linux application project that uses a shared library

To set the launch configuration for a Linux application project that uses a shared library, perform the following steps:

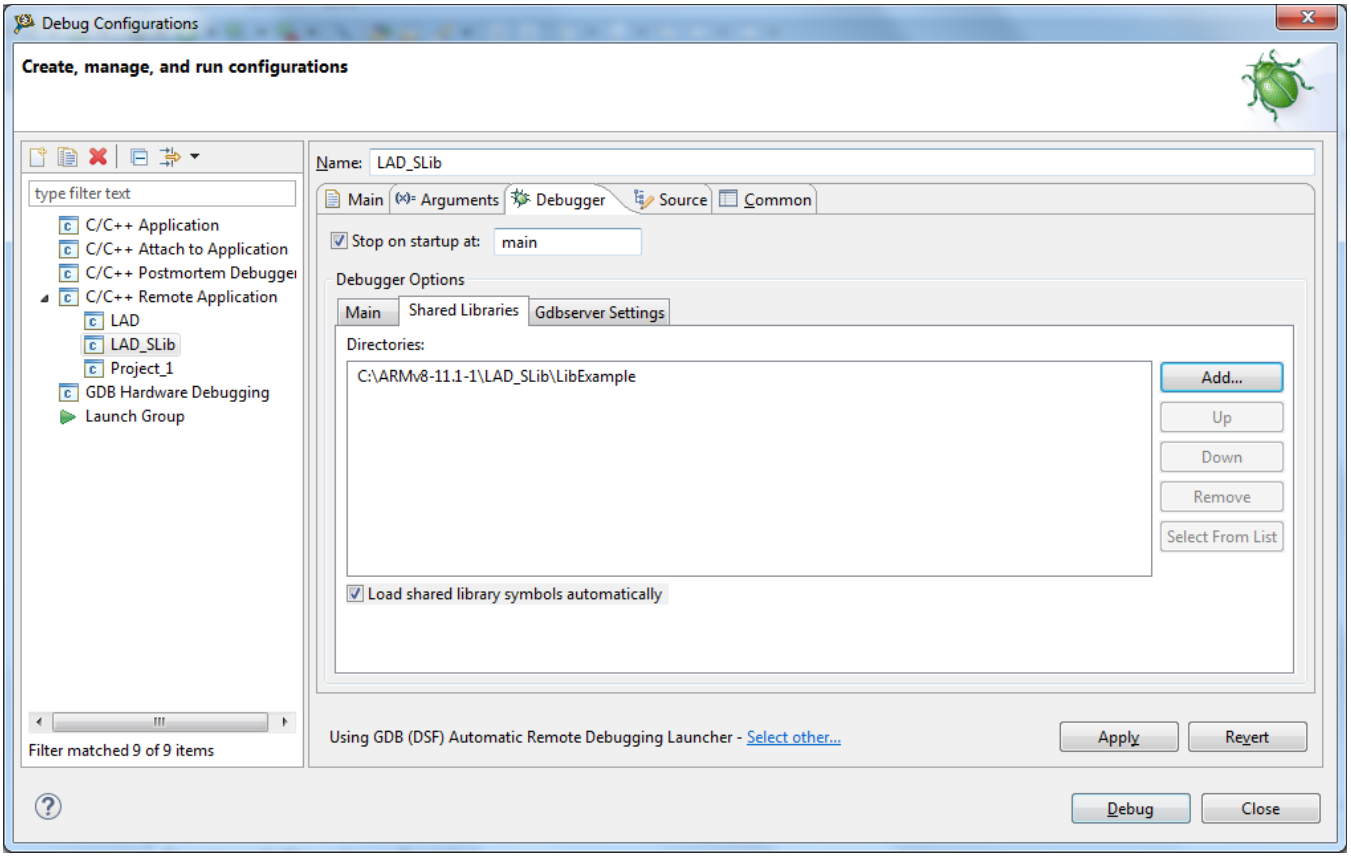
1. Perform all the steps in Using sysroot.
2. Manually download the `.so` shared library to the Linux target (to the`/lib` path).
3. Copy the `.so` shared library to the `sysroot` location. (Refer Using sysroot, step 1d)

   The location can be:
   a. The `rootfs/lib/` folder you created on your host PC (Refer Using sysroot, step 1a)
   b. The `lib` from the `sysroot` location from yocto, if you are using the CodeWarrior software on the same Linux machine where you have compiled the yocto and you are using the `sysroot` from yocto.

      Example:

      ```
       /home/u12345/Desktop/LS2_setup/SDK_phase_2.0/Layerscape2-SDK-20140829- yocto/
      build_ls2085a-simu_release/tmp/sysroots/ls2085a-simu/lib
      ```
4. Add missing settings in launch configuration file.
   a. Right-click the project and select **Debug As > Debug Configurations**. The **Debug Configurations** dialog appears.
   b. Expand **C/C++ Remote Application**, select the Linux shared library project you want to debug.
   c. Click the **Shared Libraries** sub tab and click **Add** to add the path to the `*.so` library you created in Creating Linux shared library project. The path is

      ```
      ${ProjDirPath}/LibExample
      ```



Name: LAD_SLib

☑ Stop on startup at: main

Directories:

C:\ARMv8-11.1-1\LAD_SLib\LibExample

☑ Load shared library symbols automatically

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

Figure 8-14. Debugger tab - Shared Libraries

d. Click **Apply**.

8.2.4.4 Debugging Linux shared library project

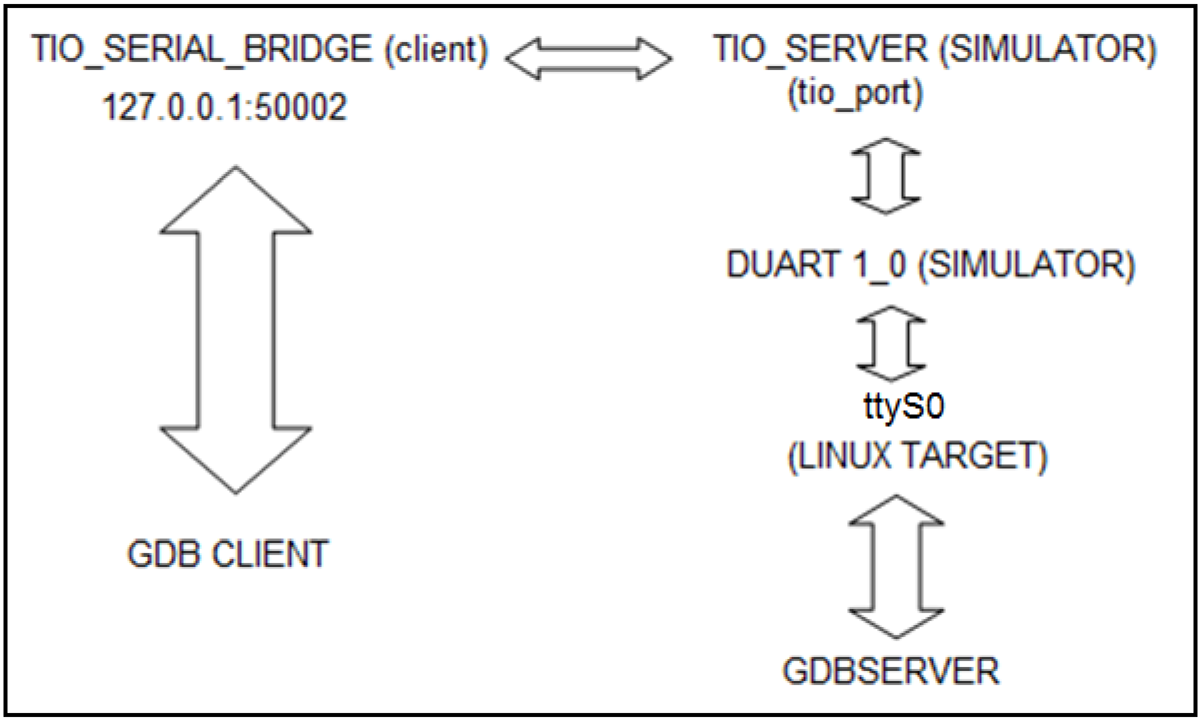
To debug Linux shared library project, refer to the steps in Debugging Linux application project

8.2.5 Troubleshooting

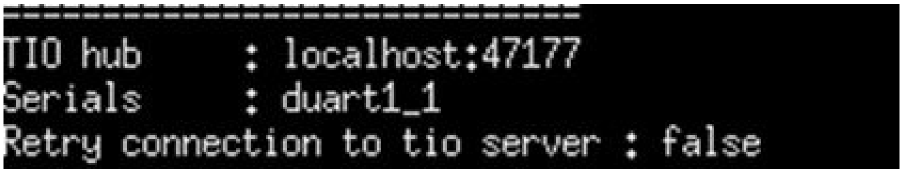
This topic explains steps to troubleshoot networking support in Linux target running on the simulator.

8.2.5.1 Networking

If networking support is not available on the simulator target, the CodeWarrior software and the target can still communicate over a virtual serial device. The simulator provides a TIO client acting as a pass-through between a simulated serial device (target) and TCP/IP port (host running the simulator). The full schema about how gdb client will interconnect with the gdb server can be observed below:



1. On the linux host, start in a new terminal window the tio serial bridge by using the TIO hub settings reported by tio_console: “TIO hub : localhost 47177”



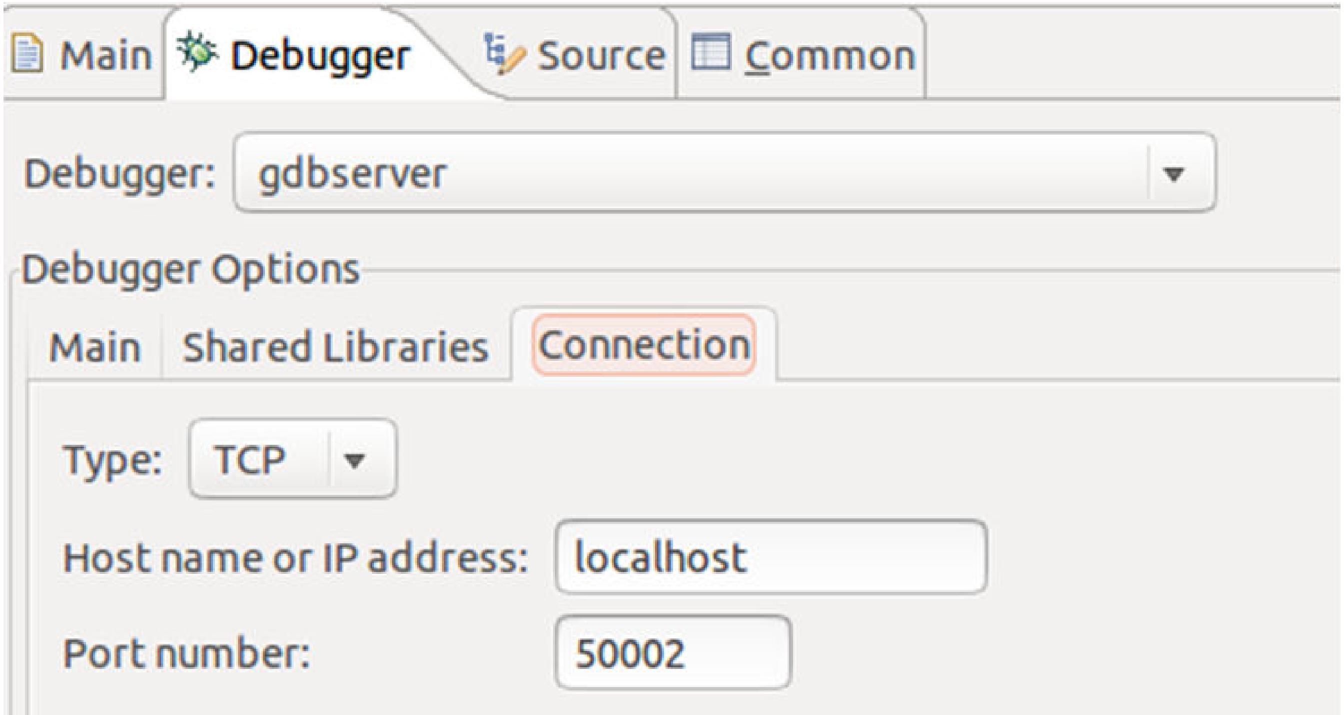
```
-----------------------------
TIO hub      : localhost:47177
Serials      : duart1_1
Retry connection to tio server : false
```

```
cd <sim_path>/dtsim_release/linux64/
./bin/tio_serial_bridge -ser duart1_0 -port 50002 -hub localhost:47177
```

2. On the linux target, start the gdbserver manually as below:

```
gdbserver --multi /dev/ttyS0
```

3. If your Linux Application is already deployed on the linux target you should make the steps from chapter 6.2 (attach to a running application or you can set up the application as a parameter for the gdbserver), but with the next gdb parameters in IDE:



Debugger: gdbserver

Type: TCP

Host name or IP address: localhost

Port number: 50002

4. If you want to download the application over gdb, please make next steps:
   a. cd <path_to>/Layout/ARMv8/gdb/bin
   b. ./aarch64-fsl-gdb
   c. set remotetimeout 10
   d. target extended-remote localhost:50002
   e. remote put <path_to_local_elf_file> <remote_elf_name>
5. Once the elf file is downloaded on the target go back to step 3 (attach to the application using gdbserver and gdb).
6. [Optional step] From gdb command line (without eclipse) can be performed next steps to make debug:
   a. cd <path_to>/Layout/ARMv8/gdb/bin
   b. ./aarch64-fsl-gdb
   c. set remotetimeout 10
   d. target extended-remote localhost:50002
   e. remote put <path_to_local_elf_file> <remote_elf_name>
   f. set remote exec-file a.elf
   g. file <path_to_local_elf_file>
   h. break main
   i. run

8.3 Linux kernel debug

This document describes the steps required to perform Linux kernel debug using CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA. This document explains:

- Building the U-Boot, Linux sources, and the auxiliary tools.
- Performing Linux Kernel debug in CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.

8.3.1 Linux Kernel setup

For details on Linux kernel build, refer SDK Documentation.

NOTE

In order to perform Linux kernel debug, please ensure that the kernel image is build with debug symbols. For enabling the debug symbols:

1. Run `bitbake virtual/kernel -c menuconfig`.

2. Go to **General Setup**, disable the option **Compile also drivers which will not load**. Note that without performing this step the option below will not appear in menuconfig.

3. Select **Kernel Hacking -> Compile-time checks and compiler options**, enable option “Compile the kernel with debug info”.

8.3.2 Create an ARMv8 project for Linux kernel debug

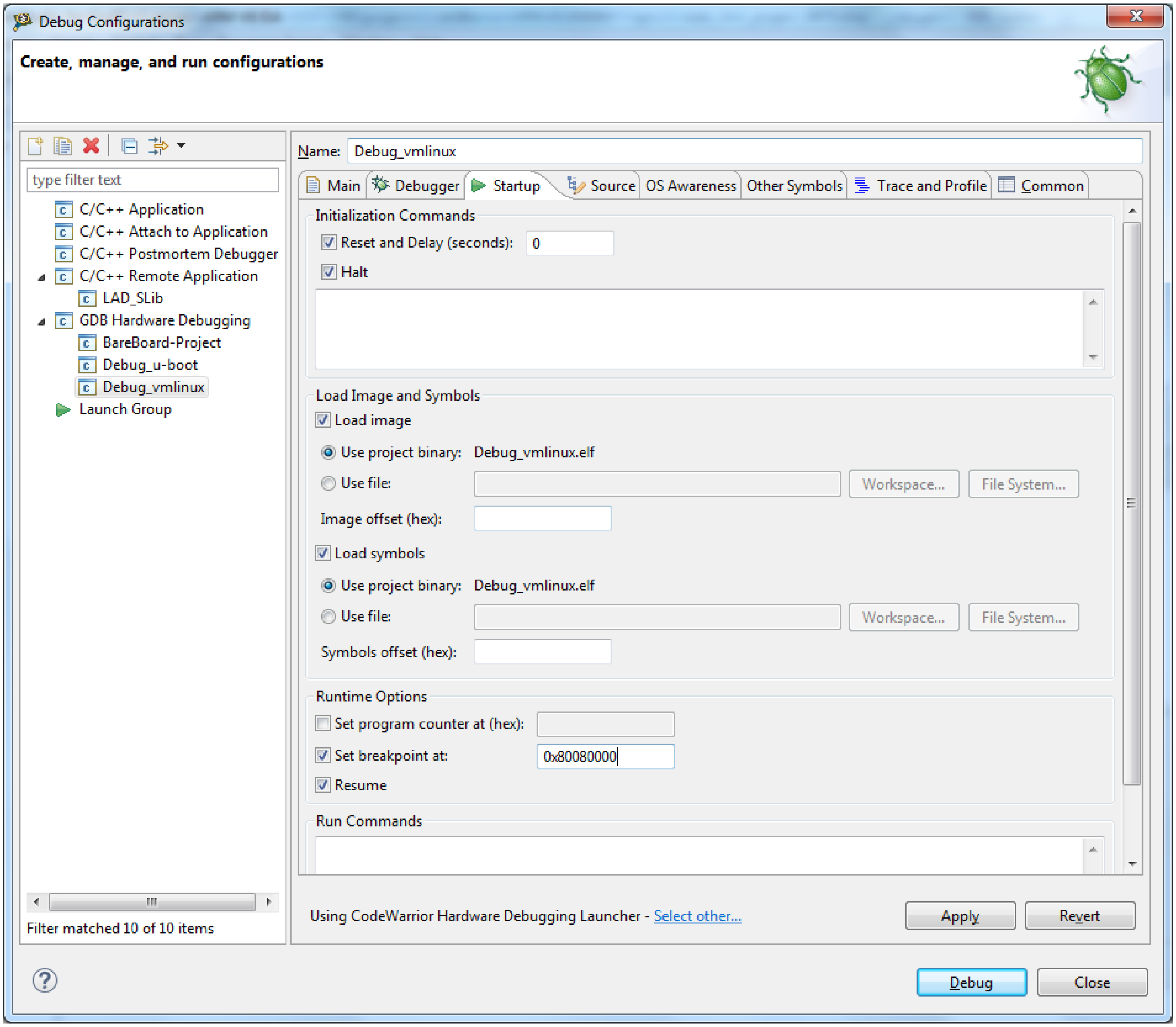
To create an ARMv8 bare metal project for U-Boot debug, perform these steps:

1. Open CodeWarrior for ARMv8.
2. Import a Linux Kernel image as described in CodeWarrior ELF Importer wizard.
3. Select **Run > Debug Configurations** to open the Debug Configurations dialog.
4. Click the **Startup** tab.
   a. Set breakpoint at: 0x80080000.
   b. Check the **Resume** button.

NOTE

Step (b) should be done only if nothing is running yet on the target board, or in case you have just started the target board but have not started the Linux Kernel.

However, in case you simply attach it to a running the Linux Kernel session the above step should be skipped. PC will reflect the current PC while the Linux Kernel is running.



Debug Configurations

Create, manage, and run configurations

Name: Debug_vmlinux

Initialization Commands

☑ Reset and Delay (seconds): 0

☑ Halt

Load Image and Symbols

☑ Load image

● Use project binary: Debug_vmlinux.elf

○ Use file:

Image offset (hex):

☑ Load symbols

● Use project binary: Debug_vmlinux.elf

○ Use file:

Symbols offset (hex):

Runtime Options

☐ Set program counter at (hex):

☑ Set breakpoint at: 0x80080000

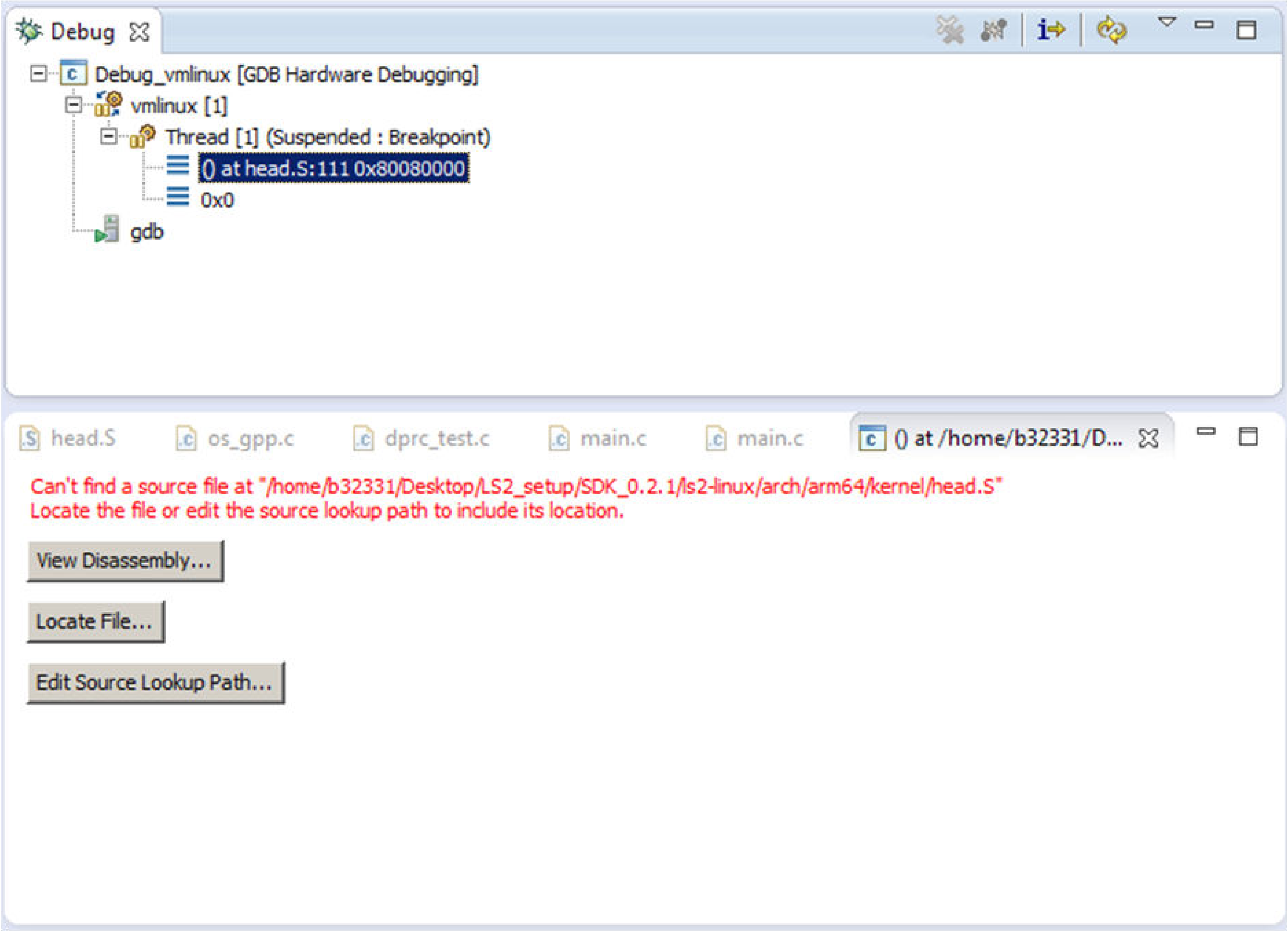
☑ Resume

Run Commands

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 8-15. Startup tab

5. Set up the target connection configuration, as explained in Configuring Target.
6. Click the **Debug** button to initiate the debug session. The debugger should stop at 0x80080000 – kernel entry point address.


Debug
Debug_vmlinux [GDB Hardware Debugging]
vmlinux [1]
Thread [1] (Suspended : Breakpoint)
() at head.S:111 0x80080000
0x0
gdb
head.S
os_gpp.c
dprc_test.c
main.c
main.c
() at /home/b32331/D...
View Disassembly...
Locate File...
Edit Source Lookup Path...

Can't find a source file at "/home/b32331/Desktop/LS2_setup/SDK_0.2.1/ls2-linux/arch/arm64/kernel/head.S"
Locate the file or edit the source lookup path to include its location.

Figure 8-16. Debug Session Window

8.3.3 Linux Kernel debug support

This section explains the steps required to perform Linux kernel debug in CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.

This section includes:

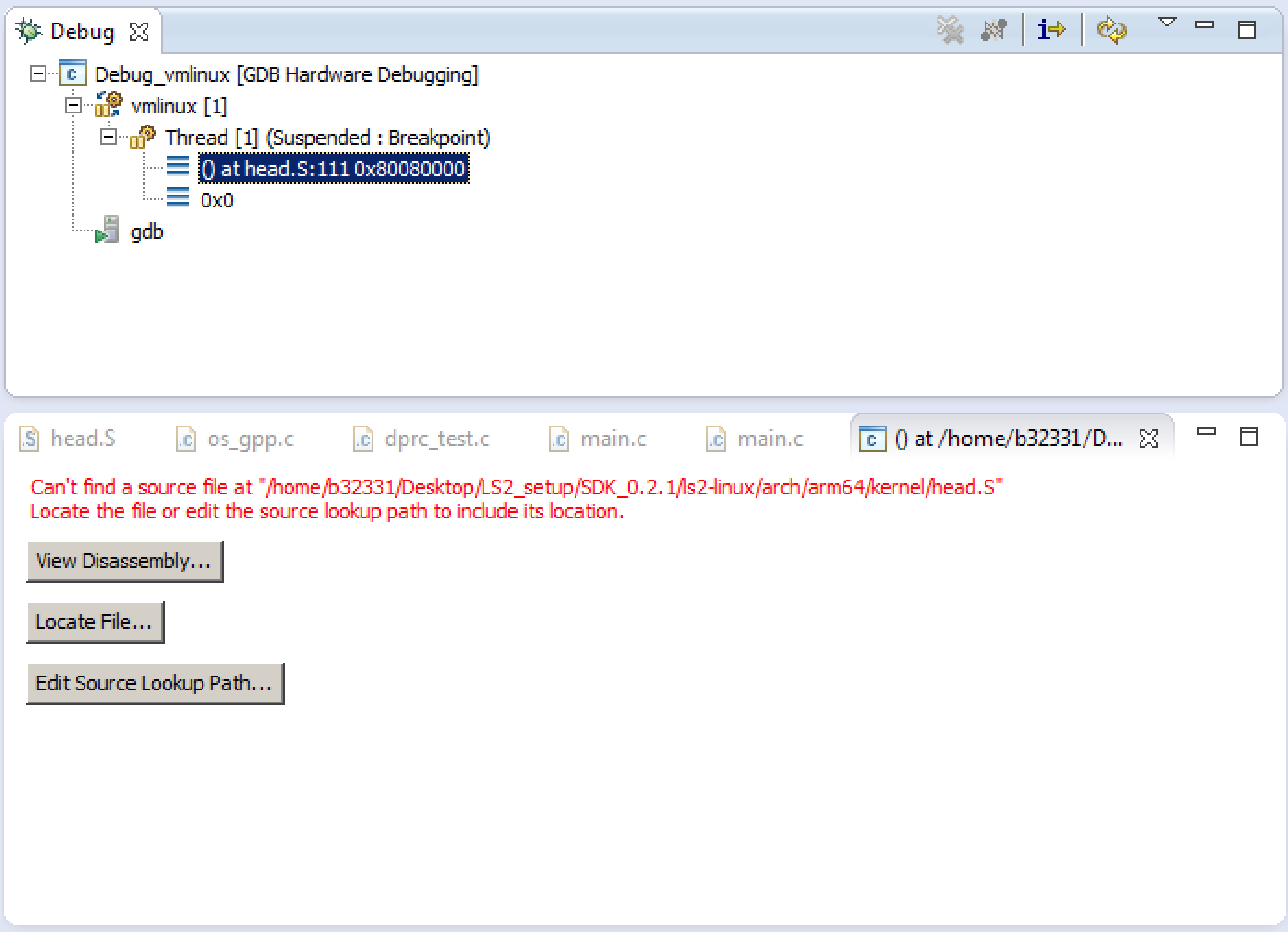
- Setting the source path mapping
- Debug and Kernel Awareness capabilities

8.3.3.1 Setting the source path mapping

This section explains the steps required to load symbols and set source path mapping.

Perform the following steps:

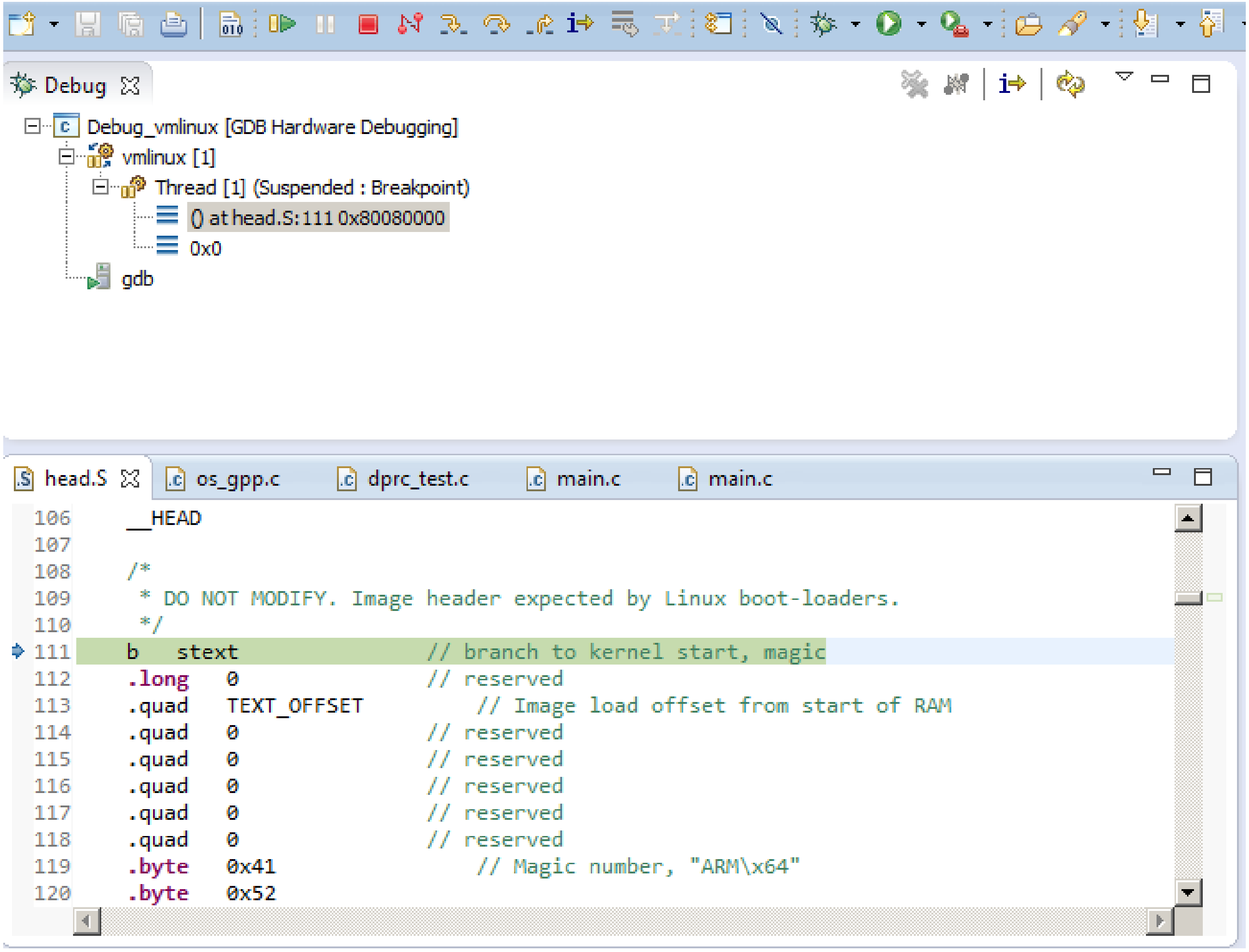
1. Click the **Refresh Debug Views** button to refresh the debug views updated with the new stack and the registers view.
2. Close the **Source not found** window.
3. Double- click the stack for triggering the source-level mapping request.
4. Locate the file suggested by the debugger.


Debug
Debug_vmlinux [GDB Hardware Debugging]
vmlinux [1]
Thread [1] (Suspended : Breakpoint)
() at head.S:111 0x80080000
0x0
gdb
head.S
os_gpp.c
dprc_test.c
main.c
main.c
() at /home/b32331/D...
View Disassembly...
Locate File...
Edit Source Lookup Path...

Can't find a source file at "/home/b32331/Desktop/LS2_setup/SDK_0.2.1/ls2-linux/arch/arm64/kernel/head.S"
Locate the file or edit the source lookup path to include its location.

Figure 8-17. Locate source window

The following figure shows stack and the source views added.



```
Debug_vmlinux [GDB Hardware Debugging]
    vmlinux [1]
        Thread [1] (Suspended : Breakpoint)
            () at head.S:111 0x80080000
            0x0
    gdb
```

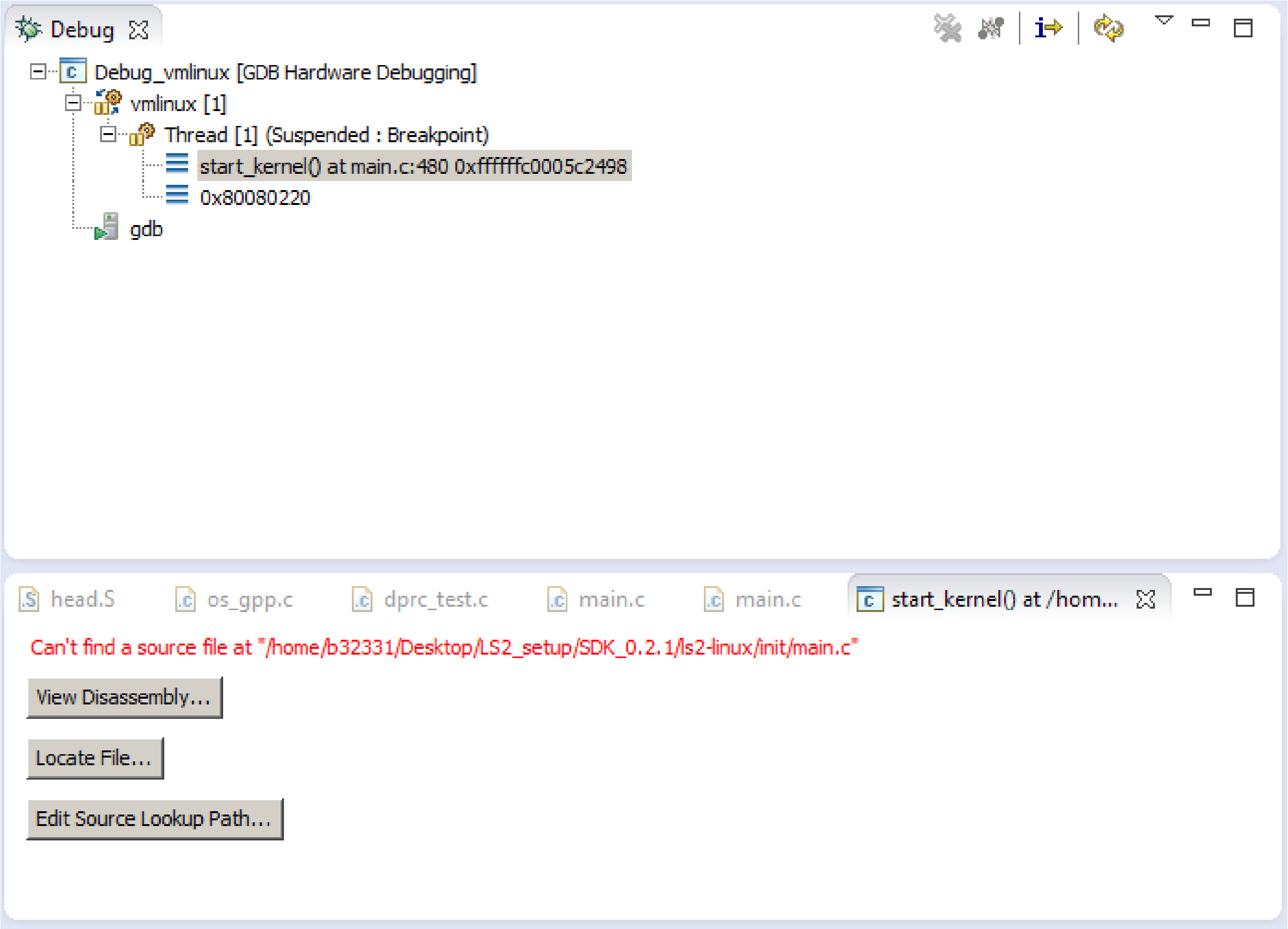
```
	__HEAD

	/*
	 * DO NOT MODIFY. Image header expected by Linux boot-loaders.
	 */
	b	stext				// branch to kernel start, magic
	.long	0			// reserved
	.quad	TEXT_OFFSET			// Image load offset from start of RAM
	.quad	0			// reserved
	.quad	0			// reserved
	.quad	0			// reserved
	.quad	0			// reserved
	.quad	0			// reserved
	.byte	0x41				// Magic number, "ARM\x64"
	.byte	0x52
```

Figure 8-18. Stack and sources added

NOTE

You can add a static map entry using the Edit Source Lookup Path button to avoid locating file using the Locate File button, whenever a new file is requested

5. To go ahead with next important step in Linux kernel debug (start_kernel), you need to set up a breakpoint there using this command: *break start_kernel* in the same gdb console.
6. Click the **Resume** button. Alternatively, press the F8 key. The breakpoint will be hit.
7. Click the **Refresh Debug Views** button to refresh the debug views updated with the new stack and the registers view.
8. Close the **Source not found** window.
9. Double-click the stack for triggering the source-level mapping request.
10. Locate the file suggested by the debugger.



```
Debug_vmlinux [GDB Hardware Debugging]
  vmlinux [1]
    Thread [1] (Suspended : Breakpoint)
      start_kernel() at main.c:480 0xffffffc0005c2498
      0x80080220
  gdb
```

Can't find a source file at "/home/b32331/Desktop/LS2_setup/SDK_0.2.1/ls2-linux/init/main.c"

Figure 8-19. Debug console

11. For details about debug and kernel awareness capabilities, see Debug and Kernel Awareness capabilities.
12. Click the **Resume** button to run the vmlinux. Alternatively, press the F8 key.

NOTE

If everything is setup correctly, clicking the **Resume** button (F8) will show the next linux log in the tio_console from the Linux machine.

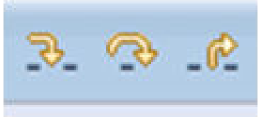
13. To start the Linux Kernel debug again, close/terminate the actual connection. If your target is the simulator, stop the simulator consoles, restart the simulator consoles, and debug again.

8.3.3.2 Debug and Kernel Awareness capabilities

This section explains various Debug and Kernel Awareness capabilities.

Perform the following steps:

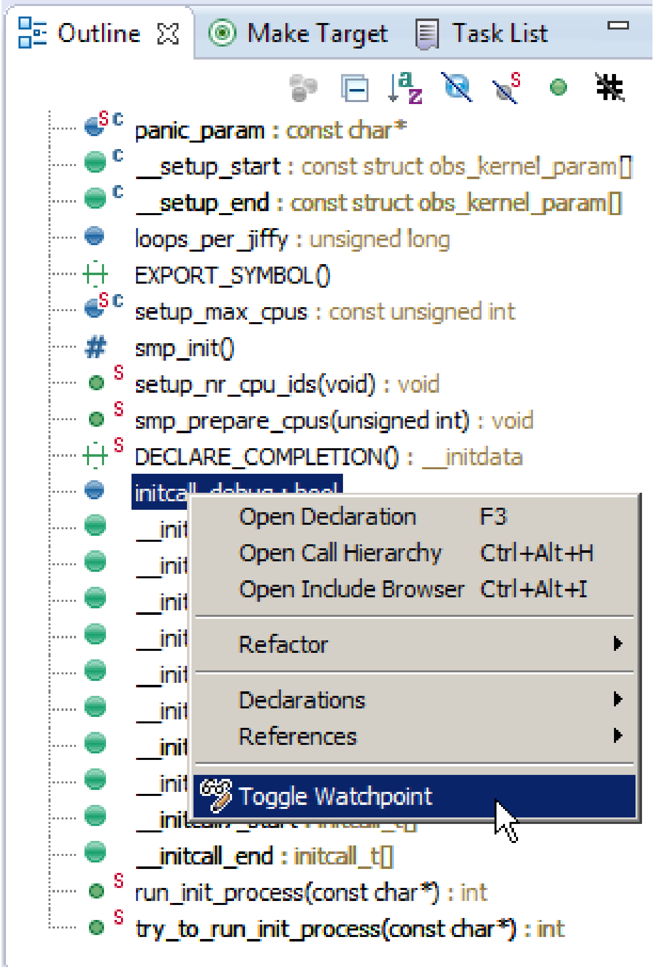
1. Select **Window > Show view > Disassembly** to enable the Disassembly view.
2. Double-click a line to inspect breakpoints. You can inspect them using:
   - Breakpoints view
   - `info breakpoints` command from GDB shell
3. Set up hardware breakpoints using `hbreak` command from GDB console.
4. You can also perform step in, step over, and step return finctions from the GUI.



5. Add watchpoints (data breakpoints) using the **Toogle Watchpoint** option from the context menu.

NOTE

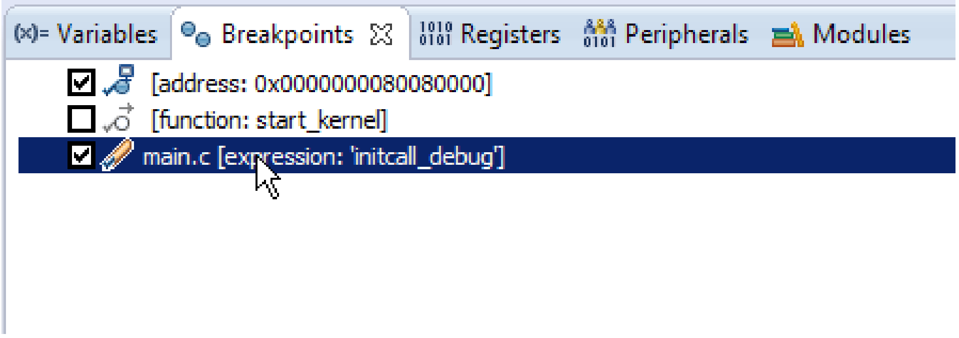
A watchpoint only makes sense for a global variable (or to a global memory address).



```
panic_param : const char*
__setup_start : const struct obs_kernel_param[]
__setup_end : const struct obs_kernel_param[]
loops_per_jiffy : unsigned long
EXPORT_SYMBOL()
setup_max_cpus : const unsigned int
smp_init()
setup_nr_cpu_ids(void) : void
smp_prepare_cpus(unsigned int) : void
DECLARE_COMPLETION() : __initdata
initcall_debug : bool
__init
__init
__init
__init
__init
__init
__init
__init
__init
__initcall_end : initcall_t[]
run_init_process(const char*) : int
try_to_run_init_process(const char*) : int
```

Figure 8-20. Toggle Watchpoint option

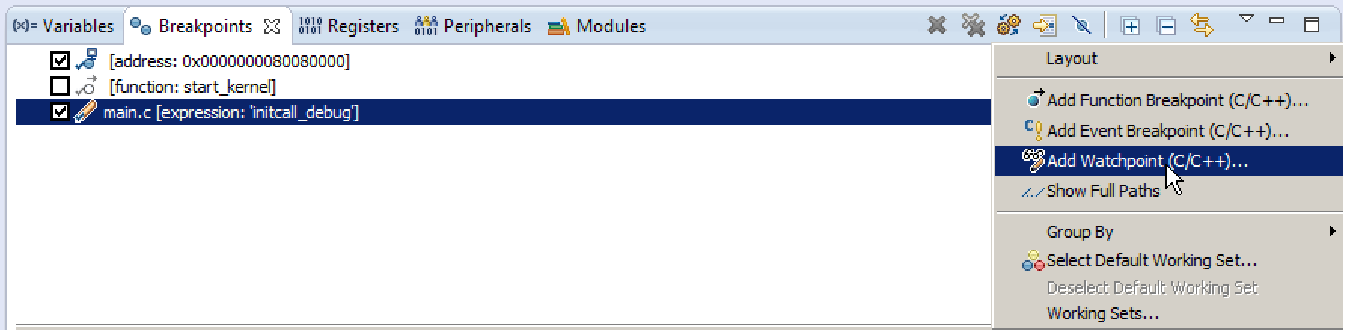
The watchpoint is then listed in the Breakpoints view.



☑ [address: 0x0000000080080000]
□ [function: start_kernel]
☑ main.c [expression: 'initcall_debug']

Figure 8-21. Breakpoints view

You can also add watchpoints using the drop-down menu in the Breakpoint view.


(x)= Variables
Breakpoints
Registers
Peripherals
Modules
Layout
Add Function Breakpoint (C/C++)...
Add Event Breakpoint (C/C++)...
Add Watchpoint (C/C++)...
Show Full Paths
Group By
Select Default Working Set...
Deselect Default Working Set
Working Sets...

☑ [address: 0x0000000080080000]
□ [function: start_kernel]
☑ main.c [expression: 'initcall_debug']

Use CodeWarrior software to see some important information about the linux kernel, for example general information, build time, modules list, threads list and so on. To see the full Kernel Awareness capabilities, refer Linux kernel awareness.

8.3.4 Module debugging

This topic explains:

- **Module debugging use cases**
- Module debugging from Eclipse GUI

8.3.4.1 Module debugging use cases

1. Loading and unloading module’s symbols file

The runtime address when the kernel relocates the kernel module it is known only at runtime after the module is loaded while the module's symbols file contains only the compile time address information. Therefore module symbols file can be loaded only when the module is loaded into the kernel (e.g. using insmod or modprobe command).

Two symbols files are generated after a module compilation: <module_name>.ko and <module_name>.o. The .ko file should be copied to the target and loaded using insmod/modprobe Linux command. The .o file it is the symbols file to be loaded into the debugger.

The kernel module's symbols file can be loaded in two ways:

a. Manually, using the ka-module-load command. The command should be executed after the module is loaded. The typical use cases are:
   - Configure debugger to suspend when the module insert is detected. When the debugger suspend, load the corresponding module's symbols file.
   - The module being inserted, suspend the target execution and load the corresponding symbols file.

b. Automatically (ka-module-config-auto-load=True.), using ka-module-config-map-loadWhen automatic loading mode is enabled, the debugger detects when a module is inserted (insmod or modprobe) and automatically searches the configured symbols file mapping and loads the symbols file. Before inserting the module, the user should add the corresponding symbols file into the symbols file mapping using ka-module-config-map-load command. This command can be run at any time (before and after module loading), but the symbols file is loaded only when the debugger detects that the corresponding module has been inserted.

The user can unload the module's symbol file if the symbols file is already loaded. When the module is removed (rmmod), the debugger automatically unloads the module symbols file, independent of the value of ka-module-config-auto-load. This is done because the module relocation addresses are not valid anymore and even on a new module insertion there will be different relocation addresses.

2. Setting breakpoints in module
   Breakpoints in module's source code or at a specific module function can be set at any time, even the module symbols file is not loaded into the debugger.If the module's symbols file is loaded, the breakpoint is set/enabled and the module relocation address is displayed in the breakpoint properties.

```
(gdb)  break krng_mod_initBreakpoint 3
        at 0xffffffbffc03a000: file crypto/krng.c, line 50.(gdb) info
        breakpoints Num    Type            Disp Enb Address            What3
breakpoint     keep y
      0xffffffbffc03a000 in krng_mod_init at crypto/krng.c:50
```

If the module's symbols file is not loaded, the debugger could not resolve the corresponding breakpoint relocation address, but will set the breakpoint as "pending". When the module is inserted and the module's symbols file is loaded, the debugger will refresh the "pending" breakpoints resolving the relocation address.

The debugger behavior for "pending" breakpoint is configurable using "set breakpoint pending" command with the following values:

- "auto": this is the default value. When the breakpoint is set from command line interface, the debugger asks the user to select one of the following values. From Eclipse/gdb-MI, the "auto" value will make the breakpoint pending "on"
- "on" breakpoint "pending" is enabled
- "off" breakpoint "pending" is disabled. With this setting, the breakpoint can not be set when the module's symbols file is not loaded

3. Debug Linux kernel module from the module_init function

   There are several ways of doing kernel module debug from the module_init function:

   a. Without suspend at module insertion
      - Add the symbols file to the configured map using the command ka-module-config-map-load.
      - Enable module auto-load
      - Set a breakpoint to the module's init function. The breakpoint will be "pending", as the module is not loaded yet.
      - Insert the module (insmod). The debugger will stop at the module's init function

   b. With suspend at module insertion
      - Enable suspend at module insertion
      - Insert the module. The debugger will suspend the target
      - Load the symbols file using ka-module-load command
      - Set a breakpoint to the module's init function. The breakpoint will be resolved as the module and the symbols file are loaded
      - Run. The debugger will stop at the module's init function

4. Module insertion and removal detection

   Module insertion and removal detection is implemented by setting a special breakpoint (named eventpoint) in Linux Kernel code (not module code).

   - When the module is prepared to be executed, but before running the module's init function
   - And when the module is prepared to be remove, after running the module's delete function

   These debugger specific breakpoints are not visible to the user. The command " info breakpoints " displays no information about these breakpoints.

   The eventpoints information can be displayed using the command " maintenance info breakpoints ":

```
(gdb)
        maintenance info breakpoints Num      Type            Disp Enb
Address             What-1      breakpoint      keep y    0xffffffc0000ef8fc in
        load_module at kernel/module.c:3020 inf 1-2      breakpoint      keep y
0xffffffc0000eddd4 in
        free_module at kernel/module.c:1840 inf 1(gdb)
```

The eventpoints have negative breakpoint numbers and the user can not modify the breakpoint properties (e.g. delete breakpoint).

8.3.4.2 Module debugging from Eclipse GUI

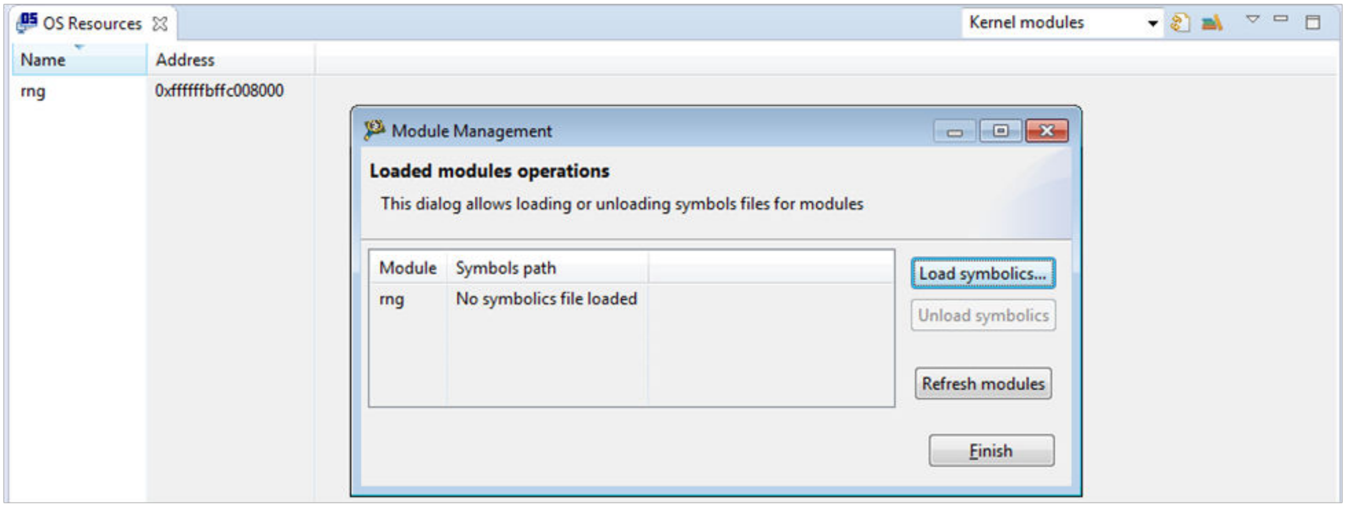
Before launching the module debugging session, set the following options in the **OS Awareness** tab:

- Check **Suspend target when module insert or removal is detected**.
- Check **Automatically load configured symbolic files at module init detection**.

If the option, **Automatically load configured symbolic files at module init detection** is enabled, the debugger loads the user defined list of module symbolics files, used to configure the gdb, in the *Auto-load module symbolic files list* section . The module symbolics file name signifies the module name, for example the symbolics file *rng.o* will refer to module *rng*.

To load a different symbolics file, the Module Management dialog, which is available at runtime from OS resources View, should be used.

1. In the *OS resources View* , click .
   The **Module Management** dialog appears with the currently available modules. The dialog will also show if the symbolics files for a module is loaded.



Name	Address
rng	0xffffffbffc008000

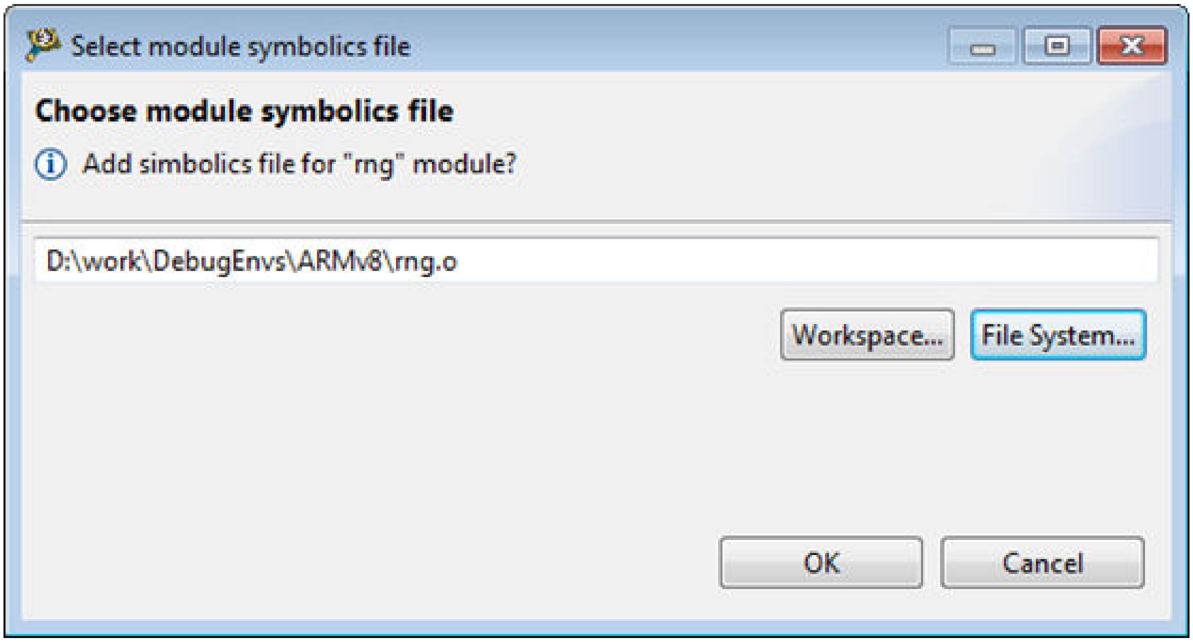
Loaded modules operations

This dialog allows loading or unloading symbols files for modules

Module	Symbols path
rng	No symbolics file loaded

2. Click *Load symbolics* to load a symbolics file for a module.

   The **Select module symbolics file** dialog appears.



Select module symbolics file

Choose module symbolics file

Add simbolics file for "rng" module?

D:\work\DebugEnvs\ARMv8\rng.o

3. The user can choose a different symbolics file for a module if before opening the **Select module symbolics file** dialog the module was selected from the list. In this case, the dialog will ask to confirm the mapping between the current symbolics file and the module.
4. Click **OK**.

Chapter 9
Troubleshooting

This section lists:

- Diagnostic Information Export
- Logging
- I/O support

9.1 Diagnostic Information Export

The Diagnostic Information Wizard feature allows you to export error log information to Freescale support group to diagnose the issue you have encountered while working on the CodeWarrior product.

You can export diagnostic information in the following two ways:

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

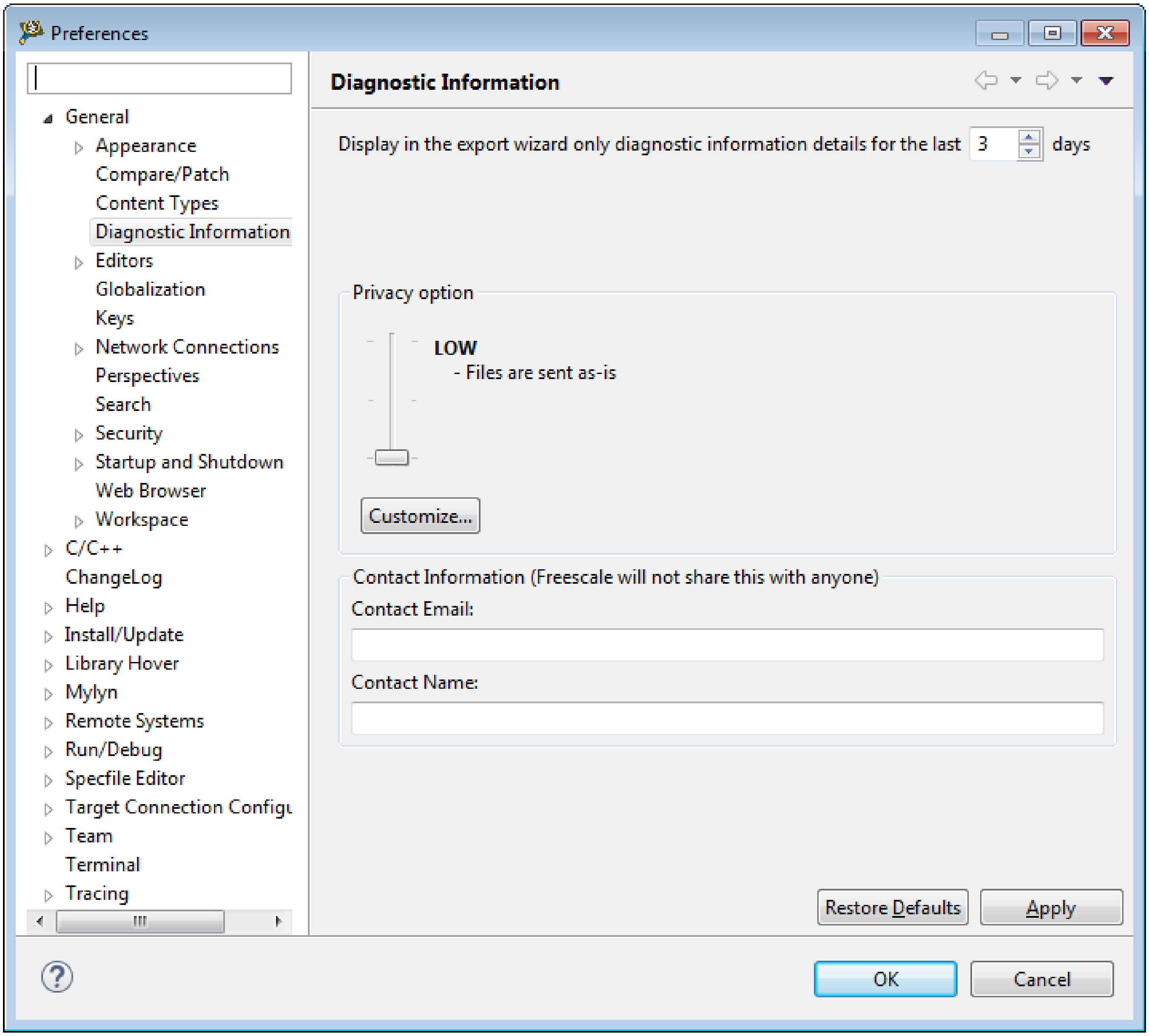
9.1.1 General settings for Diagnostic Information

You can specify general settings for diagnostic information using the **Preferences** dialog.

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

1. Choose **Windows** > **Preferences** from the IDE menu bar.

   The **Preferences** dialog appears.



Diagnostic Information

Display in the export wizard only diagnostic information details for the last 3 days

Privacy option

LOW
- Files are sent as-is

Contact Information (Freescale will not share this with anyone)

Contact Email:

Contact Name:

Figure 9-1. Preferences dialog - Diagnostic Information

2. Expand the **General** group and choose **Diagnostic Information** .

   The **Diagnostic Information** page appears.

3. Enter the number of days for which you want to display the diagnostic information details in the export wizard.
4. Select the **Privacy** option by dragging the bar to low, medium and high.

   Privacy level setting is used to filter the content of the logs.

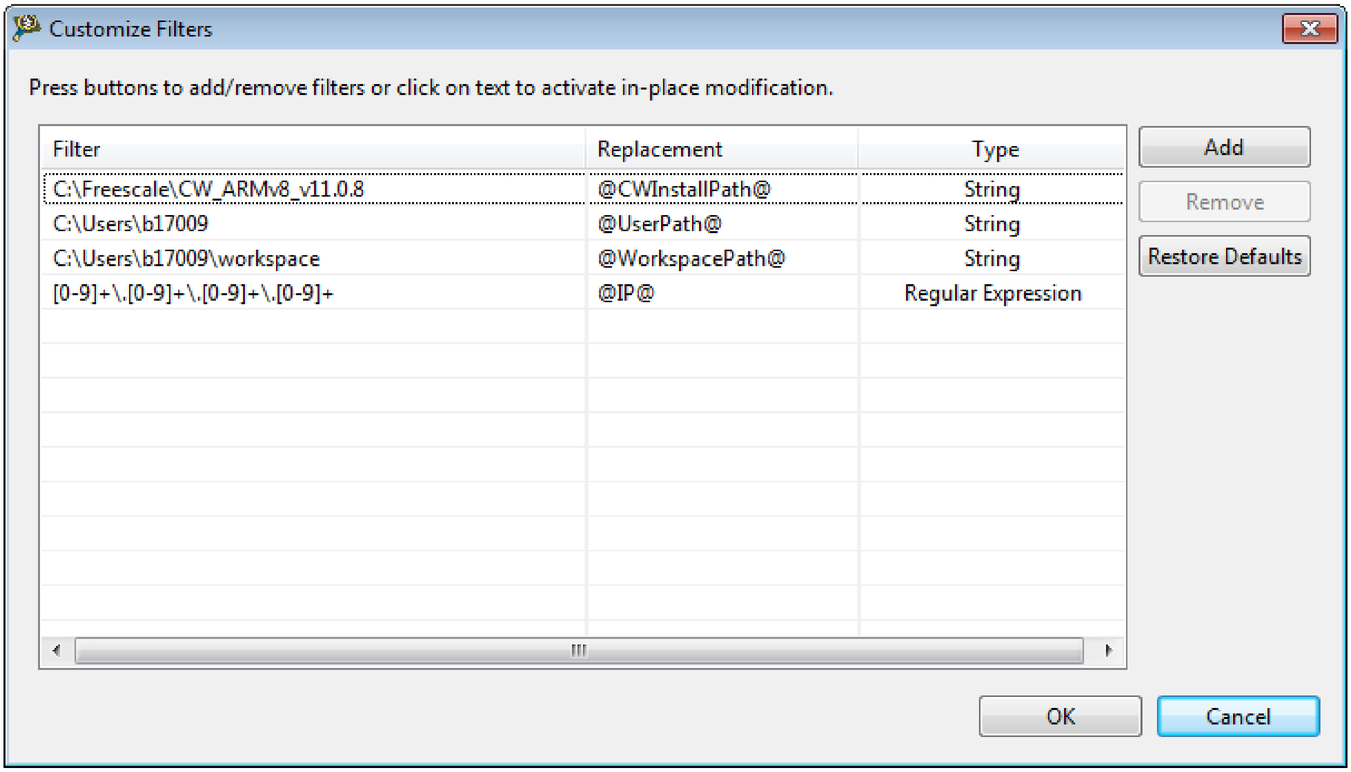
   - Low: The file is sent as is.

- Medium: The personal information is obfuscated. You can click on the customize option to view or modify filter.
- High: The personal information is removed. Filters are used in the rest of the content.

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

   The **Customize Filters** dialog appears. You can add, remove, and modify filters.

6. Click **OK** .



Press buttons to add/remove filters or click on text to activate in-place modification.

Filter	Replacement	Type
C:\Freescale\CW_ARMv8_v11.0.8	@CWInstallPath@	String
C:\Users\b17009	@UserPath@	String
C:\Users\b17009\workspace	@WorkspacePath@	String
[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+	@IP@	Regular Expression

Figure 9-2. Diagnostic Information - Customize Filters

7. Enter **Contact Name** and **Contact Email** in the contact information textbox. This information is optional though Freescale will not share this information with anyone.
8. Click **Restore Defaults** to apply default factory settings.
9. Click **OK** .

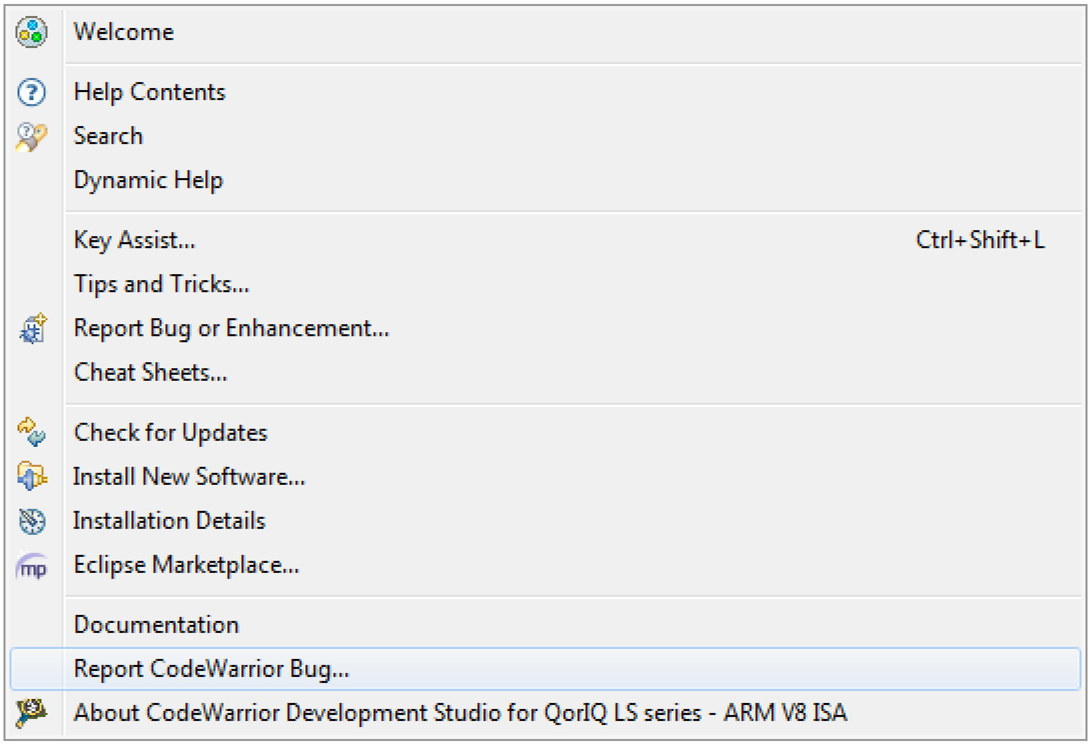
9.1.2 Export Diagnostic Information

You can export diagnostic information into an archive file in workspace.

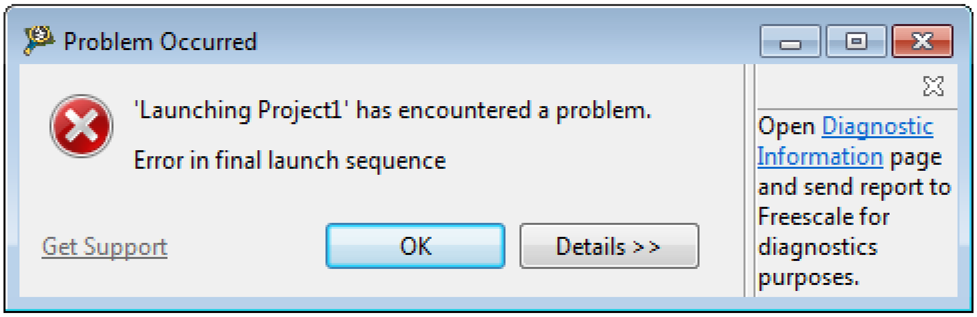
Follow the steps given below to export diagnostic information into an archive.

1. Open **Diagnostic Information** wizard, either by:

- Selecting **Help > Report CodeWarrior Bug**, or



- Through an error reporting dialog such as below. Click the **Diagnostic Information** link in the error dialog.

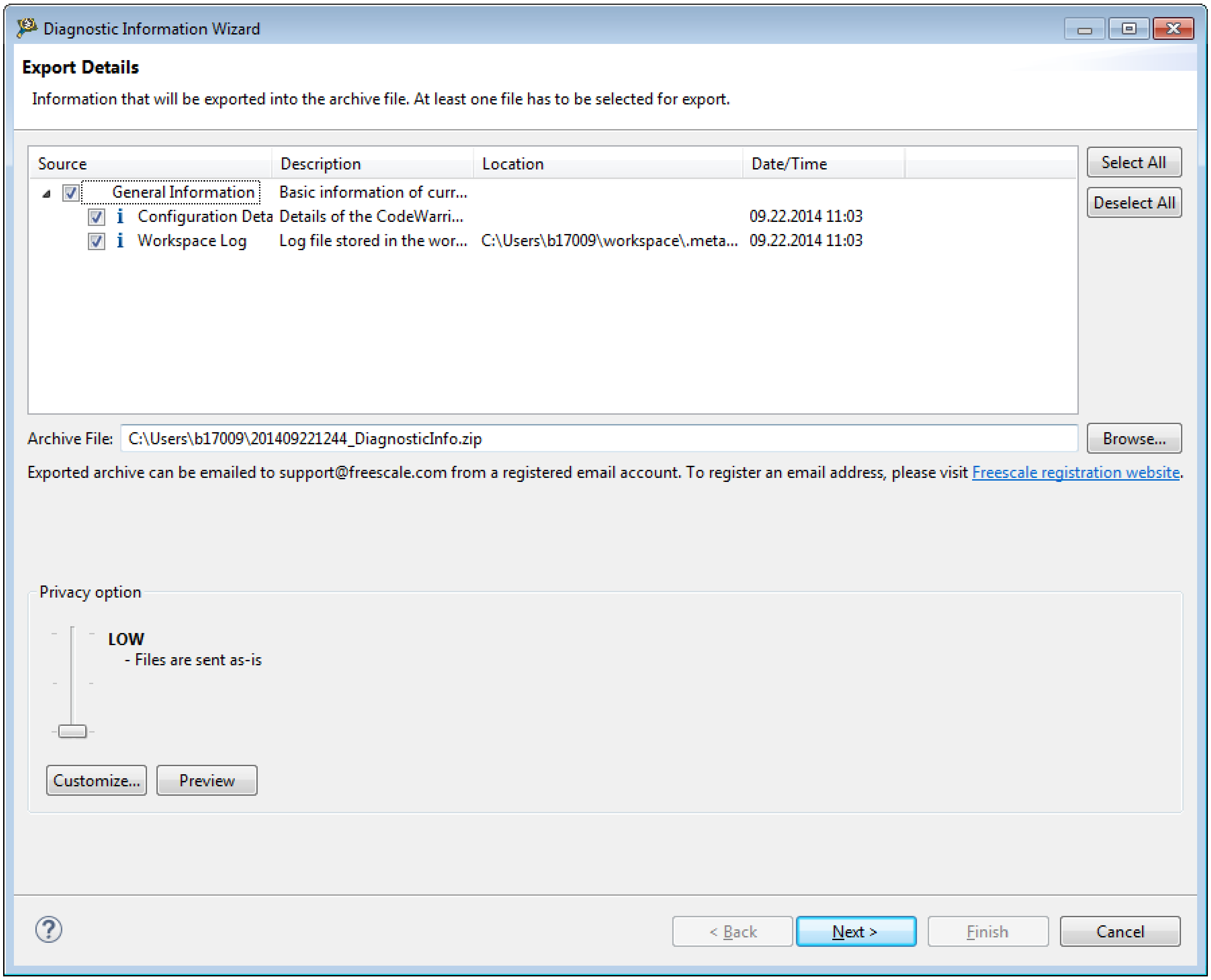


'Launching Project1' has encountered a problem.

Error in final launch sequence

Open Diagnostic Information page and send report to Freescale for diagnostics purposes.

The **Diagnostic Information Wizard** appears.



Export Details

Information that will be exported into the archive file. At least one file has to be selected for export.

Source	Description	Location	Date/Time
☑ General Information	Basic information of curr...		
☑ i Configuration Deta	Details of the CodeWarri...		09.22.2014 11:03
☑ i Workspace Log	Log file stored in the wor...	C:\Users\b17009\workspace\.meta...	09.22.2014 11:03

Archive File: C:\Users\b17009\201409221244_DiagnosticInfo.zip

Exported archive can be emailed to support@freescale.com from a registered email account. To register an email address, please visit Freescale registration website.

Privacy option

LOW
- Files are sent as-is

Figure 9-3. Export - Diagnostic Information Wizard

2. Select the checkbox under the **Source** column to select the information that will be exported into the archive file.

NOTE

You must select at least one file for export.

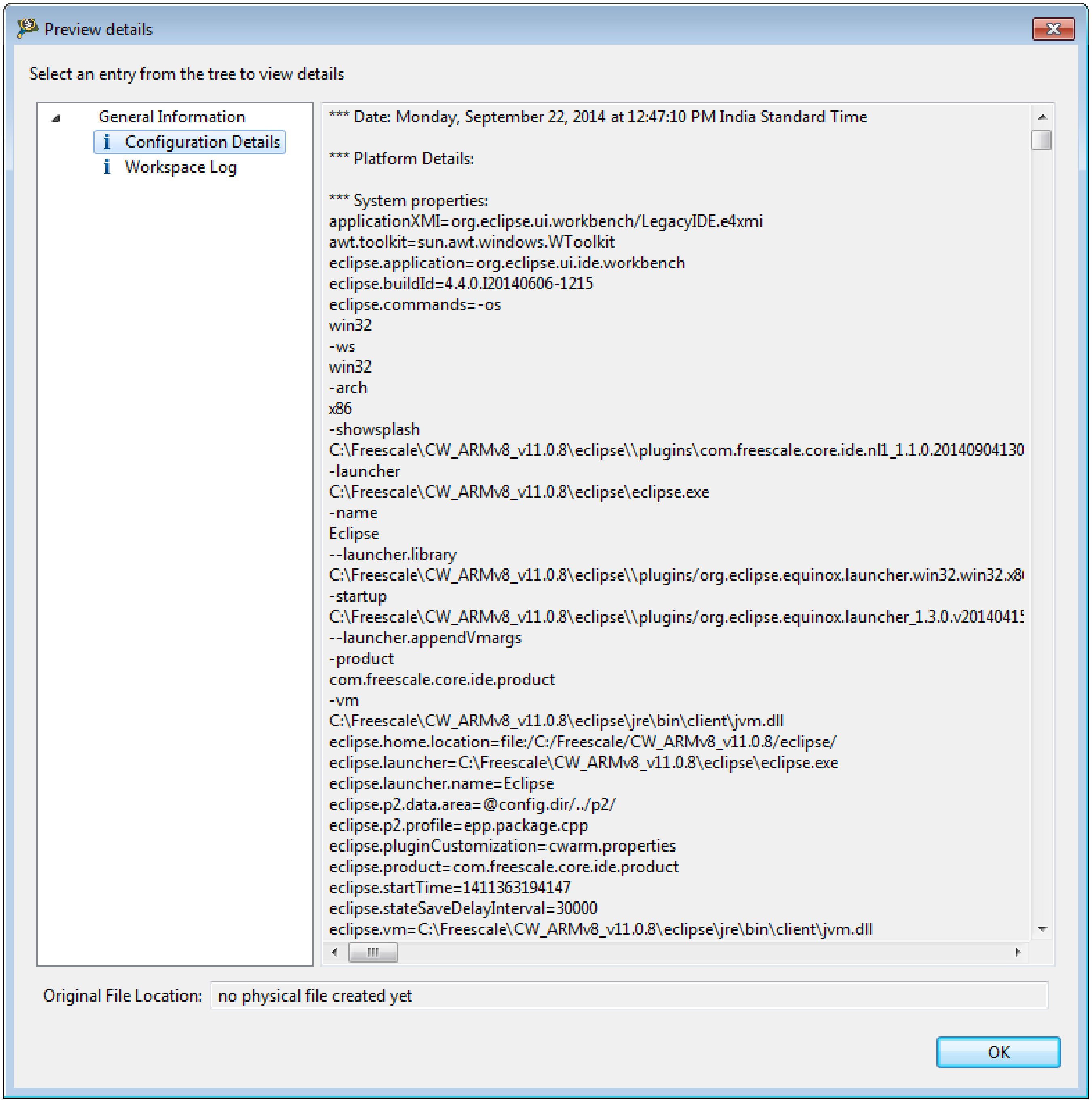
3. Click **Browse** to select a different archive file location.
4. Select the **Privacy option** or click **Customize** to set your privacy level. The **Customize Filters** dialog appears.

NOTE

You can open the **Customize Filters** dialog through **Customize** button in the **Diagnostic Information Export Wizard** (General settings for Diagnostic Information)or in the **Preferences** dialog (General settings for Diagnostic Information).

5. Click **Preview** to view the text that will be sent to Freescale from the wizard.

   The **Preview details** dialog appears.



```
*** Date: Monday, September 22, 2014 at 12:47:10 PM India Standard Time

*** Platform Details:

*** System properties:
applicationXMI=org.eclipse.ui.workbench/LegacyIDE.e4xmi
awt.toolkit=sun.awt.windows.WToolkit
eclipse.application=org.eclipse.ui.ide.workbench
eclipse.buildId=4.4.0.I20140606-1215
eclipse.commands=-os
win32
-ws
win32
-arch
x86
-showsplash
C:\Freescale\CW_ARMv8_v11.0.8\eclipse\\plugins\com.freescale.core.ide.nl1_1.1.0.20140904130
-launcher
C:\Freescale\CW_ARMv8_v11.0.8\eclipse\eclipse.exe
-name
Eclipse
--launcher.library
C:\Freescale\CW_ARMv8_v11.0.8\eclipse\\plugins/org.eclipse.equinox.launcher.win32.win32.x8
-startup
C:\Freescale\CW_ARMv8_v11.0.8\eclipse\\plugins/org.eclipse.equinox.launcher_1.3.0.v2014041
--launcher.appendVmargs
-product
com.freescale.core.ide.product
-vm
C:\Freescale\CW_ARMv8_v11.0.8\eclipse\jre\bin\client\jvm.dll
eclipse.home.location=file:/C:/Freescale/CW_ARMv8_v11.0.8/eclipse/
eclipse.launcher=C:\Freescale\CW_ARMv8_v11.0.8\eclipse\eclipse.exe
eclipse.launcher.name=Eclipse
eclipse.p2.data.area=@config.dir/../p2/
eclipse.p2.profile=epp.package.cpp
eclipse.pluginCustomization=cwarm.properties
eclipse.product=com.freescale.core.ide.product
eclipse.startTime=1411363194147
eclipse.stateSaveDelayInterval=30000
eclipse.vm=C:\Freescale\CW_ARMv8_v11.0.8\eclipse\jre\bin\client\jvm.dll
```

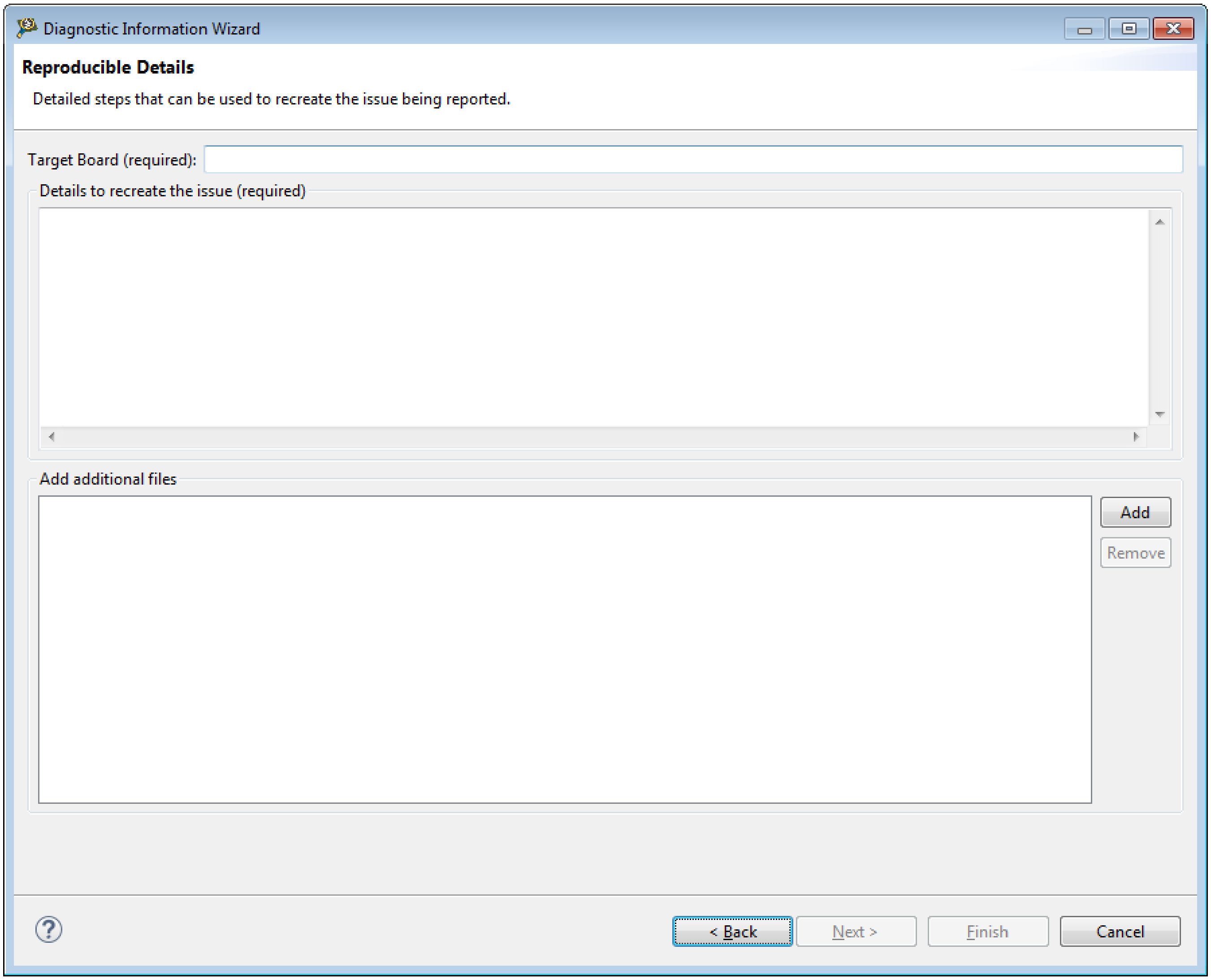
Original File Location: no physical file created yet

Figure 9-4. Preview details dialog

You can also check if more filters are needed to protect any sensitive information from leakage.

6. Click **OK** .
7. Click **Next** in the **Diagnostic Information Export Wizard**.

   The **Reproducible Details** page appears.



Diagnostic Information Wizard

Reproducible Details

Detailed steps that can be used to recreate the issue being reported.

Figure 9-5. Reproducible Details page

8. Enter the reproducible steps and any other relevant information in the **Details to recreate the issue** textbox.
9. Click **Add** to add additional files to the archive file for diagnosis.
10. Click **Finish**.

9.2 Prevent core from entering non-recoverable state due to unmapped memory access

The ARM core can enter in a non-recoverable state when a speculative access to an unmapped memory happens.

Also this can happen for accesses to memory regions that are marked as valid in the MMU, but the undelying memory interface is either misconfigured or absent. For example, access to a memory range dedicated to PCIe without a proper initialization for the PCIe controller or access to memory holes as defined in the SoC memory map can cause core to enter in a non-recoverable state.

If the debugger detects a failed attempt to stop the core in such situation, it samples the value of the External PC Debug register (EDPCSR) in order to provide the program location where the program has hanged. An error message is displayed informing the user that the stop attempt has failed and listing the collected PC sample value.

Although the debug session is not reliable from this point onwards and must be terminated, the PC value allows the user to identify and fix the application problem that has caused the core to enter into the non-recoverable state. The user needs to make sure that the MMU is configured from the application in such a way that all valid translations point to the actual memory.

9.3 Logging

GDB logs are used to save output of the GDB commands to a file.There are two types of logs: GDB and GDB RSP server.

- GDB logs - Configured with standard GDB log control commands.

  For details about GDB log control commands, refer https://sourceware.org/gdb/onlinedocs/gdb/Logging-Output.html

- GDB RSP server log - Configured with GDB monitor commands. For details about GDB monitor commands, run the command `monitor help log`.

  The log messages from the GDB RSP server are grouped in different categories, and each category can be associated with one or more log destinations, such as console, file, and socket.

9.4 Recording

GDB provides the possibility to record all commands typed during a command-line debug session and save these to a file.

To enable this feature from command line GDB:

- (gdb) set history size unlimited – command history size defaults to 256; “unlimited” recommended
- (gdb) set history filename <filename> - the file where to save the recording (default: “.gdb_history”, located in the GDB executable home directory)
- (gdb) set history save on – all following commands will be recorded;

NOTE

The recorded command history is written to a file only upon exiting GDB.

After ending a debug session and exiting GDB, the “.gdb_history” file can be inspected and eventually edited. Optionally, when restarting the debug session, all commands from the recording may be replayed as a gdb script:

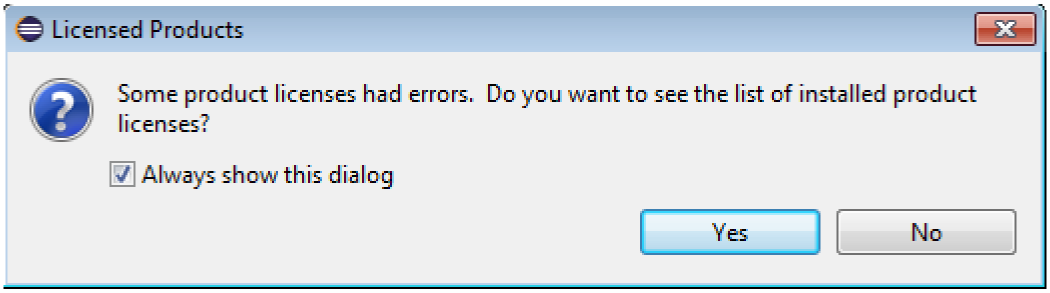
```
(gdb) source .gdb_history
```

9.5 Freescale Licensing

The Freescale Eclipse licensing feature lets the user see and manage the available licenses for the installed Freescale products.

The Freescale Eclipse Licensing feature appears to the user in two different ways:

- A *warning dialog box* appears after each time the CodeWarrior starts if a licensed product is going to expire soon, hasn’t been activated yet, or is disabled because of license expiration.



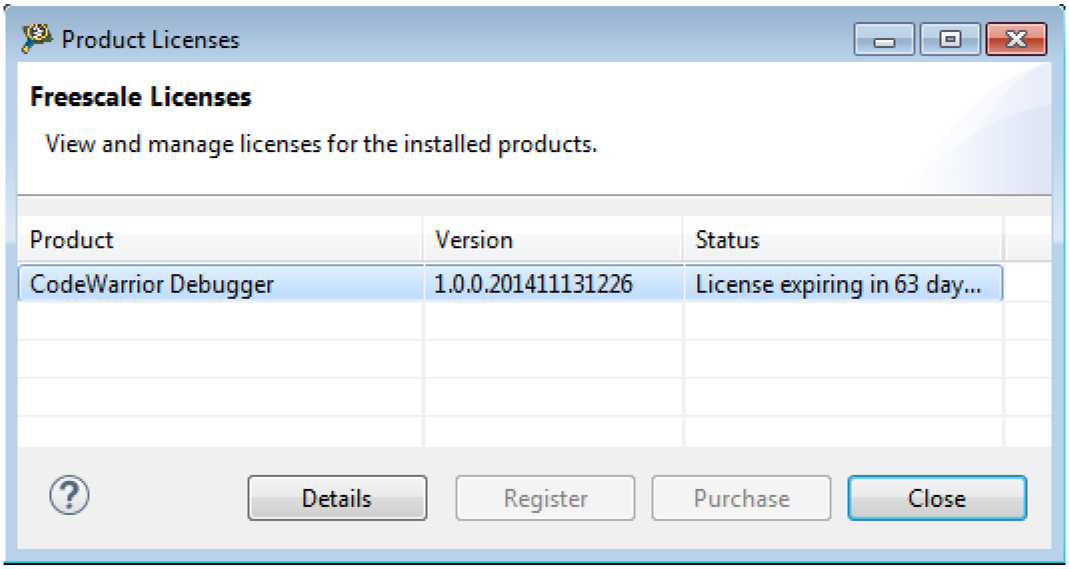
Licensed Products

Some product licenses had errors. Do you want to see the list of installed product licenses?

☑ Always show this dialog

Figure 9-6. Freescale Licensing warning dialog

- The *Freescale Licenses* window displays all installed licensed products and their status (“licensed”, “expiring in X days”, “expired”). It can be opened from **Help > Freescale Licenses**.



Product Licenses

Freescale Licenses

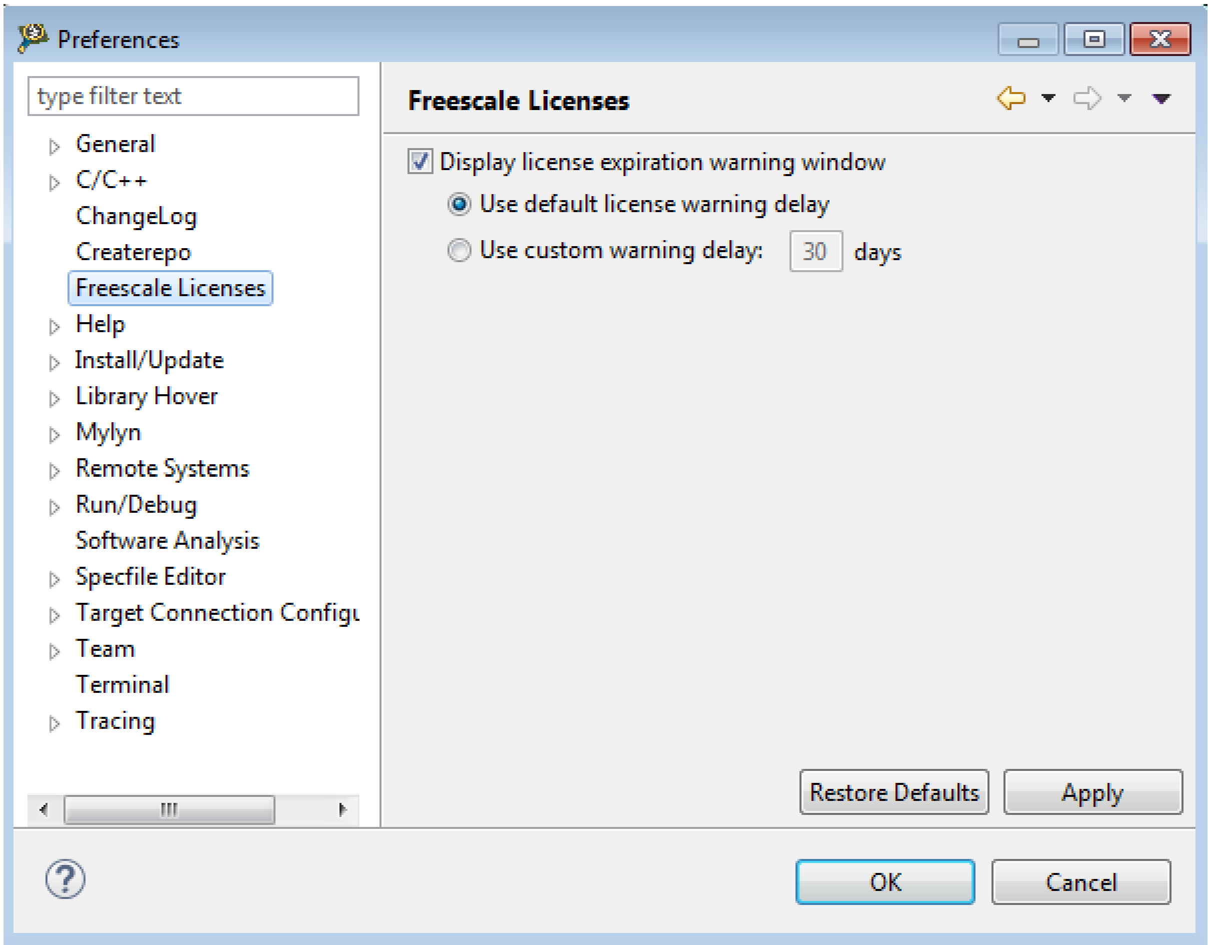
View and manage licenses for the installed products.

Product	Version	Status
CodeWarrior Debugger	1.0.0.201411131226	License expiring in 63 day...

Figure 9-7. Freescale Licenses dialog

There is also a Freescale Licenses preference panel which allows the user to customize specific aspects of the license plugins:

- whether the license expiration warning window should be displayed or not
- after how much delay, the expiration warning window should appear



Freescale Licenses

☑ Display license expiration warning window

● Use default license warning delay

○ Use custom warning delay: 30 days

Figure 9-8. Freescale Licensing preference page

NOTE

The Freescale License plugin is not responsible for enabling or disabling a feature based on its license status, but only to monitor that status, and display it to the user. The plugin itself is responsible to enable or disable itself.

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.