CodeWarrior Development Studio for MPC55xx/MPC56xx Microcontrollers Version 2.xx Targeting Manual

Revised: 9 February 2012

freescale semiconductor

How to Contact Us

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

Table of Contents

1

Introduction

This manual explains how to use CodeWarrior Development Studio for MPC55xx/MPC56xx Microcontrollers to develop software for members of the MPC55xx and MPC56xx families of microcontrollers.

In this chapter:

The MPC55xx/MPC56xx Platform

Members of the MPC55xx/MPC56xx microcontroller families have one or more Power Architecture e200 cores (such as the e200z3 and e200z6 cores). Table 1.1 lists each MPC55xx/MPC56xx microcontroller, along with the particular e200 core (or cores) on this device.

NOTE In addition to an e200 core, many of the microcontrollers listed in Table 1.1 include a co-processor. This CodeWarrior product includes the build tools required to use these co-processors.

Table 1.1 MPC55xx/MPC56xx Microcontrollers and Cores

Microcontroller Family	Description	Device Name	Mask Set	Cores	Flash Memory
MPC5510	Single and dual core Power Architecture microcontrollers for body and gateway applications	MPC5514E		e200z1 and e200z0	512K
		MPC5514G		e200z1 and e200z0	512K
		MPC5515S		e200z1	768K

Table 1.1 MPC55xx/MPC56xx Microcontrollers and Cores

Microcontroller Family	Description	Device Name	Mask Set	Cores	Flash Memory
		MPC5516E		e200z1 and e200z0	1M
		MPC5516G		e200z1 and e200z0	1M
		MPC5516S		e200z1	1M
		MPC5517E		e200z1 and e200z0	1.5M
		MPC5517G		e200z1 and e200z0	1.5M
		MPC5517S		e200z1	1.5M
MPC553x	Ultra low-cost 32bit microcontrollers targeting low-end engine management applications	MPC5533		e200z3	768K
		MPC5534		e200z3	1M
MPC555x	High performance microcontrollers for engine management	MPC5553		e200z6	1,5M
		MPC5554		e200z6	2M
MPC556x	Microcontrollers for advanced driver assistance engine management, general body	MPC5561		e200z6	1M
		MPC5565		e200z6	2M
		MPC5566		e200z6	3M
		MPC5667		e200z6	2M

Table 1.1 MPC55xx/MPC56xx Microcontrollers and Cores

Microcontroller Family	Description	Device Name	Mask Set	Cores	Flash Memory
MPC56xxB	32-bit MCU with CAN, LIN and other peripherals for a range of automotive body applications	MPC5602B	*M07N	e200z0h	256K
		MPC5602B	*M27V	e200z0h	256K
		MPC5602C	*M07N	e200z0h	256K
		MPC5602C	*M27V	e200z0h	256K
		MPC5603B	*M07N	e200z0h	384K
		MPC5603B	*M27V	e200z0h	384K
		MPC5603C	*M07N	e200z0h	384K
		MPC5603C	*M27V	e200z0h	384K
		MPC5604B	*M07N	e200z0h	512K
		MPC5604B	*M27V	e200z0h	512K
		MPC5604C	*M07N	e200z0h	512K
		MPC5604C	*M27V	e200z0h	512K
		MPC5605B		e200z0h	768K
		MPC5606B		e200z0h	1M
		MPC5607B		e200z0h	1.5M
		MPC5644B		e200z4	1.5M
		MPC5645B		e200z4	2M
		MPC5646B		e200z4	3M
		MPC5644C		e200z4 and e200z0h	1.5
		MPC5645C		e200z4 and e200z0h	2M
		MPC5646C		e200z4 and e200z0h	3M

Table 1.1 MPC55xx/MPC56xx Microcontrollers and Cores

Microcontroller Family	Description	Device Name	Mask Set	Cores	Flash Memory
MPC56xxP	32-bit MCU for chassis and safety applications	MPC5601P	*M07N	e200z0h	192K
		MPC5601P	*M26V	e200z0h	192K
		MPC5602P	*M07N	e200z0h	256K
		MPC5602P	*M26V	e200z0h	256K
		MPC5603P	*M07N	e200z0h	384K
		MPC5603P	*M26V	e200z0h	384K
		MPC5604P	*M07N	e200z0h	512K
		MPC5604P	*M26V	e200z0h	512K
MPC56xxS	32 bit MCUs for next-generation dashboards, with TFT-drive. Cost-effective for entry-level cluster applications	MPC5606S	*M07N	e200z0h	1M
		MPC5606S	FS60X2	e200z0h	1M
		MPC5645S		e200z4	2M
MPC56xxM	32 bit MCU for entry level powertrain with on-chip emission control.	MPC5633M		e200z335	1M
		MPC5634M		e200z335	1.5M
MPC56xxL	32-bit system-on-chip devices intended for Electric Power Steering and those applications requiring a high Safety Integrity Level (SIL).	MPC5643L		e200z4 (2x)	1MB
MPC56xxA	32-bit system-on-chip devices intended for use in mid-range engine control and automotive transmission control applications.	MPC5644A		e200z4	4MB

Table 1.1 MPC55xx/MPC56xx Microcontrollers and Cores

Microcontroller Family	Description	Device Name	Mask Set	Cores	Flash Memory
MPC5668E/G	Dual core 32-bit MCUs for Gateway Applications	MPC5668E		e200z6 and e200z0	
		MPC5668E		e200z6 and e200z0	
MPC56xxF	32-bit Power Architecture MCU for green powertrain applications	MPC5674F	MVx264 MVxA264	e200z7	4M
MPC56xxK	32-bit embedded controller designed for advanced driver assistance systems, motor control, and applications that require a high safety integrity level	MPC5675K		e200z7 (2x)	2M
MPC56xxR	32-bit dual core, dual issue Power Architecture microcontroller platform for chassis and safety applications including braking, steering, domain control, and entry level radar.	MPC5676R		e200z7 (2x)	6M
MPC56xxE	32-bit dual core, dual issue Power Architecture microcontroller platform for chassis and safety applications including braking, steering, domain control, and entry level radar.	MPC5604E		e200z0h	512K

CodeWarrior Build Tools

CodeWarrior MPC55xx/MPC56xx build tools consist of an integrated development environment (IDE), a compiler, an assembler, a linker, and the Main Standard Libraries (MSL). Additionally, you can control separately purchased eTPU build tools or PC-lint from within the CodeWarrior tools.

Integrated Development Environment

If working from the command line, you have to create, maintain, and run makefiles for each project by hand. Alternatively, an IDE provides a graphical user interface (GUI) with which you can use create and manage projects.

With the CodeWarrior IDE, you can perform all aspects of software development; it controls the project manager, the source code editor, the class browser, the compiler, assembler, linker, debugger, and more.

The project manager lets you define the source code files and build settings for a project and automatically updates a project's "internal makefile" as you modify the project.

See the *CodeWarrior IDE User's Guide* for documentation that explains how to use the CodeWarrior IDE.

C/C++ Compiler

The CodeWarrior build tools include an ANSI-compliant C/C++ compiler for MPC55xx/MPC56xx microcontrollers. Used with the CodeWarrior Power Architecture linker, this compiler generates MPC55xx/MPC56xx applications and libraries that conform to the Power Architecture Embedded Application Binary Interface (EABI) standard.

See the *Build Tools Reference* for instructions that explain how to use the CodeWarrior C/C++ compiler.

Assembler

Your CodeWarrior build tools include a standalone assembler for MPC55xx/MPC56xx microcontrollers. This assembler features an easy-to-use syntax.

See the *Assembler Reference* for instructions that explain how to use the CodeWarrior assembler.

Linker

Your CodeWarrior build tools include a linker that generates Executable and Linkable Format (ELF) binaries for MPC55xx/MPC56xx microcontrollers. This linker lets you use

absolute addressing and create multiple user-defined sections. In addition to ELF format, the linker can output S-record format.

Libraries

The CodeWarrior Main Standard Libraries (MSL) and Embedded Warrior Libraries (EWL) are ANSI-compliant standard C and C++ standard libraries. The CodeWarrior CD contains the source code of these libraries. Freescale has customized these libraries and adapted the runtime libraries for MPC55xx/MPC56xx development.

For more information on MSL, refer to *MSL C Reference* and *MSL C++ Reference*.

For more information on EWL, refer to *EWL C Reference* and *EWL C++ Reference*.

PC-lint

Your CodeWarrior build tools support separately purchased PC-lint software, which finds errors and inconsistencies in C programs. This software verifies that your source code conforms to any of these standards: Kernighan & Ritchie (K&R) C, ANSI C, or ANSI/ISO C++. PC-lint also verifies conformance with such other standards as the Motor Industry Software Reliability Association (MISRA) standard.

PC-lint checks source code more closely than the C/C++ compiler can. The tool finds bugs, inconsistencies, non-portable constructs, redundant code, and other such problems. For more information about the PC-lint software package, go to:

http://www.gimpel.com

Development Process

In general, when writing software for an MPC55xx/MPC56xx microcontroller, follow these steps:

1. Connect a debug probe between your development PC and the MPC55xx/MPC56xx board you are using.
2. Use the CodeWarrior editor to write your source code.
3. Use the CodeWarrior build tools to generate an ELF executable from your source code.
4. Use the debugger of your choice to debug this executable.

NOTE Your CodeWarrior Build Tools product comes with the P&E In-Circuit-Debugger (ICD).

Related Documentation

CodeWarrior documentation is in the `\Help\pdf` directory of your CodeWarrior installation directory. Table 1.2 lists these documents, as well as non-CodeWarrior documents that provide additional, relevant information.

Table 1.2 Related Power Architecture Documentation

Document	Description
InstallDir`\(CodeWarrior_Examples)` directory	CodeWarrior example projects
IDE User's Guide, in directory *InstallDir*`\Help\PDF`	General IDE information
CodeWarrior Development Studio for Power Architecture Processors Build Tools Reference, in directory *InstallDir*`\Help\PDF`	Instructions for using the CodeWarrior C/C++ compiler and linker.
MSL C Reference, in directory *InstallDir*`\Help\PDF`	Information on the CodeWarrior standard C library
MSL C++ Reference, in directory *InstallDir*`\Help\PDF`	Information on the CodeWarrior standard C++ library
System V Application Binary Interface, Third Edition, published by UNIX System Laboratories, 1994 (ISBN 0-13-100439-5)	Power Architecture Application Binary Interface (Power Architecture EABI) information
System V Application Binary Interface, PowerPC Processor Supplement, published by Sun Microsystems and IBM (1995).	Power Architecture Application Binary Interface (Power Architecture EABI) information
Power Architecture Embedded Binary Interface, 32-Bit Implementation, published by Freescale Semiconductor, Inc.; available at World Wide Web address: `http://www.freescale.com/files/32bit/doc/ref_manual/E500ABIUG.pdf`	Power Architecture Application Binary Interface (Power Architecture EABI) information
EREF: A Programmer's Reference Manual for Freescale Embedded Processors (Including the e200 and e500 Families) (2007).	Explanation of the e500 core complex programming model
Variable-Length Encoding (VLE) Programming Environments Manual: A Supplement to the EREF (2007).	Variable-Length Encoding programming information

Table 1.2 Related Power Architecture Documentation

Document	Description
Various Power Architecture processor manuals, available at: `http://www.freescale.com/powerarchitecture`	Information specific to individual processors of the Power Architecture family
Executable and Linking Format Specification, Version 1.2, available at: `http://refspecs.freestandards.org/elf/elf.pdf`	Documents the format of an ELF file.
DWARF Debugging Information Format Specification, Version 2.0.0, available at: `http://dwarfstd.org/doc/dwarf-2.0.0.pdf`	Documents the DWARF 2.0 symbolic debugging information format.

2

Creating Projects

This chapter explains how to create projects using the MPC55xx New Project Wizard.

The New Project Wizard makes project creation fast and easy. Just step through the wizard's pages, choosing appropriate options for your application as you go.

Based upon your choices, the New Project Wizard generates a "stub" project that automatically includes the correct files, libraries, and settings. To create your final program, just add your custom code to this foundation.

Among others, the New Project Wizard lets you make these choices:

- The MPC55xx/MPC56xx device to target
- Programming language
- VLE code generation
- PC-lint support
- Floating-point support

In this chapter:

Creating a Project for a Single Core Device

This section explains how to create a project for one of the single-core devices in the MPC55xx or MPC56xx families.

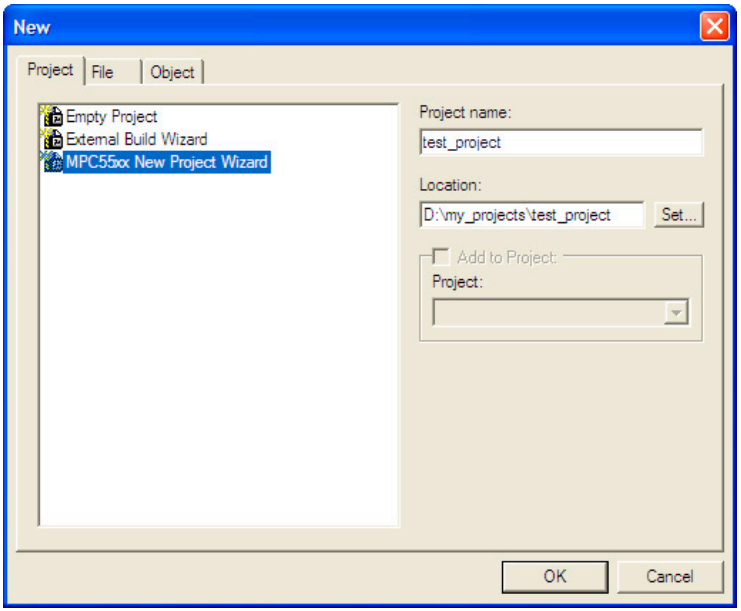
NOTE Refer to "Creating a Project for a Multicore Device" for instructions explaining how to create a project for a multicore device. Refer to "Creating a Project for a Multicore Device that Supports LSM/DPM" for instructions explaining how to create a project for a multicore device that supports LSM/DPM.

To create a project for a single-core MPC55xx/MPC56xx target board, follow these steps:

1. From the CodeWarrior IDE's menu bar, select **File > New**.

   The **New** dialog box (Figure 2.1) appears.

Figure 2.1 New Dialog Box

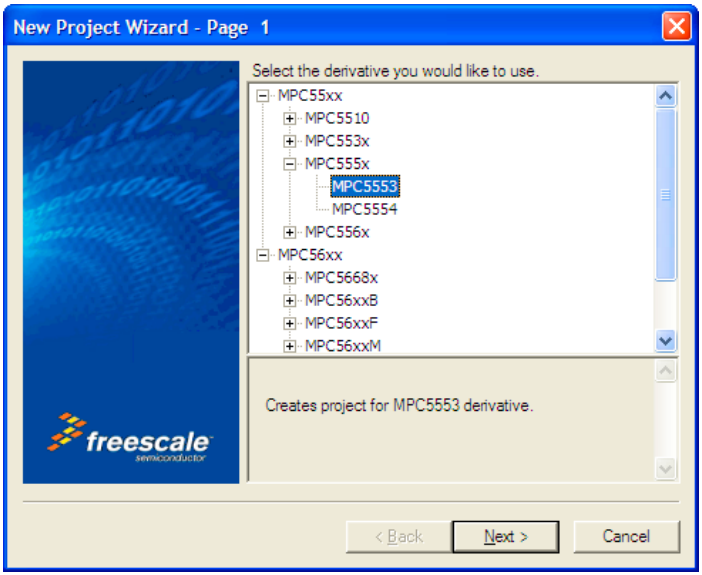


2. In the Project name text box, type the name of the new project.
3. In the location text box, type the path in which to create this project.

   Alternatively, click **Set** to display a dialog box with which to select this path.
4. Click **OK**.

   The New Project Wizard starts and displays the microcontroller derivatives page (Figure 2.2).

Figure 2.2 New Project Wizard — Microcontroller Derivatives Page



5. From the derivatives list of this page, select one of the single-core derivatives listed below.
   - MPC553x
   - MPC555x
   - MPC556x
   - MPC56xxA
   - MPC56xxB
   - MPC56xxF
   - MPC56xxM
   - MPC56xxP
   - MPC56xxS
   - MPC5644B
   - MPC5645B
   - MPC5646B

- MPC5644C
- MPC5645C
- MPC5646C
- MPC5604E
- MPC5676R

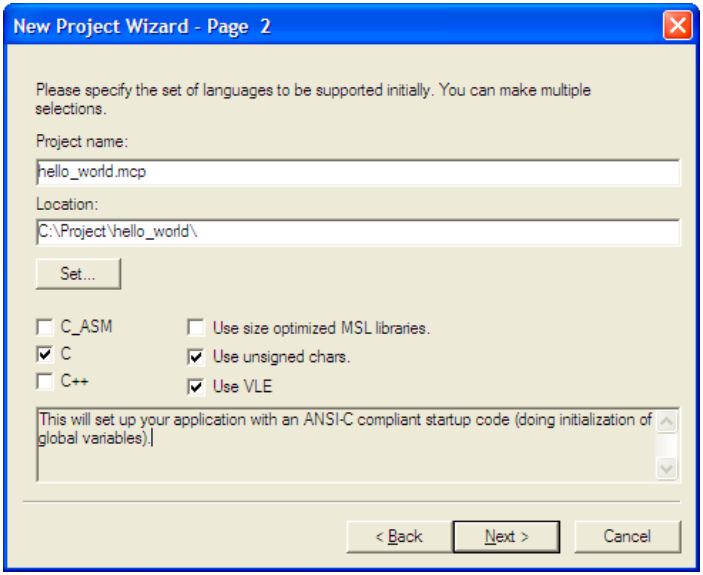
NOTE Although devices MPC5644C, MPC5645C, MPC5646C, and MPC5676R have two cores, they are listed here because code *must* be generated for both cores; as a result, the wizard does not provide the option to select which core you want to use, just as for a real single-core device.

6. Click **Next**.

   The languages and libraries page (Figure 2.3) appears.

NOTE The Project name and Location text boxes of this wizard page default to the values you entered in the **New** dialog box.

Figure 2.3 New Project Wizard — Languages and Libraries Page



7. In the languages group of this page, select the programming language(s) you will use in this project:

- C_ASM

  ANSI-C source code with a call to an assembly language function

- C

  ANSI-C source code (default option)

- C++

  C++ source code

NOTE Based on these selections, the wizard automatically includes the required startup code in the new project.

8. Check the **Use size optimized MSL libraries** checkbox to configure the project to use the size-optimized versions of the Main Standard C/C++ Libraries (MSL).

   Clear this checkbox to use the time-optimized versions of these libraries.

9. Check the **Use unsigned char** checkbox to configure the project to use versions of the MSL and runtime libraries that treat variables declared as type `char` as if declared as type `unsigned char`.

   Clear this checkbox to use versions of these libraries that treat variables declared as type `char` as if declared `signed char`.

10. Check the **Use VLE** checkbox to configure the project to generate variable length encoding (VLE) instructions and to use versions of the support libraries containing VLE instructions.

    Clear this checkbox to configure the project such that it does not use VLE instructions.

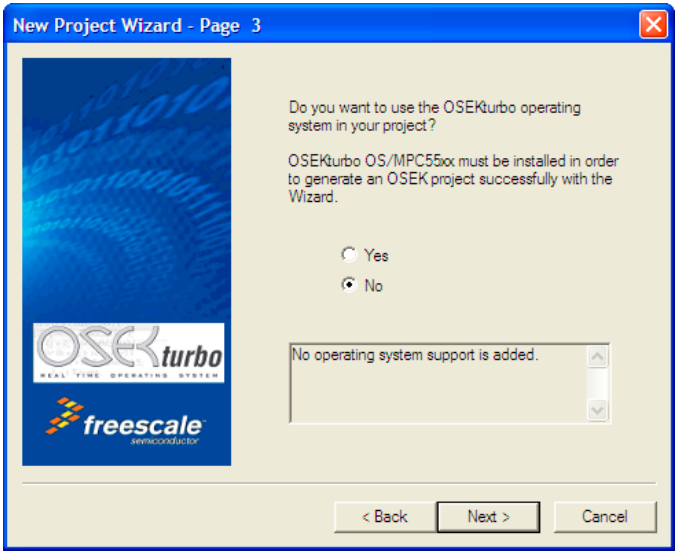
NOTE If the target microcontroller derivative does not support VLE, then uncheck the **Use VLE** checkbox. (The MPC5553 and MPC5554 microcontroller derivatives do not support VLE.)

11. Click **Next**.

    If OSEKturbo is installed, the OSEK page (Figure 2.4) appears.

NOTE If OSEKturbo is not installed, the PC-lint page (Figure 2.5) appears.

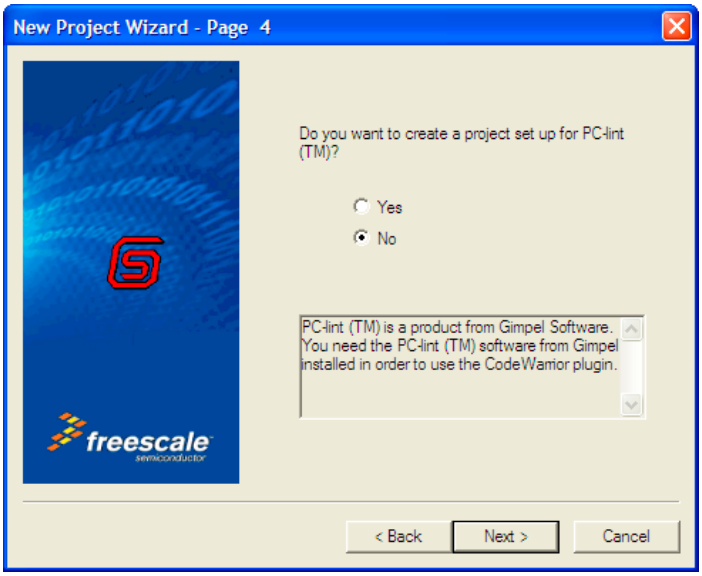
Figure 2.4 New Project Wizard — OSEKturbo Page



12. To create a project that uses OSEKturbo, select **Yes**; otherwise, select **No**.
13. Click **Next**.

    The PC-lint page (Figure 2.5) appears.

Figure 2.5 New Project Wizard — PC-lint Page



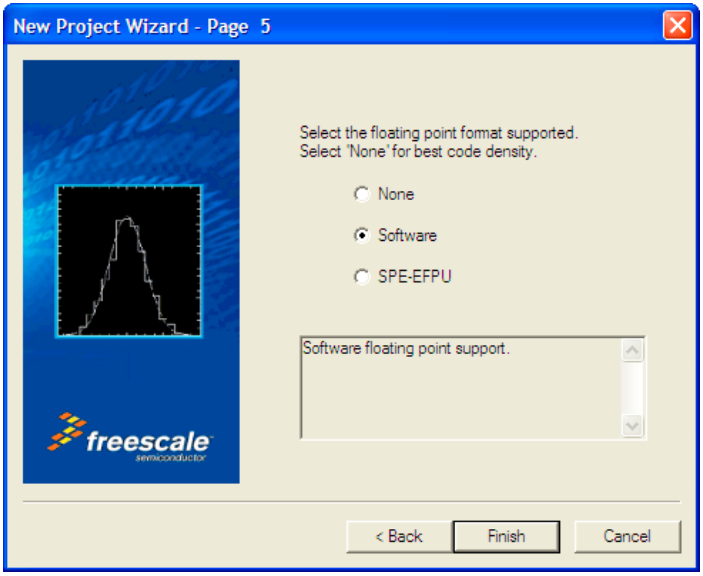
14. To create a project that uses PC-lint, select **Yes**; otherwise, select **No.**

NOTE If you select **Yes**, you must install PC-lint on your development PC. To obtain this software, visit the Gimpel Software website: http://www.gimpel.com.

15. Click **Next**.

The floating-point support page (Figure 2.6) appears.

Figure 2.6 New Project Wizard – Floating-Point Support Page



16. In this page, select the type of floating-point support the new project requires:
    - None

      No floating-point support. Project source code files cannot contain floating-point operations.
    - Software

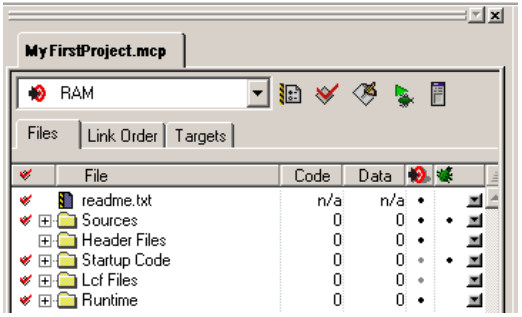
      All floating-point operations are performed by software routines; a C/C++ runtime library containing these routines is included in the new project.
    - SPE-EFPU

      Single-precision floating-point operations are performed by the e200 core's SPE-EFPU (Signal Processing Engine-Embedded Floating-Point Unit) auxiliary processing unit.

      Double-precision floating-point operations are performed by software routines. A runtime library containing these routines is included in the new project.
17. Click **Finish**.

    The wizard creates a project according to you specifications and displays it in a project window (Figure 2.7).

Figure 2.7 Project Window



18. Select **Project > Make**.

    The CodeWarrior IDE compiles the project's source code and links the resulting object code into an executable ELF file.

Use the debugger of your choice to debug this file.

Creating a Project for a Multicore Device

The CodeWarrior for MPC55xx/MPC56xx product lets you create a project that generates a binary for each core of a multicore microcontroller.

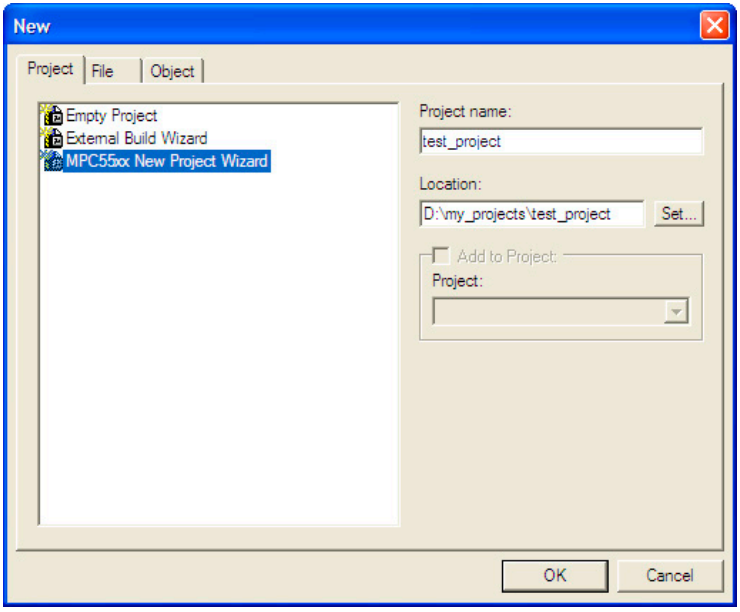
NOTE See "Creating a Project for a Single Core Device" for instructions explaining how to create a project for a single core device. See "Creating a Project for a Multicore Device that Supports LSM/DPM" for instructions explaining how to create a project for a multicore device that supports LSM/DPM.

To create a project for a multi-core MPC55xx/MPC56xx device, follow these steps:

1. From the CodeWarrior IDE's menu bar, select **File > New**.

   The **New** dialog box appears.

Figure 2.8 New Dialog Box

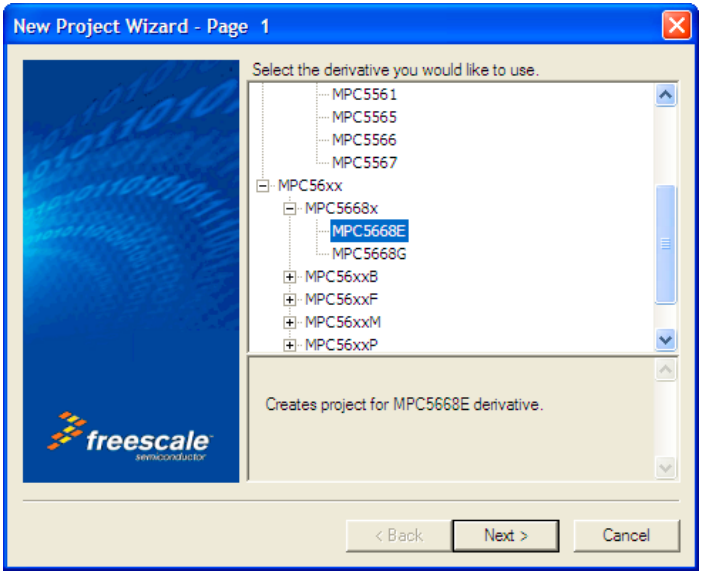


2. From the list on the left side of the **New** dialog box, select `MPC55xx New Project Wizard`.
3. In the Project name text box, type the name of the new project.
4. In the location text box, type the path in which to create the project.

   Alternatively, click **Set** to display a dialog box with which to select this path.
5. Click **OK**.

   The New Project Wizard starts and displays the microcontroller derivatives page (Figure 2.9).

Figure 2.9 New Project Wizard — Microcontroller Derivatives Page

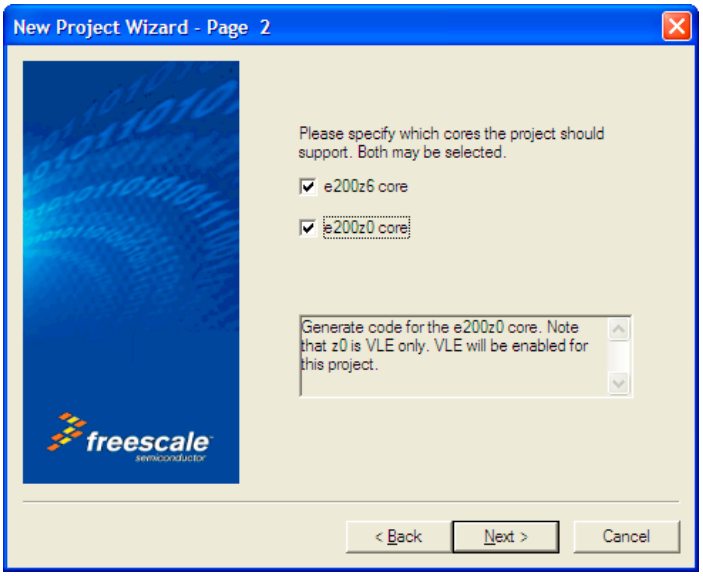


6. From the derivatives list of this page, select one of the multicore derivatives listed below.
   - MPC5514E
   - MPC5514G
   - MPC5516E
   - MPC5516G
   - MPC5517E
   - MPC5517G
   - MPC5668E
   - MPC5668G
7. Click **Next**.

   The multicore microcontroller page (Figure 2.10) appears.

NOTE The core names this page displays vary depending on the device chosen on the microcontroller derivatives page. Figure 2.10 shows the multicore microcontroller page for the MPC5668x microcontroller.

Figure 2.10 New Project Wizard — MPC5668x Multicore Microcontroller Page



8. Select the core or cores for which to generate a binary.

   For example, for the MPC5668x microcontroller, you can choose to generate a binary for this chip's z0 core, for its z6 core, or for both.

9. Click **Next**.

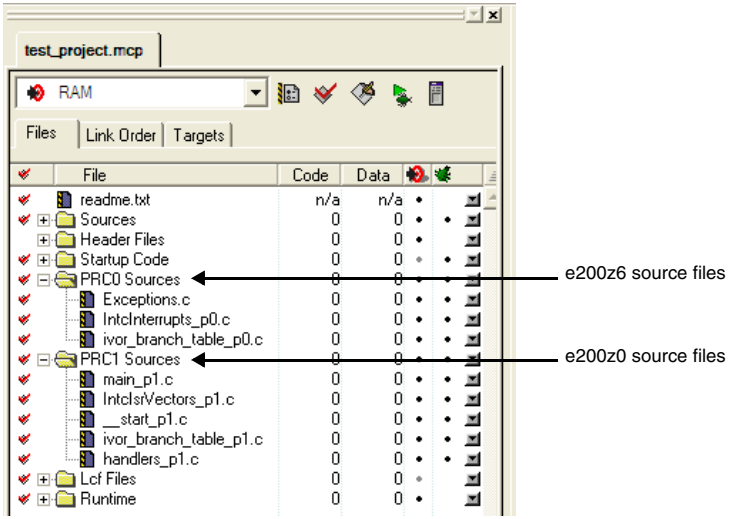
   The languages and libraries page appears.

10. Complete the rest of the wizard by following the instructions in the topic "Creating a Project for a Single Core Device", starting from step 11.

11. Click **Finish**.

    The wizard creates a project according to you specifications and displays it in a project window (Figure 2.11).

Figure 2.11 Project Window for a Multicore MPC5668E Project



12. Select **Project > Make**.

    The IDE compiles project's source code and generates a single executable file that contains instructions for each core you selected in the New Project Wizard's multicore microcontroller page.

You can now use the debugger of your choice to debug this file.

Creating a Project for a Multicore Device that Supports LSM/DPM

Some members of the MPC55xx/MPC56xx family have two cores that can be run in either of these modes:

- Lock-Step Mode (LSM)
- Decoupled Parallel Mode (DPM)

LSM is for safety-critical systems that require redundancy. DPM provides superior performance. The performance of a device running in DPM is about 1.6 times the performance of this device running in LSM at the same frequency.

These devices support LSM/DPM:

- MPC5643L
- MPC5675K

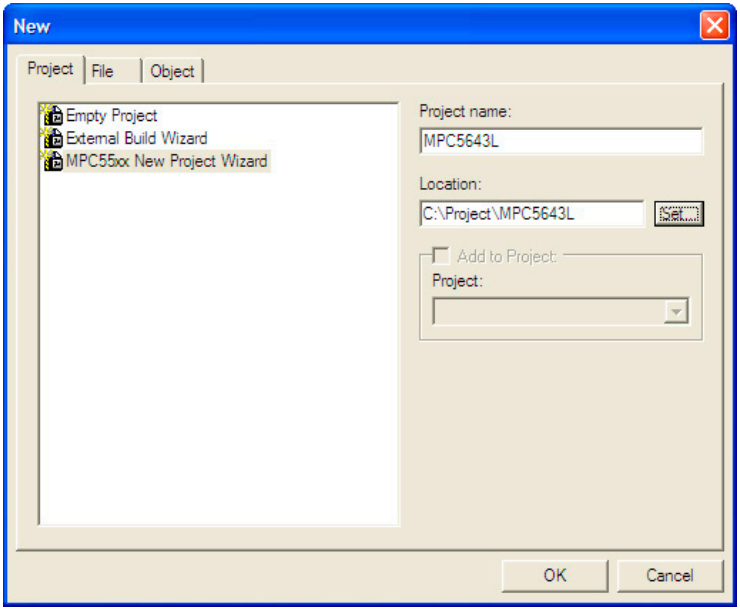
NOTE Refer to "Creating a Project for a Single Core Device" for instructions explaining how to create a project for a single core device. Refer to "Creating a Project for a Multicore Device" for instructions explaining how to create a project for a multicore device that does not support LSM/DPM.

To create a project for a device that supports LSM/DPM mode, follow these steps:

1. From the CodeWarrior IDE's menu bar, select **File > New**.

   The **New** dialog box appears.

Figure 2.12 New Dialog Box

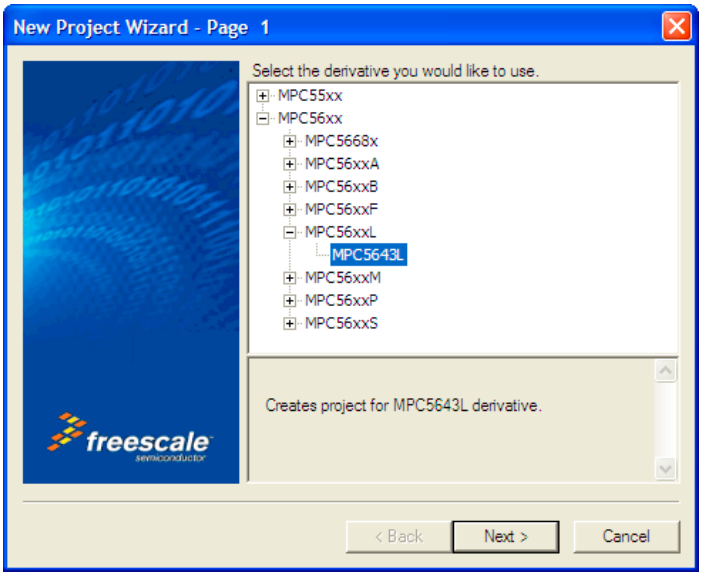


2. From the list on the left side of the **New** dialog box, select `MPC55xx New Project Wizard`.
3. In the Project name text box, type the name of the new project.
4. In the location text box, type the path in which to create the project.

   Alternatively, click **Set** to display a dialog box with which to select this path.
5. Click **OK**.

   The New Project Wizard starts and displays the microcontroller derivatives page (Figure 2.13).

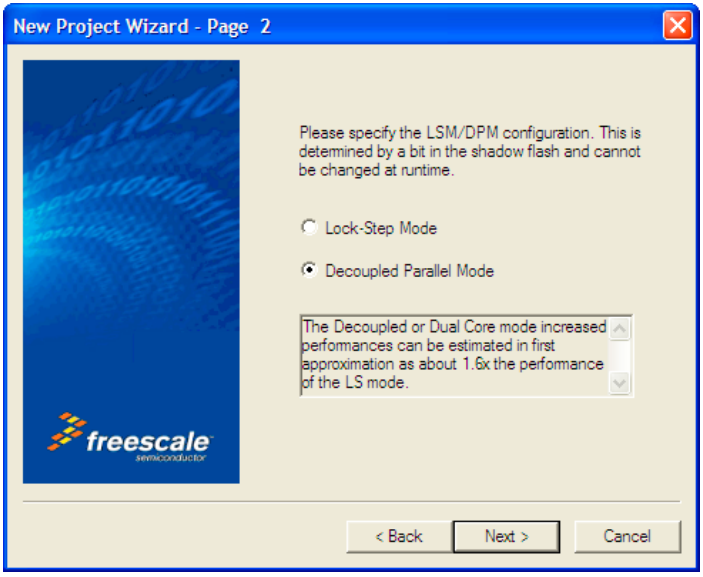
Figure 2.13 New Project Wizard — Microcontroller Derivatives Page



6. From the derivatives list of this page, select one of the multicore derivatives listed below. These devices support LSM/DPM.
   - MPC5643L
   - MPC5675K
7. Click **Next**.

   The LSM/DPM configuration page appears.

Figure 2.14 New Project Wizard — LSM/DPM Configuration Page



8. If you want the selected device to run in LSM, select **Lock-Step Mode**.

   If you want the device to run in DPM, select **Decoupled Parallel Mode**.

9. Click **Next**.

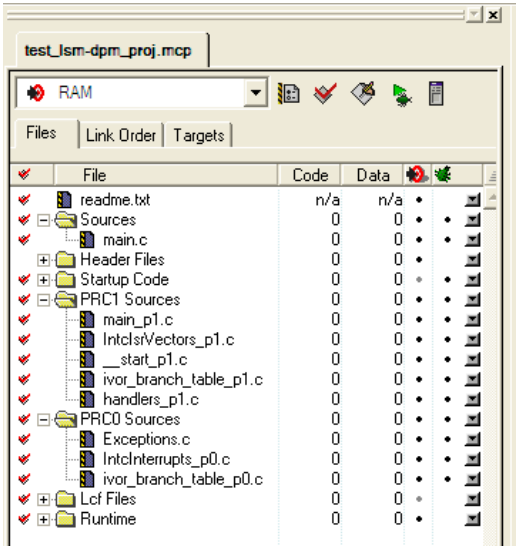
   The languages and libraries page appears.

10. Complete the rest of the wizard by following the instructions in the topic "Creating a Project for a Single Core Device", starting from step 11.

11. Click **Finish**.

    The wizard creates a project according to you specifications and displays it in a project window (Figure 2.15).

Figure 2.15 Project Window for a Multicore MPC5643L LSM/DPM Project



12. Select **Project > Make**.

    The IDE compiles and assembles the project's source code and generates a single executable file that runs the selected device in the selected mode, either LSM or DPM.

3

Target Settings Reference

This chapter documents the target settings panels that are specific to the CodeWarrior Development Studio for MPC55xx/MPC56xx Microcontrollers product. Use these panels to control the behavior of the compiler, linker, debugger, and other software development tools included in this product.

NOTE For documentation of the target settings panels common to all CodeWarrior products, refer to the *IDE User's Guide* and the *Power Architecture™ Build Tools Reference*.

In this chapter:

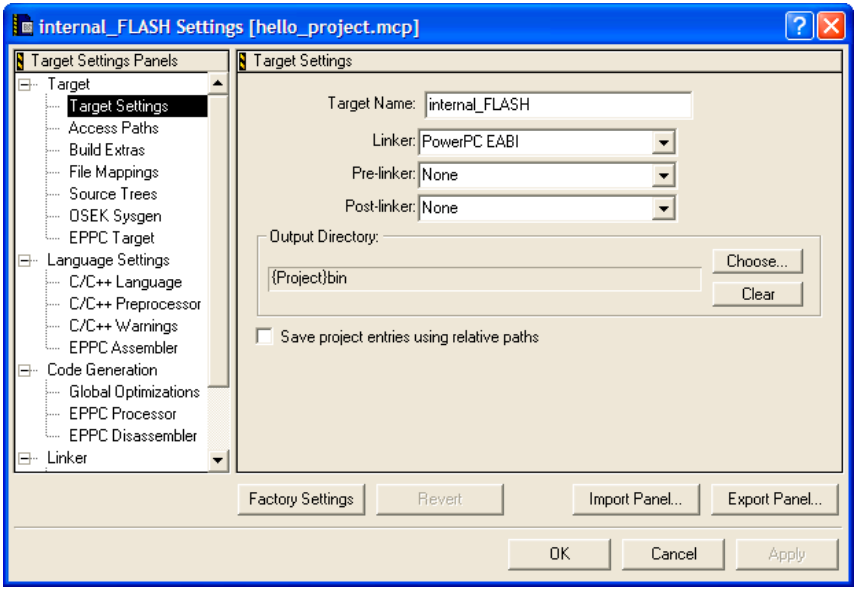
Target Settings Overview

In a CodeWarrior project, each build target has its own settings for compiling, linking, and other aspects of code generation. Your controls for these settings are the target settings *panels* that you access through the **Target Settings** window.

To open this window, select **Edit > *Build Target Name***, where ***Build Target Name*** is a placeholder for the project's currently selected build target. Another way to bring up the **Target Settings** window is to click the **Targets** tab of the project window and then double-click one of the listed build target names.

Figure 3.1 shows the **Target Settings** window (see the *CodeWarrior IDE User's Guide* for a description of the window's elements). Use the list of panels on the left side of the this window to display any settings panel. If necessary, click the expand control to see a category's list of panels. Clicking a panel's name displays that panel in the right side of the **Target Settings** window.

Figure 3.1 Target Settings Window



Note these buttons, at the bottom of the window:

- **Apply** — Implements your changes, leaving the ***Target*** **Settings** window open. This lets you bring a different target settings panel to the front of the window.
- **OK** — Implements your changes, closing the ***Target*** **Settings** window. Use this button when you make the last of your settings changes.
- **Revert** — Changes panel settings back to their most recently saved values. (Modifying any panel settings activates this button.)
- **Factory Settings** — Restores the original default values for the panel.
- **Import Panel** — Copies panel settings previously saved as an XML file.
- **Export Panel** — Saves settings of the current panel to an XML file.

NOTE If you use the New Project Wizard to create a new project, the wizard assigns default values to all options of all settings panels.

e200 Core Settings Panels

Table 3.1 lists the target settings panels used to control the build tools for the Power Architecture e200 core. Each table entry includes a link or cross reference to detailed documentation of a settings panel.

NOTE If you have the separately purchased PC-lint software package, your CodeWarrior build tools also include two PC-lint panels. Section PC-lint Settings Panels, at the end of this chapter, explains these additional panels.

Table 3.1 e200 Core Settings Panels

Panel	Explanation
Target Settings	Refer to "Target Settings"
Access Paths	See CodeWarrior IDE User's Guide
Build Extras	See CodeWarrior IDE User's Guide
File Mappings	See CodeWarrior IDE User's Guide
Source Trees	See CodeWarrior IDE User's Guide
OSEK Sysgen	Refer to "OSEK Sysgen"
EPPC Target	Refer to "EPPC Target"
C/C++ Language	See IDE User's Guide
C/C++ Preprocessor	Refer to "C/C++ Preprocessor"
C/C++ Warnings	Refer to "C/C++ Warnings"
EPPC Assembler	Refer to "EPPC Assembler"
Global Optimizations	See CodeWarrior IDE User's Guide
EPPC Processor	Refer to "EPPC Processor"
EPPC Disassembler	Refer to "EPPC Disassembler"
EPPC Linker	Refer to "EPPC Linker"
EPPC Linker Optimizations	Refer to "EPPC Linker Optimizations"
Custom Keywords	See CodeWarrior IDE User's Guide

Target Settings

Configure the **Target Settings** panel (Figure 3.2) first, because it affects other settings panels. Use this panel to specify the name of your build target and to select a linker. When you select a linker, you specify the target operating system or CPU: this is the setting that controls the availability of elements in other settings panels. Table 3.2 explains the elements of this panel.

NOTE The **Target Settings** panel is not the same as the **EPPC Target** panel. You specify the build target in the **Target Settings** panel; you set other target-specific options for the EPPC target in the **EPPC Target** panel.

Figure 3.2 Target Settings Panel

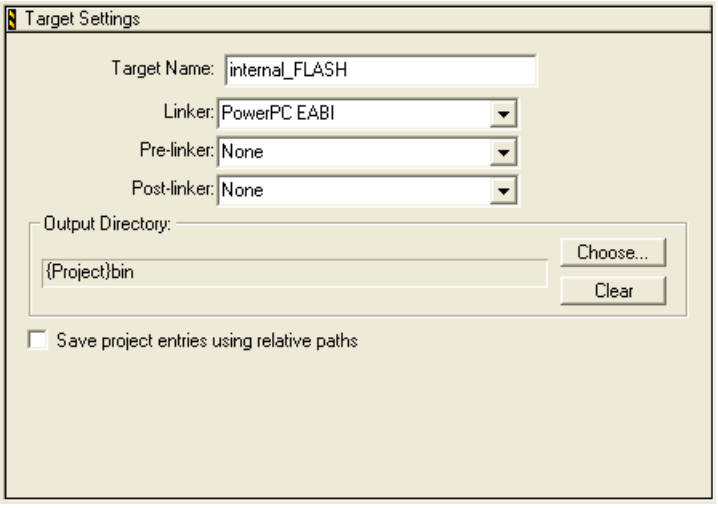


Table 3.2 Target Settings Panel Elements

Element	Purpose	Comments
Target Name text box	Specifies the name of the build target: 26 or fewer characters. This name appears subsequently on the Targets page of the project window.	This build-target name is *not* the name of your final output file.

Table 3.2 Target Settings Panel Elements (*continued*)

Element	Purpose	Comments
Linker list box	Specifies the linker: • PowerPC EABI — for an e200 core. • PCLint Linker — for PC-lint source code checking (bugs, inconsistencies, and non-portable constructs). • External Build Linker —allows the IDE to use an external application to perform the task.	This selection affects the list of panels in the Target Settings Panels pane. PC-lint is a development tool from Gimpel Software (www.gimpel.com). You must obtain and install a copy of this tool before a CodeWarrior build target can use it.
Pre-linker list box	Specifies the pre-linker that performs work on object code before linking. Selections are **None** and **BatchRunner**.	If you select **BatchRunner**, a new panel name appears in the Target Settings Panels pane.
Post-linker list box	Specifies the post-linker that performs additional work on the final executable. Selections are **None** and **BatchRunner**.	If you select **BatchRunner**, a new panel name appears in the Target Settings Panels pane.
Output Directory text box	Specifies the directory for the final linked output file. To specify a non-default directory, click **Choose**. To clear this text box, click **Clear**.	
Save project entries using relative paths checkbox	Clear — Specifies minimal file searching; each project file must have a unique name. Checked — Specifies relative file searching; project may include two or more files that have the same name.	

OSEK Sysgen

Use the **OSEK Sysgen** panel (Figure 3.3) to control the output of the OSEK Sysgen tool.

When you build a CodeWarrior build target that contains an object implementation language (OIL) file, the OSEK Sysgen tool compiles the OIL file and generates C language files used in the generation of an OSEK operating system image as well as other types of files. The OSEK Sysgen panel lets you define the names, locations, and other attributes of these files.

Next, the CodeWarrior C compiler compiles the generated C language files, the OSEK operating system's source code, and any application source code files the build target contains. Finally, the CodeWarrior linker links the resulting object code into an executable OSEK operating system image that contains your application.

Compilation of the OSEK operating system source code depends on the definition of several macros; the OSEK Sysgen tool helps with these macro definitions. Specifically, the tool generates file `options.h`, *which you must include in your build target's prefix file*. The tool also defines macros `APPTYPESH`, `OSPROPH`, and `OSCFGH`, extracting macro values from corresponding user types, property, and object-declaration files. Table 3.3 explains the elements of this panel.

NOTE We recommend that you not edit the generated files. Doing so may lead to data inconsistency, compilation errors, or unpredictable application behavior.

Figure 3.3 OSEK Sysgen Settings Panel

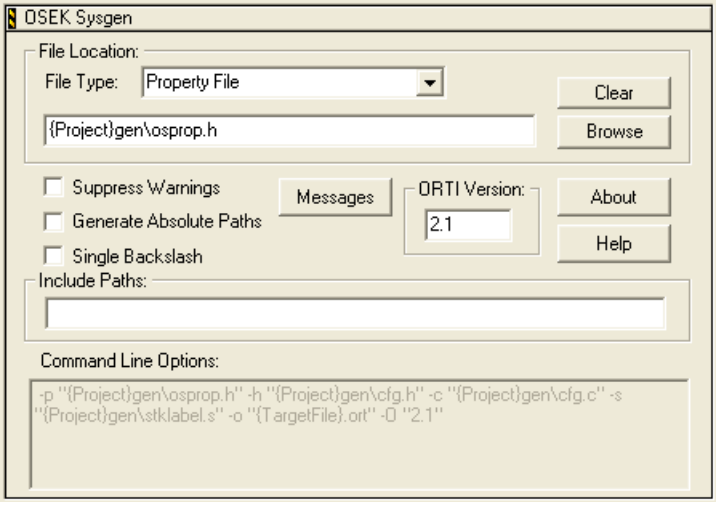


Table 3.3 OSEK Sysgen Settings Panel Elements

Element	Purpose	Comments
File Type list box	Specifies the file type: • **Property file** — header file that describes current operating-system configuration. Used at compile time to build the OS kernel; contains preprocessor directives `#define` and `#undef`. • **Objects Declaration File** — header file that contains definitions of data types, constants, and variable external declarations necessary to describe system objects. • **Objects Definition File** — source file that contains initialized data and allocates memory for system objects. • **Stack Labels File** — file that contains labels for the bottom and top of the stack, for extended tasks implemented in the OSEK OS. • **ORTI (OSEK Run Time Interface) File** — file that contains internal OSEK OS data, available to the ORTI Aware Debugger. • **Sysgen Tool File** — file that specifies the path and name of the OSEK Sysgen utility, which processes an OIL file.	You can assign any path and name to any file type, but default names are: Property — `{Project}gen\osprop.h`, Objects Declaration — `{Project}gen\cfg.h`, Objects Definition — `{Project}gen\cfg.c`, Stack Labels — `{Project}gen\stklabel.s`, ORTI — same as the path and name of the `.abs` file. Sysgen Tool — `{Compiler}osek\shared\bin\sysgen.exe`,

Table 3.3 OSEK Sysgen Settings Panel Elements (*continued*)

Element	Purpose	Comments
	• **Sysgen Command Line File** — optional file that contains additional, advanced command-line options for the OSEK Sysgen utility. • **User Types File** — file that contains definitions of your message types; defines macro `APPTYPESH` equal to the location of this file. • **Prefix File Path** (for `option.h`) — optional file that contains a path for file options.h, which you must include in the prefix file of your build target.	Sysgen Command Line — no default for this file type, User Types — `{Project}Sources\usertypes.h`, Prefix Path — `{Project}gen`,
File Location text box	Specifies the path and name for the file that the **File Type** list box specifies. Type this entry, or click **Browse**, then use the subsequent dialog box to specify the location. Browsing works with either an absolute path or a location macro. Clicking **Clear** removes the contents of this text box.	To make project definitions portable, you may use any of these macros: • `{Compiler}` — path to the CodeWarrior build tools installation. • `{Project}` — path to the `.mcp` file. • `{System}` — path to the operating system.
Suppress Warnings checkbox	Clear — Allows warnings. Activates the **Messages** button, which you can use to suppress individual warnings. Checked — Suppresses all warnings; deactivates the **Messages** button.	

Table 3.3 OSEK Sysgen Settings Panel Elements (*continued*)

Element	Purpose	Comments
Messages button	Suppress Messages dialog box, which you can use to suppress individual messages. Selecting **Enable All** in this dialog box is equivalent to clearing the **Suppress Warnings** checkbox; selecting **Disable All** is equivalent to checking the **Suppress Warnings** checkbox.	
Generate Absolute Paths checkbox	Clear — Lets Sysgen generate relative paths in the object definition file. Checked — Sysgen generates absolute paths in the object definition file.	
Single Backslash checkbox	Clear — Does not use single backslash characters for include path definitions. Checked — Uses single backslash characters for include path definitions.	Freescale MPC targets require single backslash characters.
ORTI Version text box	Lets you specify any appropriate alternative OSEK Run Time Interface (ORTI) version. For information about this version, click **About**; for additional information, click **Help**.	
Include Paths text box	Specifies include paths for files that the `.oil` file includes.	Separate each directory path with a comma or semicolon.
Command Line Options area	Shows a summary of options in effect.	

EPPC Target

Use the **EPPC Target** settings panel (Figure 3.4) to specify the name and configuration of your final output file. Table 3.4 explains the elements of this panel.

Figure 3.4 EPPC Target Settings Panel

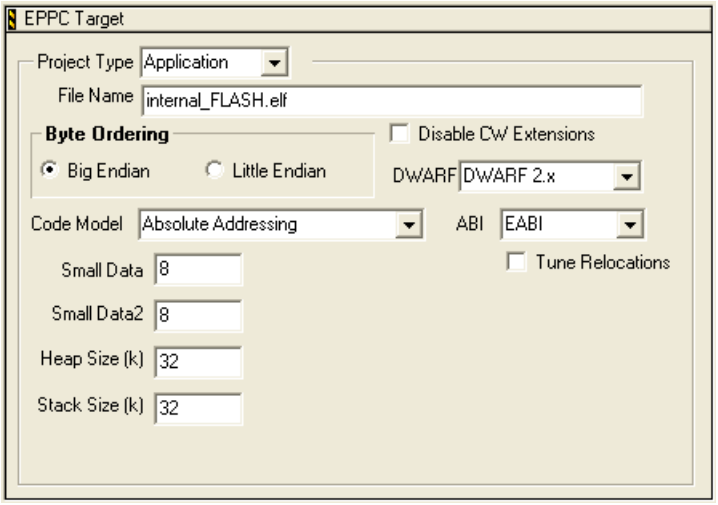


Table 3.4 EPPC Target Settings Panel Elements

Element	Purpose	Comments
Project Type list box	Specifies the kind of project. Options are: • Application • Library • Partial Link	Choosing **Library** or **Partial Link** removes from this panel irrelevant elements Heap Size, Stack Size, and Tune Relocations. Choosing **Partial Link** adds to this panel elements Optimize Partial Link, Deadstrip Unused Symbols, and Require Resolved Symbols.

Table 3.4 EPPC Target Settings Panel Elements (*continued*)

Element	Purpose	Comments
File Name text box	Specifies the name of the output file. End the file name of an executable application that the wizard generates with extension `.out` or `.elf`. End a library file name with extension `.a`.	If you specify **S-record** or **Map-file generation** (in the **EPPC Linker** panel), the system replaces the name extension with `.mot` or `.MAP`.
Big Endian option button	Specifies big endian format for generated code and data: the most significant byte comes first.	
Little Endian option button	Specifies little endian format for generated code and data: the least significant byte comes first.	
Disable CW Extensions checkbox	Clear — Retains extensions, minimizing C application size. Also appropriate for assembly files and C++ libraries. Checked — Disables C extensions possibly incompatible with third-party compilers/linkers.	If checked, the CodeWarrior linker cannot deadstrip files. Not all third-party linkers require checking this checkbox.
DWARF list box	Specifies the version of the Debug With Arbitrary Record Format.	The linker ignores debugging information not in the specified version.
ABI list box	Specifies the Application Binary Interface for function calls and structure layout.	
Tune Relocations checkbox	Clear — Ignores relocation tuning possible for EABI or SDA PIC/PID. Checked — For EABI, changes 14-bit branch relocations to 24 bits, if they cannot reach the calling site from the original location. For SDA PIC/PID, changes absolute-addressed data references to use a small data register instead of r0; changes absolute code to use PC relative relocations.	This checkbox appears only if the **Project Type** list box specifies Application.

Table 3.4 EPPC Target Settings Panel Elements (*continued*)

Element	Purpose	Comments
Code Model list box	Specifies addressing mode for the generated executable file: **Absolute Addressing** or **SDA PIC/PID**.	
Small Data text box	Specifies threshold size (bytes) for items the linker treats as small data.	The linker stores small data items in the Small Data address space; the compiler can generate faster code to address such data.
Small Data2 text box	Specifies threshold size (bytes) for read-only items the linker treats as small data.	The linker stores read-only small data items in the Small Data2 address space; the compiler can generate faster code to address such data.
Heap Size text box	Specifies kilobytes of RAM allocated for the heap, which your program uses if it calls `malloc` or `new`.	This checkbox appears only if the **Project Type** list box specifies Application. Combined heap/stack allocation must not exceed available RAM.
Stack Size text box	Specifies kilobytes of RAM allocated for the stack.	This checkbox appears only if the **Project Type** list box specifies Application. Combined heap/stack allocation must not exceed available RAM.
Optimize Partial Link checkbox	Clear — Output file remains as if you passed the `-r` argument in the command line. Checked — Specifies direct downloading of partial link output.	This checkbox appears only if the **Project Type** list box specifies **Partial Link**. Text immediately after this table explains more about optimizing partial links.

Table 3.4 EPPC Target Settings Panel Elements (*continued*)

Element	Purpose	Comments
Deadstrip Unused Symbols checkbox	Clear — Linker does *not* deadstrip unused symbols. Checked — Linker deadstrips all unused symbols. This reduces program size, by removing symbols that neither the main entry point or force-active entry points reference.	This checkbox appears only if the **Project Type** list box specifies Partial Link.
Require Resolved Symbols checkbox	Clear — Linker does not have to resolve all symbols of the partial link. Checked — Linker must resolve all symbols in the partial link.	This checkbox appears only if the **Project Type** list box specifies **Partial Link**. Check this option if your RTOS does not allow unresolved symbols.

Check the **Optimize Partial Link** checkbox to directly download the output of your partial link. This instructs the linker to:

1. Let the project use a linker command file (LCF). This is important for correct merging of all diverse sections into either `.text`, `.data`, or `.bss` sections. If you do not let an LCF do this merge, the debugger may not be able to show source code properly.
2. Allow optional dead stripping. (This is recommended — but the project must have at least one entry point for the linker to know how to dead strip.)
3. Collect all of the static constructors and destructors in a similar way to the tool `munch`.

NOTE Do not use **munch** yourself, because the linker needs to put the C++ exception handling initialization as the first constructor. If **munch** is in your makefile, you need an optimized build.

4. Change common symbols to `.bss` symbols, letting you examine variables in the debugger.
5. Allow a special type of partial link that has no unresolved symbols — the same as the Diab linker's `-r2` command-line argument.

C/C++ Preprocessor

Use the **C/C++ Preprocessor** settings panel (Figure 3.5) to configure preprocessing options. Table 3.5 explains the elements of this panel.

Figure 3.5 C/C++ Preprocessor Settings Panel

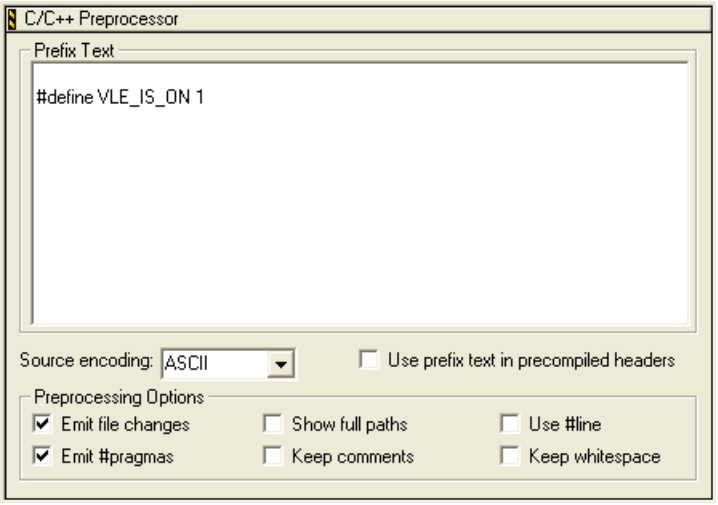


Table 3.5 C/C++ Preprocessor Settings Panel Elements

Element	Purpose	Comments
Prefix Text area	Displays all prefix text created by option selections.	
Source encoding list box	Specifies type: • ASCII • Autodetect • System • UTF-8 • Shift-JIS • EUC-JP • ISO-2022-JP	

Table 3.5 C/C++ Preprocessor Settings Panel Elements (*continued*)

Element	Purpose	Comments
Emit file changes checkbox	Clear — Compiler does *not* emit file changes. Checked — Compiler emits file changes.	
Emit #pragmas checkbox	Clear — Compiler does *not* emit pragmas. Checked — Compiler emits pragmas.	
Show full paths checkbox	Clear — Compiler does *not* show full pathnames. Checked — Compiler shows full path names.	
Keep comments checkbox	Clear — Compiler does *not* keep comments. Checked — Compiler keeps comments.	
Use #line checkbox	Clear — Compiler does *not* use #lines. Checked — Compiler uses #lines.	
Keep whitespace checkbox	Clear — Compiler does *not* keep whitespace. Checked — Compiler keeps whitespace.	

C/C++ Warnings

Use the **C/C++ Warnings** settings panel (Figure 3.6) to control how the IDE displays language-specific warnings. Table 3.6 explains the elements of this panel.

Figure 3.6 C/C++ Warnings Settings Panel

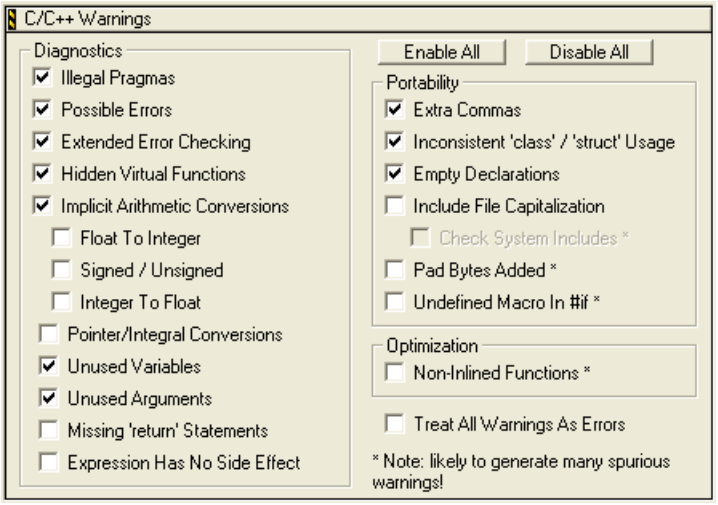


Table 3.6 C/C++ Warnings Settings Panel Elements

Element	Purpose	Comments
Illegal Pragmas checkbox	Clear — Compiler does *not* issue a warning if it does not recognize a pragma keyword. Checked — Compiler issues a warning if it does not recognize a pragma keyword.	
Possible Errors checkbox	Clear — Compiler does *not* issue a warning if it finds inappropriate semicolons or operators. Checked — Compiler issues a warning if it finds unintended semicolons, or confusing operators = and ==.	

Table 3.6 C/C++ Warnings Settings Panel Elements (*continued*)

Element	Purpose	Comments
Extended Error Checking checkbox	Clear — Compiler does *not* issue a warning if it finds a common type misuse. Checked — Compiler issues a warning if it finds a common type misuse (which is valid C/C++ code).	
Hidden Virtual Functions checkbox	Clear — Compiler does *not* issue a warning if it finds a hidden virtual function. Checked — Compiler issues a warning if it finds a hidden virtual function.	
Implicit Arithmetic Conversions checkbox	Clear — Compiler does *not* issue a warning if an operation's destination is too small. Checked — Compiler issues a warning if an operation's destination is too small for all possible results.	Checking this checkbox activates subordinate checkboxes **Float To Integer, Signed/Unsigned**, and **Integer To Float.**
Float To Integer checkbox	Clear — Compiler does *not* issue a warning if it finds a float-to-integer conversion. Checked — Compiler issues a warning if it finds a float-to-integer conversion.	This checkbox is inactive unless the **Implicit Arithmetic Conversions** checkbox is checked.
Signed / Unsigned checkbox	Clear — Compiler does *not* issue a warning if it finds a signed-to-unsigned conversion. Checked — Compiler issues a warning if it finds a signed-to-unsigned conversion.	This checkbox is inactive unless the **Implicit Arithmetic Conversions** checkbox is checked.

Table 3.6 C/C++ Warnings Settings Panel Elements (*continued*)

Element	Purpose	Comments
Integer To Float checkbox	Clear — Compiler does *not* issue a warning if it finds an integer-to-float conversion. Checked — Compiler issues a warning if it finds an integer-to-float conversion.	This checkbox is inactive unless the **Implicit Arithmetic Conversions** checkbox is checked.
Pointer/Integral Conversions checkbox	Clear — Compiler does *not* issue a warning if it finds a pointer-to-integral conversion. Checked — Compiler issues a warning if it finds a pointer-to-integral conversion.	
Unused Variables checkbox	Clear — Compiler does *not* issue a warning if it finds an unused variable. Checked — Compiler issues a warning if code does not use a declared variable.	
Unused Arguments checkbox	Clear — Compiler does *not* issue a warning if it finds an unused function argument. Checked — Compiler issues a warning if code does not use a declared function argument.	
Missing 'return' Statements checkbox	Clear — Compiler does *not* issue a warning if it detects a missing return statement. Checked — Compiler issues a warning if detects a missing return statement.	
Expression Has No Side Effect checkbox	Clear — Compiler does *not* issue a warning if it finds an expression with no side effects. Checked — Compiler issues a warning if it finds an expression with no side effects.	

Table 3.6 C/C++ Warnings Settings Panel Elements (*continued*)

Element	Purpose	Comments
Enable All button	Enables (checks) all checkboxes of the panel.	
Disable All button	Disables (clears) all checkboxes of the panel.	
Extra Commas checkbox	Clear — Compiler does *not* issue a warning if code contains extra commas. Checked — Compiler issues a warning if code contains extra commas.	
Inconsistent 'class' / 'struct' Usage checkbox	Clear — Compiler does not issue a warning if it finds inconsistent use of either keyword. Checked — Compiler issues a warning if it finds inconsistent use of either keyword.	
Empty Declarations checkbox	Clear — Compiler does *not* issue a warning if it finds a declaration that does not contain a variable. Checked — Compiler issues a warning if it finds a declaration that does not contain a variable.	
Include File Capitalization checkbox	Clear — Compiler does *not* issue a warning if it finds inappropriate capitalization. Checked — Compiler issues a warning if it finds inappropriate capitalization.	
Pad Bytes Added checkbox	Clear — Compiler does *not* issue a warning if alignment requires padding bytes. Checked — Compiler issues a warning if alignment requires padding bytes.	Checking this checkbox may lead to many spurious warnings.

Table 3.6 C/C++ Warnings Settings Panel Elements (*continued*)

Element	Purpose	Comments
Undefined Macro in #if checkbox	Clear — Compiler does *not* issue a warning if finds an undefined macro in an #if pragma. Checked — Compiler issues a warning if it finds an undefined macro in an #if pragma.	Checking this checkbox may lead to many spurious warnings.
Non-Inlined Functions checkbox	Clear — Compiler does *not* issue a warning if it finds a non-inlined function. Checked — Compiler issues a warning if it finds a non-inlined function.	Checking this checkbox may lead to many spurious warnings.
Treat All Warnings As Errors checkbox	Clear — Compilation continues, despite warnings. Checked — Compilation stops upon warnings. You must resolve warnings, just as errors, before compilation can continue.	

EPPC Assembler

Use the **EPPC Assembler** settings panel (Figure 3.7) to define the syntax that the EPPC assembler will accept for certain language elements, such as labels. Table 3.7 explains the elements of this panel.

Figure 3.7 EPPC Assembler Settings Panel

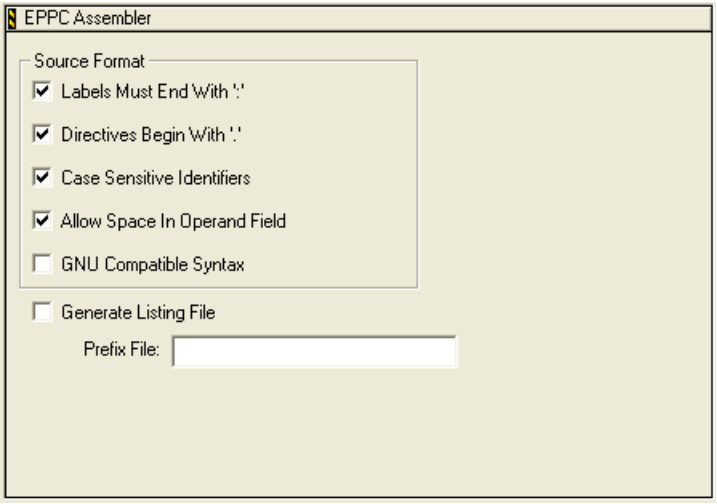


Table 3.7 EPPC Assembler Settings Panel Elements

Element	Purpose	Comments
Labels Must End With ':' checkbox	Clear — Source-file labels need not end with colon characters. Checked — Source-file labels *must* end with colon characters.	
Directives Begin With '.' checkbox	Clear — Assembly directives need not begin with period characters. Checked — Assembly directives *must* begin with period characters.	

Table 3.7 EPPC Assembler Settings Panel Elements (*continued*)

Element	Purpose	Comments
Case Sensitive Identifiers checkbox	Clear — Assembler ignores case in identifiers. Checked — Case matters in identifiers.	
Allow Space In Operand Field checkbox	Clear — Spaces are *not* allowed in fields. Checked — Spaces *are* allowed in fields.	
GNU Compatible Syntax checkbox	Clear — Indicates that your application does *not* use GNU-compatible syntax. Checked — Indicates that your application does use GNU-compatible syntax.	Text immediately after this table explains more about GNU-compatible syntax.
Generate Listing File checkbox	Clear — Specifies no listing file. Checked — Assembler generates a listing file that includes files source, line numbers, relocation information, and macro expansions.	
Prefix File text box	Specifies a file automatically included in all project assembly files.	Put common definitions in a prefix file, to avoid repeating them in all assembly files.

Check the **GNU compatible syntax** checkbox to indicate that your application uses GNU-compatible assembly syntax. This compatibility allows:

- Redefining all equates, regardless if from the `.equ` or `.set` directives.
- Ignoring the `.type` directive.
- Treating undefined symbols as imported.
- Using GNU-compatible arithmetic operators — symbols **<** and **>** mean left-shift and right-shift instead of less than and greater than; the symbol **!** means bitwise-or-not rather than logical not
- Using GNU-compatible precedence rules for operators
- Implementing GNU-compatible numeric local labels, from 0 to 9

- Treating numeric constants beginning with 0 as octal
- Using semicolons as statement separators
- Using a single unbalanced quote for character constants — for example, `.byte 'a`.

EPPC Processor

The **EPPC Processor** settings panel (Figure 3.8) controls processor-dependent code-generation settings. Table 3.8 explains the elements of this panel.

Figure 3.8 EPPC Processor Settings Panel

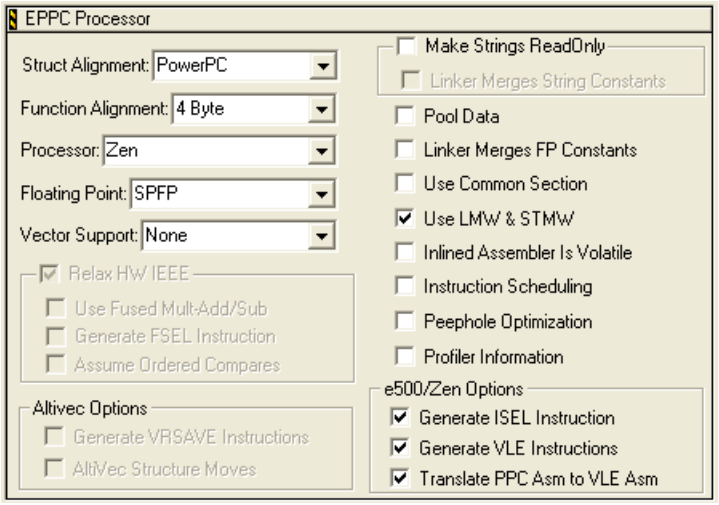


Table 3.8 EPPC Processor Settings Panel Elements

Element	Purpose	Comments
Struct Alignment list box	Specifies appropriate alignment for compatibility with Power Architecture EABI and third-party object code.	If you choose a different alignment value, your code may not work correctly.

Table 3.8 EPPC Processor Settings Panel Elements (*continued*)

Element	Purpose	Comments
Function Alignment list box	Specifies function alignment (bytes) to the width of multiple-instruction hardware fetches. Possible values range from 4 to 128. (Does not affect boards not capable of multiple-instruction fetches.)	The `st_other` field of `.symtab` (ELF) entries has been overloaded, so function dead-stripping does not interfere with this alignment. This may result in code that is incompatible with some third-party linkers.
Processor list box	Specifies the individual processor for which the system will tailor code. Specifying **Generic** results in code that runs on any Power Architecture processor.	Section Processor Selection, after this table, explains additional effects of this selection.
Floating Point list box	Specifies how the compiler handles floating-point operations in your code. Possible values are: • None • Software • Hardware • SPFP (single-precision floating-point) • DPFP (double-precision floating-point)	Also activates the **Relax HW IEEE** checkbox (and, in turn, its subordinate checkboxes). Hardware and DPFP selections do not pertain to MPC55xx/MPC56xx processors. Section Floating Point Operations, after this table, provides additional information.
Vector Support list box	Specifies generation of instructions for the target processor's type of vector execution. Possible values are: • None • Altivec • SPE • SPE Addl • SPE2	Selecting `Altivec` activates the checkboxes of the **Altivec Options** area. Section Vector Operations provides additional information.

Table 3.8 EPPC Processor Settings Panel Elements (*continued*)

Element	Purpose	Comments
Relax HW IEEE checkbox	Clear — Maintains strict IEEE floating-point requirements; deactivates subordinate checkboxes. Checked — Activates subordinate checkboxes; permits generation of faster code by ignoring the corresponding strict IEEE floating-point requirements.	This checkbox is active only if the **Floating Point** list box specifies Hardware.
Use Fused Mult-Add/Sub checkbox	Clear — Does *not* generate Power Architecture Fused Multi-Add/Sub instructions. Checked — Generates PowerPC Fused Multi-Add/Sub instructions, resulting in smaller, faster floating-point code.	This checkbox is active only if the **Relax HW IEEE** checkbox is active and checked. Calculations with this option are slightly more accurate, due to an extra rounding bit, so may lead to unexpected results.
Generate FSEL Instruction checkbox	Clear — Maintains standard FSEL instructions. Checked — For floating-point values `x` and `y`, lets the compiler optimize the pattern `x = (condition ? y : z)`, resulting in a faster-executing FSEL instruction.	This checkbox is active only if the **Relax HW IEEE** checkbox is active and checked. FSEL is not accurate for denormalized numbers, and may adversely affect unordered compares.
Assume Ordered Compares checkbox	Clear — Maintains strict IEEE floating-point compares: all compares against NAN except not-equal-to return FALSE. Checked — Lets the compiler ignore unordered-number issues with floating-point compares. This permits conversion of `if (a <= b)` to `if (a > b)`,	This checkbox is active only if the **Relax HW IEEE** checkbox is active and checked.

Table 3.8 EPPC Processor Settings Panel Elements (*continued*)

Element	Purpose	Comments
Generate VRSAVE Instructions checkbox	Clear — Does *not* generate VRSAVE instructions. Checked — Generates instructions to save/restore bit settings of the VRSAVE register, and related non-volatile vector register values.	This checkbox is active only if the **Vector Support** list box specifies `Altivec`. This option is *not* appropriate for MPC55xx/MPC56xx microcontrollers, which do not have Altivec vector execution units.
Altivec Structure Moves checkbox	Clear — Does *not* use Altivec instructions to copy structures. Checked — Uses Altivec instructions to copy structures.	This checkbox is active only if the **Vector Support** list box specifies `Altivec`. This option is *not* applicable for MPC55xx/MPC56xx microcontrollers, which do not have Altivec vector execution units.
Make Strings ReadOnly checkbox	Clear — Stores string constants in the ELF-file data section. Checked — Stores string constants in the read-only .rodata section. Also activates the subordinate checkbox Linker Merges String Constants.	Corresponds to pragma `readonly_strings`.
Linker Merges String Constants checkbox	Clear — Keeps individual the strings of each file. (This permits deadstripping of unused strings.) Checked — Compiler pools strings of a file.	This checkbox is active only if the **Make Strings ReadOnly** checkbox is checked.
Pool Data checkbox	Clear — Maintains default data organization, permitting stripping of unused data. Checked — Organizes some data of sections `.data`, `.bss`, and `.rodata` for faster program access.	This option affects only data defined in the current source file. This option is not compatible with tentative data: it warns that you must force tentative data into the common section.

Table 3.8 EPPC Processor Settings Panel Elements (*continued*)

Element	Purpose	Comments
Linker Merges FP Constants checkbox	Clear — Compiler does *not* name floating-point constants for automatic merging. Checked — Compiler names floating-point constants so that names contain constants.	This option lets the linker automatically merge floating-point constants.
Use Common Section checkbox	Clear — Two variables with the same name leads to a link error. Checked — Compiler places global uninitialized data in the common section — multiple variables with the same name share the same storage address if at least one is in the common section.	Clear is appropriate for development. But after you debug your program, change names of especially large variables to be the same, initialize them before use, and check this checkbox.
Use LMW & STMW checkbox	Clear — Compiler does *not* use `LMW` or `STMW` instructions; code executes faster, even if it is larger. Checked — Lets the compiler use single `Load-Multiple-Word` and `Store_Multiple_Word` Power Architecture instructions for register loads and stores. This leads to smaller (but slower executing) code.	`LMW` and `STMW` instructions are not compatible with little-endian code: for such code, the compiler ignores this checkbox. If a smaller function fits better in microcontroller cache lines, it is possible that the function using `LMW/STMW` executes faster than one using multiple `LWZ/STW` instructions. To see if this is the case, you may use pragmas `no_register_save_helpers` and `use_lmw_stmw`.

Table 3.8 EPPC Processor Settings Panel Elements (*continued*)

Element	Purpose	Comments
Inlined Assembler Is Volatile checkbox	Clear — Compiler does *not* treat `asm` blocks as if the volatile keyword were present. This permits optimization of `asm` blocks. Checked — Compiler treats all `asm` blocks (including inline `asm` blocks) as if the volatile keyword were present.	Checking this checkbox prevents optimization of `asm` blocks. To enable `asm` block optimization selectively, use the `.nonvolatile` directive.
Instruction Scheduling checkbox	Clear — Compiler does *not* perform this optimization. Checked — Optimizes scheduling of instructions for the processor that the **Processor** list box specifies.	This optimization changes instruction execution order, so can make source-level debugging difficult. You may find it helpful to clear this checkbox until most debugging is done.
Peephole Optimization checkbox	Clear — Compiler does *not* perform this optimization. Checked — Compiler performs small, local optimizations that can lead to reductions of multiple instructions into one, elimination of some compare instructions, and improvement of branch sequences.	Checking this checkbox corresponds to using pragma `peephole`.
Profiler Information checkbox	Clear — Does *not* generate profiler information. Checked — Generates special object code during runtime, to collect information for a code profiler.	Checking this checkbox corresponds to using pragma `profile`.

Table 3.8 EPPC Processor Settings Panel Elements (*continued*)

Element	Purpose	Comments
Generate ISEL Instruction checkbox	Clear — Does *not* generate ISEL instructions. Checked — Generates Integer Select (ISEL) instructions, which pertain only to MPC55xx/MPC56xx targets.	
Generate VLE instructions checkbox	Clear — Does *not* generate VLE instructions. Checked — Generates Variable-Length Encoding instructions.	
Translate PPC Asm to VLE Asm checkbox	Clear — Does *not* translate PPC Asm to VLE Asm. Checked — Translates PPC Asm to VLE Asm.	.

Processor Selection

Your selection in the **Processor** list box has significance in these areas:

- Instruction scheduling — If you check the **Instruction Scheduling** checkbox, the processor selection helps determine how the compiler makes scheduling optimizations.
- Preprocessor symbol generation — The system defines a preprocessor symbol based on your target processor. It conforms to:

  `#define __PPC`*`number`*`__ 1`

  where *number* is the four-digit processor number: For the 5561 processor, for instance, the symbol is `__PPC5561__`. If you specify `Generic`, the symbol is `__PPCGENERIC__`.
- Floating-point support — The **Floating Point** list box lets you specify the **None**, **Software**, or **Hardware** value, regardless of the target processor you specify, even if this processor lacks a floating-point unit. If the processor does not support floating-point exception handling, however, you should select **None** or **Software**. (Not selecting **Hardware** deactivates the **Use FusedMult-Add/Sub** checkbox.

Floating Point Operations

Your selection in the **Floating Point** list box defines how the compiler handles floating-point operations. You must also include in your project the runtime library that corresponds to your selection. For example, if you select the **None** option, you must also include the library `Runtime.PPCEABI.N.a`.

Your list-box options are:

- None — Prevents floating-point operations.
- Software — Emulates floating-point operations in software.

  (This floating-point emulation generates calls defined in the C runtime library, so you must include the appropriate C runtime file in your project. Otherwise, enabling software emulation causes link errors.)
- Hardware — Performs hardware floating-point operations. The e200z cores does not implement the floating-point instructions as they are defined in Book E. The Hardware option does not apply to MPC55xx/MPC56xx processors.

  Also activates the **Relax HW IEEE** checkbox (checking this checkbox activates its subordinate checkboxes **Use Fused Mult-Add/Sub**, **Generate FESL Instruction**, and **Assume Ordered Compares**).

  (Do not select the Hardware option if your target processor lacks a Book E hardware floating-point unit.)
- SPFP — Single-precision Floating-Point Performs floating-point operations by the e200 core's.
- DPFP — Double-precision floating-point performs operations by software routines. (A runtime library containing these routines have to be included in the new project). Use hardware for both single float and double float arithmetic. This is only for processors that support hardware DPFP instructions.
- SPFP only — Built-in types doubles and long doubles will be treated as if they are only 4 bytes in size. This means that double is considered the same type as float that they are both 4 bytes and have the same encoding and precision for single floating point numbers. This option is only supported for e200 (Zen or VLE) and e500v1 processors that support the SPFP APU.

The DPFP option does not apply to MPC55xx/MPC56xx processors.

Vector Operations

Your selection in the **Vector Support** list box specifies generation of instructions for the target processor's type of vector execution.

The list-box options are:

- None — Prevents vector support.

- Altivec — Enables use of vector data types for writing AltiVec-specific code.

  This option enables the **Altivec Options** panel in the **EPPC Processor Settings** panel. The **Altivec Options** panel has the following checkboxes:

  – Generate VRSAVE Instructions

  – Altivec Structure Moves

NOTE These options are not appropriate for MPC55xx/MPC56xx microcontrollers, which do not have Altivec vector execution units.

- SPE — Enables the SPE vector support. This option needs to be enabled when the floating point is set to SPFP or DPFP as both these options require support from the SPE vector unit. If the option is not turned on, the compiler generates a warning and automatically enables the SPE vector generation.
- SPE Addl — Enables the additional SPE vector support. The e200 z3 and z6 cores support eight additional SPE-fused multiply-add and multiply-subtract instructions. This option tells the compiler to generate the additional SPE instructions, when appropriate, for more optimized codes and also turns on the SPE option.
- SPE2 —Enables the SPE2 vector support. This option is supported for e200 z7 core only. When the SPE2 option is selected, the `-pragma fp_contract off` directive is used to disable the SPE additional-fused multiply-add instruction generation. By turning off this optimization, the floating point accuracy is maintained.

EPPC Disassembler

Use the **EPPC Disassembler** settings panel (Figure 3.9) to control display of disassembler information. (To see this information, select **Project > Disassemble** from the CodeWarrior main menu bar.) Table 3.9 explains the elements of this panel.

Figure 3.9 EPPC Disassembler Settings Panel

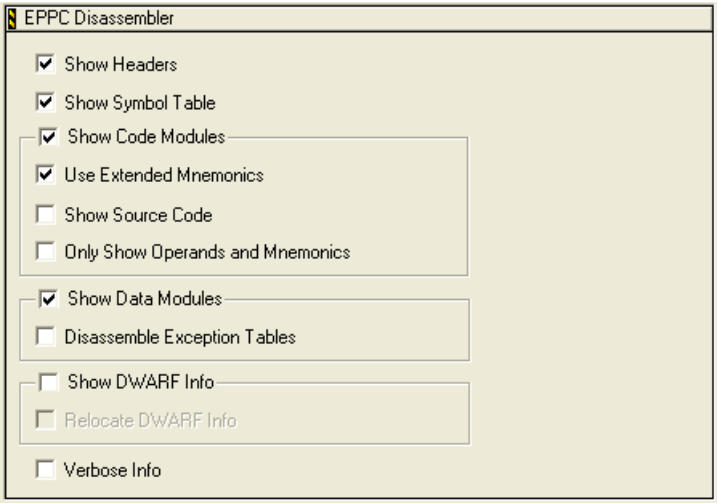


Table 3.9 EPPC Disassembler Settings Panel Elements

Element	Purpose	Comments
Show Headers checkbox	Clear — Disassembled output does *not* include ELF-header information. Checked — Assembled file lists any ELF header information in the disassembled output.	
Show Symbol Table checkbox	Clear — Disassembler does *not* list the symbol table. Checked — Disassembler lists the symbol table for the disassembled module.	

Table 3.9 EPPC Disassembler Settings Panel Elements (*continued*)

Element	Purpose	Comments
Show Code Modules checkbox	Clear — Does *not* provide ELF code sections; disables subordinate checkboxes. Checked — Provides ELF code sections in module disassembled output. Activates subordinate checkboxes **Use Extended Mnemonics**, **Show Source Code**, and **Only Show Operands and Mnemonics**.	
Use Extended Mnemonics checkbox	Clear — Disassembler does *not* list extended mnemonics. Checked — Disassembler lists extended mnemonics for each instruction.	This checkbox is active only if the **Show Code Modules** checkbox is checked.
Show Source Code checkbox	Clear — Disassembler does *not* show source code. Checked — Disassembler does show source code.	This checkbox is active only if the **Show Code Modules** checkbox is checked.
Only Show Operands and Mnemonics checkbox	Clear — Lists offsets for any functions in the disassembled module. Checked — Does *not* list offsets.	This checkbox is active only if the **Show Code Modules** checkbox is checked.
Show Data Modules checkbox	Clear — Does *not* provide ELF data sections. Checked — Disassembler provides ELF data sections, such as `.rodata` and `.bss`, in the disassembled module output. Activates subordinate checkbox **Disassemble Exception Table**.	

Table 3.9 EPPC Disassembler Settings Panel Elements (*continued*)

Element	Purpose	Comments
Disassemble Exception Tables checkbox	Clear — Does *not* provide C++ exception tables. Checked — Disassembler provides C++ exception tables in the disassembled module output.	This checkbox is active only if the **Show Data Modules** checkbox is checked.
Show DWARF Info checkbox	Clear — Does *not* include DWARF symbol information. Checked — Disassembler includes DWARF symbol information in disassembled output. Activates subordinate checkbox **Relocate DWARF Info**.	
Relocate DWARF Info checkbox	Clear — Does *not* relocate addresses. Checked — Displays relocated addresses inside debug sections.	Pertains to DWARF 1 debug information. This checkbox is active only if the **Show DWARF Info** checkbox is checked.
Verbose Info checkbox	Clear — Does not display additional information. Checked — Displays additional ELF-file information, such as descriptive constants and numeric equivalents in the `.symtab` section. Shows `.line`, `.debug`, `.extab`, and `.extabindex` sections with an unstructured hexadecimal dump.	

EPPC Linker

Use the **EPPC Linker** settings panel (Figure 3.10) to control settings related to linking your object code into executable, library, or other final form. Table 3.10 explains the elements of this panel.

Figure 3.10 EPPC Linker Settings Panel

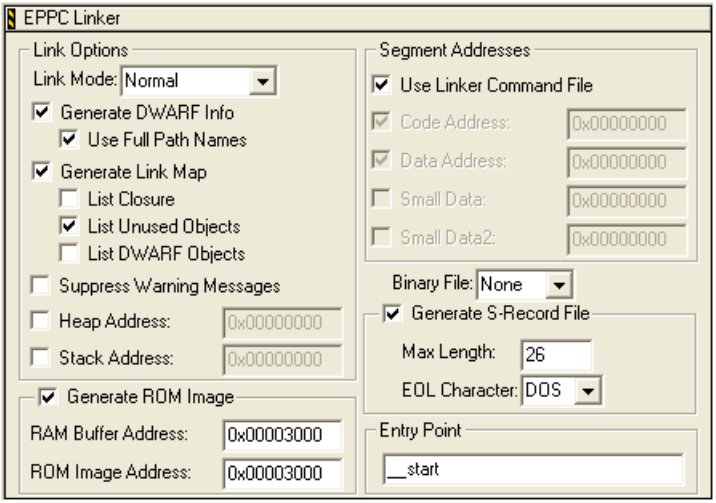


Table 3.10 EPPC Linker Settings Panel Elements

Element	Purpose	Comments
Link Mode list box	Specifies how much memory the linker uses to write output to the hard disk. Possible values are: • `Use Less RAM` — writes output file directly to disk, without using a buffer. • `Normal` — Writes to a 512-byte buffer, then writes the buffer to disk. • `Use More RAM` — Writes each segment to its own buffer, then flushes all buffers to the disk.	Linking requires enough RAM space for all input files and numerous housekeeping structures. `Normal` is the best choice for most projects; `Use More RAM` is appropriate for small projects.

Table 3.10 EPPC Linker Settings Panel Elements (*continued*)

Element	Purpose	Comments
Generate DWARF Info checkbox	Clear — Does not generate debugging information; deactivates the subordinate checkbox. Checked — Generates debugging information in the linked ELF file. Activates subordinate checkbox **Use Full Path Names**.	
Use Full Path Names checkbox	Clear — Linker uses only file names. Checked — Linker includes path names in the linked ELF file.	This checkbox is active only if the **Generate DWARF Info** checkbox is checked. Clear this checkbox if you build/debug on a different computer or platform, to help the debugger find your source code.
Generate Link Map checkbox	Clear — Does *not* generate a map file. Checked — Linker generates a link map — showing every object/function definition and address, memory map of sections, and values of linker-generated symbols. Activates subordinate checkboxes.	If you used a non-CodeWarrior compiler to build the relocatable file, the map file also lists unused but unstripped symbols. Map files have the extension `.MAP`.
List Closure checkbox	Clear — Map does *not* list functions that the program starting point calls. Checked — Map file lists all functions that the program starting point calls.	This checkbox is active only if the **Generate Link Map** checkbox is checked.
List Unused Objects checkbox	Clear — Map does *not* list unused objects. Checked — Map lists unused objects; useful for revealing that objects you expected to be used are not.	This checkbox is active only if the **Generate Link Map** checkbox is checked.

Table 3.10 EPPC Linker Settings Panel Elements (*continued*)

Element	Purpose	Comments
List DWARF Objects checkbox	Clear — Map does *not* list DWARF debugging objects. Checked — Map lists all DWARF debugging objects in section area.	This checkbox is active only if the **Generate Link Map checkbox** is checked.
Suppress Warning Messages checkbox	Clear — Linker displays warnings in the CodeWarrior message window. Checked — Linker does *not* display warnings in the CodeWarrior message window.	
Heap Address checkbox	Clear — Makes the top of the heap equal the bottom of the stack. Checked — Specifies memory location for program heap. Activates the associated text box, which you use to enter the RAM address of the *bottom* of the heap.	Subsection Heap Information, after this table provides additional heap guidance.
Stack Address checkbox	Clear — Linker uses default stack address `0x003DFFF0`. Checked — Specifies memory location for program stack. Activates the associated text box, which you use to enter the RAM address for the *top* of the stack.	Subsection Stack Information, after this table provides additional stack guidance.
Generate ROM Image checkbox	Clear — Does *not* generate ROM image; deactivates subordinate checkboxes. Checked — Linker creates a ROM image. Activates subordinate checkboxes RAM Buffer Address and ROM Image Address.	

Table 3.10 EPPC Linker Settings Panel Elements (*continued*)

Element	Purpose	Comments
RAM Buffer Address checkbox	Clear — Does *not* let you specify a RAM buffer address. Checked — Activates the text box, letting you specify the address of a RAM buffer for a flash programmer to use. Many other flash programmers use the specified, separate, buffer to load all binary segments into consecutive addresses in flash ROM. At runtime, however, the system loads these segments into addresses that the linker command file or the **Code Address** text box specify.	This checkbox is active only if the **Generate ROM Image** checkbox is checked. For the CodeWarrior flash programmer, the RAM buffer address and the ROM image address must be the same. The linker generates symbols for ROM and execution addresses. file *`installDir`*`\PowerPC_EABI_Support\Runtime\Include\__ppc_eabi_linker.h` provides more information about such symbols.
ROM Image Address checkbox	Clear — Does *not* let you specify a destination address. Checked — Activates the text box, letting you specify flash ROM destination address for your binary.	This checkbox is active only if the **Generate ROM Image** checkbox is checked. For the CodeWarrior flash programmer, the ROM image address and the RAM buffer address must be the same.
Use Linker Command File checkbox	Clear — Lets you specify addresses via other checkboxes of the **Segment Addresses** area; ignores any linker command file. Checked — Tells the linker to find segment addresses in the linker command file.	Must be clear if you check any other Segment Addresses checkboxes. If you check this checkbox but the linker command file does not specify segment addresses, the system issues an error message.

Table 3.10 EPPC Linker Settings Panel Elements (*continued*)

Element	Purpose	Comments
Code Address checkbox	Clear — Accepts linker command file address specification. Checked — Activates the corresponding text box, letting you specify the hexadecimal address for executable code.	Must be clear if the **Use Linker Command File** checkbox is checked. If both this and the **Use Linker Command File** checkboxes are clear, uses default address `0x00010000`, which may not be suitable for boards with a small amount of RAM. (Stationery projects include examples with better addresses for such boards,)
Data Address checkbox	Clear — Accepts linker command file address specification. Checked — Activates the corresponding text box, letting you specify the hexadecimal address for global data.	Must be clear if the **Use Linker Command File** checkbox is checked. If both this and the **Use Linker Command File** checkboxes are clear, the linker uses the address after sections `.text`, `.rodata`, `extab`, and `extabindex`,)
Small Data checkbox	Clear — Accepts linker command file address specification. Checked — Activates the corresponding text box, letting you specify the hexadecimal RAM address for the first small data section. This address must not conflict with the target-hardware memory map; target hardware must support this address.	Must be clear if the **Use Linker Command File** checkbox is checked. If both this and the **Use Linker Command File** checkboxes are clear, the linker places the first small data section immediately after the `.data` section.

Table 3.10 EPPC Linker Settings Panel Elements (*continued*)

Element	Purpose	Comments
Small Data2 checkbox	Clear — Accepts linker command file address specification. Checked — Activates the corresponding text box, letting you specify the hexadecimal RAM address for the second small data section. This address must not conflict with the target-hardware memory map; target hardware must support this address.	Must be clear if the **Use Linker Command File** checkbox is checked. If both this and the **Use Linker Command File** checkboxes are clear, the linker places the second small data section immediately after the `.sbss` section.
Binary File list box	Creates binary file(s). It has the following choices: • None — No binary file • One — One binary file • Multiple — Multiple binary files	Default option is **None**. No binary file will be created unless explicitly **One** or **Multiple** option is selected from the **Binary File** list box.
Generate S-Record File checkbox	Clear — Does *not* generate an S-record file. Checked — Generates an S3 S-record file, based on the application object image. Activates subordinate elements.	The name extension of the S-record file is `.mot`.
Sort S-Record checkbox	Clear — Does *not* sort S-record files. Checked — Sorts generated S-record files in ascending address order.	This checkbox is active only if the **Generate S-Record File** checkbox is checked.
Max Length text box	Specifies maximum S-record length (256 bytes or fewer) for the system. (For a non-CodeWarrior tool, you may need to reduce this value.)	This text box is active only if the **Generate S-Record File** checkbox is checked.

Table 3.10 EPPC Linker Settings Panel Elements (*continued*)

Element	Purpose	Comments
EOL Character list box	Specifies the end-of-line character for the S-record file: • DOS — `<cr> <lf>` • Unix — `<lf>` • Mac — `<cr>`	This list box is active only if the **Generate S-Record File** checkbox is checked.
Entry Point text box	Specifies the program starting point — the function that the linker uses first when you launch the program.	This default function (in file `__start.c`) is bootstrap/ glue code that sets up the EABI environment, then calls function `main()`.

Heap Information

Your program uses a heap if it calls `malloc` or `new`. If you use the Main Standard Libraries (MSL), your program may use a heap implicitly. However MSL allocation routines do not require a heap below the stack.

If you do specify a heap address, check the **Heap Address** checkbox, then enter a hexadecimal address in the text box. This address is the *bottom* of the heap; if necessary, the system aligns it up to the nearest 8-byte boundary. The top of the heap is Heap Size kilobytes above the Heap Address (the **Heap Size** text box of the EPPC Target settings panel specifies the Heap Size value). The possible address values depend on your target hardware platform and how memory is mapped. The heap must reside in RAM; the heap address may be any place in RAM that does not overlap other sections. The MSL also permit multiple memory pools, which can increase the total size of the heap.

If you clear the checkbox, the top of the heap equals the bottom of the stack:

```
_stack_end = _stack_addr - (stack_size * 1024);
_heap_end  = _stack_end;
_heap_addr = _heap_end - (heap_size * 1024);
```

You can clear the **Heap Address** checkbox if your code does not make use of a heap.

NOTE If there is not enough free space available in your program, `malloc` returns zero. If you do not call `malloc` or `new`, consider setting Heap Size (k) to 0 to maximize the memory available for code, data, and the stack.

Stack Information

The **Stack Address** checkbox and text box let you specify the top of the stack. The address you enter must be in RAM, and in hexadecimal notation. The stack grows down from that address; its size is the value of the **Stack Size** text box (of the **EPPC Target settings** panel). If necessary, the system aligns your address value up to the nearest 16-byte boundary. The possible address values depend on your target hardware platform and how the memory is mapped.

NOTE An alternative way to specify the stack address is by entering a value for the symbol `_stack_addr` in a linker command file.

If you do not specify a stack address, the linker uses `0x003DFFF0`. But this default value may not be suitable for boards that have only a small amount of RAM. If you have such a board, see the stationery projects for examples that have suitable addresses.

NOTE As the stack grows downward, it is common to place the stack as high as possible. CodeWarrior TRK, for example, puts its data in high memory, placing the stack at default address `0x003DFFF0`. CodeWarrior TRK also uses memory from `0x00000100` through `0x00002000` for exception vectors

EPPC Linker Optimizations

Use the **EPPC Linker Optimizations** settings panel (Figure 3.11) to specify a batch file that CodeWarrior build tools should run before linking your project. Table 3.11 explains the elements of this panel.

Figure 3.11 EPPC Linker Optimizations Settings Panel

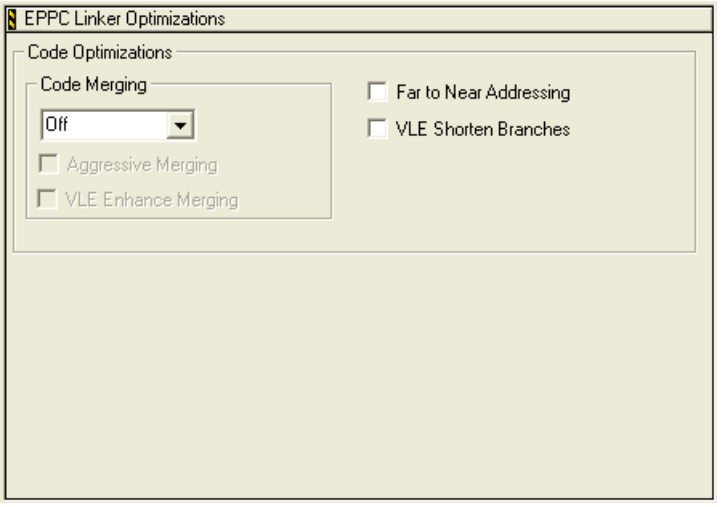


Table 3.11 EPPC Linker Optimizations Settings Panel Elements

Element	Purpose	Comments
Code Merging list box	Controls merging optimization. Selections are: • `Off` • `Safe Functions` • `All Functions`	Off deactivates subordinate checkboxes A**ggressive Merging** and **VLE Enhance Merging**. Other selections activate these checkboxes.
Aggressive Merging checkbox	Clear — Does *not* implement aggressive merging. Checked — Implements aggressive merging.	This checkbox is active only if the **Code Merging** list box specifies `Safe Functions` or `All Functions`.
VLE Enhance Merging checkbox	Clear — Does *not* implement VLE enhance merging. Checked — Implements VLE enhance merging.	This checkbox is active only if the **Code Merging** list box specifies Safe `Functions` or `All Functions`.

Table 3.11 EPPC Linker Optimizations Settings Panel Elements (*continued*)

Element	Purpose	Comments
Far to Near Addressing checkbox	Clear — Does *not* implement far-to-near addressing optimization. Checked — Implements far-to-near addressing optimization.	
VLE Shorten Branches checkbox	Clear — Does *not* implement VLE shorten branches optimization. Checked — Implements VLE shorten branches optimization.	.

PC-lint Settings Panels

PC-lint is a third-party software development tool that checks C/C++ source code for bugs, inconsistencies, non-portable constructs, redundant code, and additional problems. CodeWarrior Development Studio for MPC55xx/MPC56xx Microcontrollers include target settings panels and plug-ins that let you configure and use PC-lint from within the CodeWarrior IDE.

However, this CodeWarrior product does *not* include the PC-lint software. You must obtain and install a copy of PC-lint before you can use it with the CodeWarrior IDE. Among other places, PC-lint is available from its developers, Gimpel Software: http://www.gimpel.com.

NOTE To use the default CodeWarrior PC-lint configuration as is, install PC-lint in the `\Lint` subdirectory of the CodeWarrior installation directory. Alternatively, you can install PC-lint anywhere and then adjust the CodeWarrior configuration to match.

Once you have installed PC-lint, you can configure any build target of any CodeWarrior project to use this software. To do this, follow these steps:

1. Open a project.
2. Select the build target with which you will use PC-lint.
3. Bring up the **Target Settings** window for this build target.
4. Display the **Target Settings** panel in the **Target Settings** window.

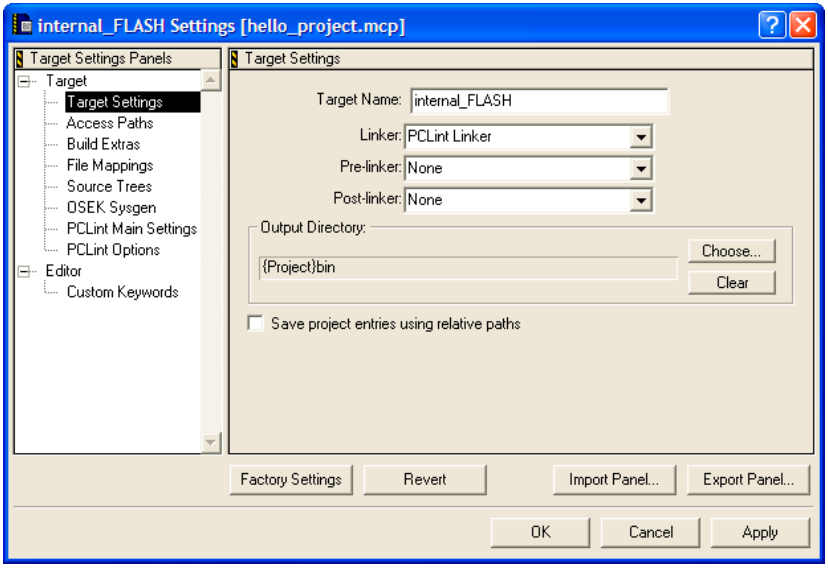
5. From the **Linker** list box, select `PCLint Linker`, as shown in Figure 3.12.

   New panel names appear in the **Target Settings Panels** pane: **PCLint Main Settings** and **PCLint Options**. (Names of panels that pertain to ELF generation disappear from this pane.)

6. Use these new panels to specify PC-lint configuration options appropriate for your build target.

The sections that follow document each of the PC-lint target settings panels.

Figure 3.12 Selecting PCLint Linker



PCLint Main Settings

Use the **PCLint Main Settings** panel (Figure 3.13) to provide the path to the PC-lint executable and to define the compiler-option/prefix files. Table 3.12 documents the elements of this panel.

Figure 3.13 PCLint Main Settings Panel

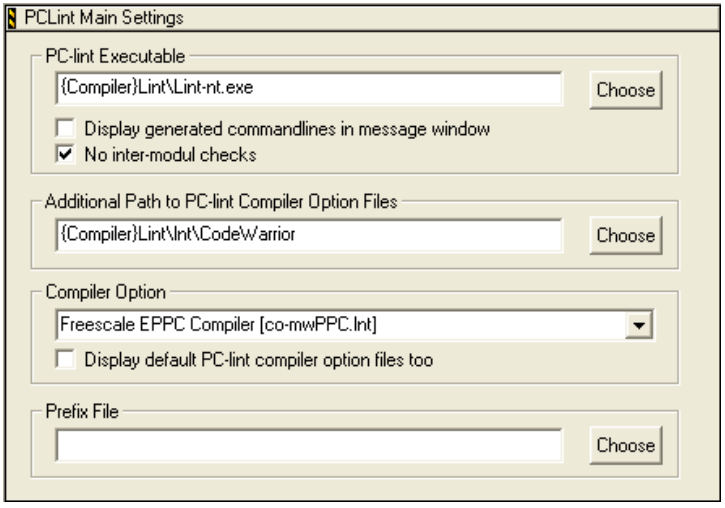


Table 3.12 PCLint Main Settings Panel Elements

Element	Purpose	Comments
PC-lint Executable text box	Specifies the PC-lint executable file. If you did not install PC-lint on the default path, enter the actual path and filename.	Click **Choose** — a subordinate dialog box appears, that you can use to specify the file. Click **OK** to return to this settings panel, placing the specified path name in the text box.
Display generated commandlines ... checkbox	Clear — There is not display of the command line the IDE passes to PC-lint. Checked — **IDE Errors and Warnings** window displays the command line the IDE passes to PC-lint.	

Table 3.12 PCLint Main Settings Panel Elements (*continued*)

Element	Purpose	Comments
No inter-module checks checkbox	Clear — Linker uses only file names. Checked — Linker includes path names in the linked ELF file.	This checkbox is active only if the **Generate DWARF Info** checkbox is checked. Clear this checkbox if you build/debug on a different computer or platform, to help the debugger find your source code.
Additional Path to ... text box	Specifies the path to the PC-lint option files. To configure a build target to use an additional option file, enter the path to the directory. (You can leave this text box empty.)	Click **Choose** — a subordinate dialog box appears that you can use to specify the directory. Click **OK** to return to this settings panel, placing the specified path name in the text box.
Compiler Option list box	Lists compiler option files of the directory that the Additional Pat to ... text box specifies. Select the appropriate file.	This text box is active only if there are option files in the specified directory.
Display default ... files too checkbox	Clear — Does *not* include default `.lnt` files in the configuration. Checked — Includes default `.lnt` files to the configuration, along with alternate `.lnt` files that previous text boxes specify.	
Prefix File text box	Specifies an optional prefix file to pass to PC-lint: enter the path and filename. (You can leave this text box empty.)	Click **Choose** — a subordinate dialog box appears that you can use to specify the file. Click **OK** to return to this settings panel, placing the specified path name in the text box.

PCLint Options

Use the **PCLint Options** settings panel (Figure 3.14) to define the syntax rules, environment options, and other settings PC-lint uses to validate source files and perform additional error checking. Table 3.13 documents the elements of this panel.

Figure 3.14 PCLint Options Settings Panel

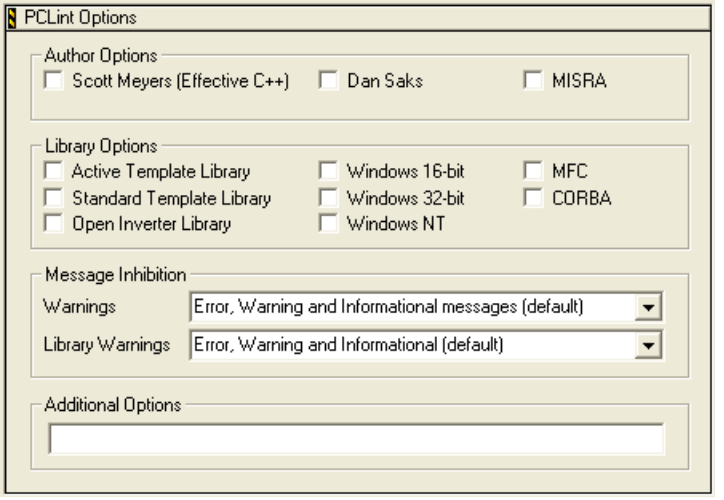


Table 3.13 PCLint Options Settings Panel Elements

Element	Purpose	Comments
Scott Meyers (Effective C++) checkbox	Clear — PC-lint does *not* verify adherence to Effective C++ syntax. Checked — PC-lint verifies that your code conforms to Effective C++ syntax rules.	(You may check none, any one, any two, or all three checkboxes of this group.)
Dan Saks checkbox	Clear — PC-lint does *not* verify adherence to Dan Saks recommendations. Checked — PC-lint verifies that your code adheres to syntax rules Dan Saks recommends.	(You may check none, any one, any two, or all three checkboxes of this group.)

Table 3.13 PCLint Options Settings Panel Elements (*continued*)

Element	Purpose	Comments
MISRA checkbox	Clear — PC-lint does *not* verify conformity with MISRA guidelines. Checked — PC-lint verifies that your code adheres to C guidelines of the Motor Industry Software Reliability Association.	(You may check none, any one, any two, or all three checkboxes of this group.)
Active Template Library checkbox	Clear — PC-lint does *not* validate Active X Template Library (ATL) code. Checked — PC-lint validates your ATL code.	
Standard Template Library checkbox	Clear — PC-lint does *not* validate Standard Template Library (STL) code. Checked — PC-lint validates your STL code.	
Open Inverter Library checkbox	Clear — PC-lint does *not* validate open inverter library code. Checked — PC-lint validates your open inverter library code.	
Windows 16-bit checkbox	Clear — PC-lint does *not* validate 16-bit **API** calls. Checked — PC-lint validates your 16-bit Windows API calls.	
Windows 32-bit checkbox	Clear — PC-lint does *not* validate 32-bit API calls. Checked — PC-lint validates your 32-bit Windows API calls.	
Windows NT checkbox	Clear — PC-lint does *not* validate NT API calls. Checked — PC-lint validates your Windows NT API calls.	

Table 3.13 PCLint Options Settings Panel Elements (*continued*)

Element	Purpose	Comments
MFC checkbox	Clear — PC-lint does *not* validate Microsoft Foundation Classes (MFC) code. Checked — PC-lint validates your MFC code.	
CORBA checkbox	Clear — PC-lint does *not* validate Common Object Request Broker Architecture (CORBA) code. Checked — PC-lint validates your COBRA code.	
Warnings list box	Specifies the kinds of messages PC-lint displays.	
Library Warnings list box	Specifies the kinds of messages PC-lint displays for libraries.	
Additional Options text box	Specifies command-line switches for the IDE to pass to PC-lint.	PC-lint manuals list the possible switches.

A

Lauterbach Debugger Adjustments

One of the debuggers you can use with your MPC55xx/MPC56xx projects is the Lauterbach TRACE32 system. This appendix explains the changes you must make to accommodate the Lauterbach debugger.

In this appendix:

NOTE If you use a debugger from a different manufacturer, that debugger's user documentation may provide corresponding guidance. Another possible source of such guidance is the manufacturer's customer support department.

Modifying Configuration Files

You need a host-system configuration file to set up the target system to work with the Lauterbach debugger. The examples of this section show the settings for a simple application.

For more complex applications, refer to the Lauterbach documentation. For detailed explanations of embedded Power Architecture registers, see the "Related Documentation" section.

Power Architecture Configuration File

The usual filename extension for the configuration file is `.cmm`. Listing A.1 shows a configuration file for a regular Power Architecture application. When you specify an executable file to be loaded, you must provide its location as a path relative to that of the configuration file.

Listing A.1 Example .cmm File for Lauterbach Debugger

```
SYStem.RESet
SYStem.CPU 55XX
SYStem.BdmClock 4.MHz
SYStem.UP

; initialize internal SRAM
Data.Set EA:0x40000000--0x4003FFFF %quad 0

; set MMU TLB1: Map SRAM (A:0x40000000) to 0x00000000
MMU.TLB1.SET 1 0x80000400 0x00000000 0x4000003f

Data.Load.ELF bin/ppc_app.elf  /GlobTypes

Data.List
```

The first few commands make microcontroller-specific settings, such as the target's microcontroller type and the debug-interface-to-target transfer clock.

The command

```
SYStem.UP
```

restarts the microcontroller, with debug mode enabled.

The next part of the configuration file sets up target-specific items. The command initializes SRAM with zeros, setting the address range as the interval between two physical addresses:

```
Data.Set EA:0x40000000-0x4003FFFF %quad 0
```

The next command sets up the MMU:

```
MMU.TLB1.SET 1 0x80000400 0x00000000 0x4000003f
```

This command initializes the first translation lookaside buffer (TLB). The first parameter is the index of the TLB entry being set up (in this case, the first). The other parameters are the MAS1, MAS2 and MAS3 (MMU Assist) registers:

- MAS1: sets the VALID bit, marking the TLB entry 1 as valid
- MAS1: sets the TSIZE field to 4, page size is set to 256 kilobytes
- MAS2: sets the effective page number to 0, the start of the virtual address space for the application
- MAS3: sets the real page number to 0x40000000, the start of the real address for the application
- MAS3: sets PERMIS bits to 0x3f to enable all permissions

This command loads the executable image relative to the directory on the host where the configuration file resides:

```
Data.Load.ELF bin/ppc_app.elf  /GlobTypes
```

The `GlobTypes` parameter must be set when the debug information is shared across different modules (such as application and runtime files that reside in different directories).

VLE Configuration File

If you use the Lauterbach TRACE32 debugger to debug a variable length encoded (VLE) Power Architecture application, you need a configuration file similar to that of Listing A.1. However, the MAS2 value must set the VLE flag (bit 58). Listing A.2 is an example.

Listing A.2 Example VLE .cmm File for Lauterbach Debugger

```
SYStem.RESet
SYStem.CPU 5534
SYStem.BdmClock 4.MHz
SYStem.UP

; initialize internal SRAM
Data.Set EA:0x40000000--0x4003FFFF %quad 0

; set MMU TLB1: Map SRAM (A:0x40000000) to 0x00000000
MMU.TLB1.SET 1 0x80000400 0x00000020 0x4000003f

Data.Load.ELF bin/vle_app.elf /GlobTypes

Data.List
```

Mixed Configuration File

Listing A.3 is an example configuration file for Lauterbach debugging of an application that uses both VLE and regular Power Architecture instruction encoding.

Note the two entries for the first TLB:

- One to specify a 4-kilobyte page for regularly encoded Power Architecture instructions,
- A second to specify a 4-kilobyte page for VLE instructions.

Also note that the Power Architecture and VLE code must reside in different pages. You may want to use a linker control file to specify this code layout in the executable image.

Listing A.3 Example Mixed Configuration File

```
SYStem.RESet
SYStem.CPU 5534
SYStem.BdmClock 4.MHz
SYStem.UP

; initialize internal SRAM
Data.Set EA:0x40000000--0x4000FFFF %quad 0

;set up MMU
; TLB1, 0x40000000--0x40000fff
MMU.TLB1.Set 1 0xC0000100 0x40000000 0x4000003F

; TLB2, 0x40001000--0x40001fff
MMU.TLB1.Set 2 0xC0000100 0x40001020 0x4000103F

Data.LOAD.ELF bin/ppc_vle_app.elf /GlobTypes

Data.List
```

Connecting the Hardware

Follow these steps to connect the Lauterbach TRACE32 Debugger hardware to your host and target systems:

1. Make sure the target system power is off.
2. Disconnect the debug cable from the target system.
3. Connect the Lauterbach TRACE32 hardware and the debug cable to the host system.
4. On the host system, start the TRACE32 software.
5. Reconnect the debug cable to the target system.
6. Switch on the target system power.

To disconnect the Lauterbach hardware, follow these steps:

1. Switch off the target power.
2. Disconnect the debug cable from the target system.

(For more details, refer to Lauterbach documentation.)

B

P&E Debugger Adjustments

One of the debuggers you can use with your MPC55xx/MPC56xx projects is the P&E ICDPPCNEXUS™ debugger. This appendix explains how to set up the P&E debugger to work with projects built with the CodeWarrior IDE.

In this appendix:

NOTE If you use a debugger from a different manufacturer, that debugger's user documentation may provide corresponding guidance. Another possible source of such guidance is the manufacturer's customer support department.

Modifying Configuration Files

You need a host-system configuration file to set up the target system to work with the P&E debugger. The examples in this section show the settings required for a simple application.

For more complex applications, refer to the P&E documentation. For a detailed documentation of Power Architecture registers, see the "Related Documentation" section.

Power Architecture Configuration File

The usual filename extension for the configuration or macro file is `.mac`. When you specify an executable file to be loaded, you must provide its location as a path relative to that of the configuration file. For more information about a regular Power Architecture configuration file, see the `MPC5516_booke.mac` file, located in the *`InstallDir`*`\pemicro` directory.

The `MPC5516_booke.mac` file configures the memory management unit (MMU) for Periph B modules, internal flash memory, external memory, and internal SRAM. For example, Listing B.1 shows the commands required to set up the third translation lookaside buffer (TLB) entry for internal SRAM.

Listing B.1 .mac file for P&E Debugger

```
REM Set up MMU for Internal SRAM
REM Base address = $4000_0000
REM TLB3, 256 KByte Memory Space, Not Guarded, Don't Cache, All Access,
VLE
spr 624t $10030000 ; MAS0
spr 625t $C0000400 ; MAS1
spr 626t $40000008 ; MAS2
spr 627t $4000003F ; MAS3
execute_opcode $7C0007A4 ; tlbwe
```

VLE Configuration File

To use the P&E ICDPPCNEXUS debugger to debug a variable length encoding (VLE) Power Architecture application, you must configure the `MPC5516_vle.mac` file located in the *`InstallDir`*`\pemicro` directory.

NOTE The MAS2 value must set the VLE flag (bit 58).

Table B.1 Macro Files for Various Targets

Microcontroller	Macro File	Notes
mpc551x	• mpc5516_booke.mac • mpc5516_vle.mac	Only the p0 core can run in BookE mode. Core(s) running in VLE mode.
mpc553x mpc555x mpc556x	• mpc5500_booke.mac • mpc5500_vle.mac	Microcontrollers running in BookE mode. Microcontrollers running in VLE mode
mpc560xB mpc560xP mpc560xS mpc560xE	• mpc5600_z0h_vle.mac	These Microcontrollers are VLE only.
mpc563xm	• mpc5633m_booke.mac • mpc5633m_vle.mac	Core(s) running in BookE mode. Core(s) running in VLE mode.

Table B.1 Macro Files for Various Targets (*continued*)

Microcontroller	Macro File	Notes
mpc5668	• mpc5668_booke.mac • mpc5668_vle.mac	Only the p0 core can run in BookE mode Core(s) running in VLE mode
mpc5674F	• mpc5674F_booke.mac • mpc5674F_vle.mac	Core(s) running in BookE mode Core(s) running in VLE mode
mpc5643L	• mpc5643L_booke.mac • mpc5643L_vle.mac	Core(s) running in BookE mode Core(s) running in VLE mode
mpc564xA	• mpc564xA_booke.mac • mpc564xA_vle.mac	Core(s) running in BookE mode Core(s) running in VLE mode
mpc5645S	• mpc5645S_booke.mac • mpc5645S_vle.mac	Core(s) running in BookE mode Core(s) running in VLE mode
mpc567xK	• mpc567xK_booke.mac • mpc567xK_vle.mac	Core(s) running in BookE mode Core(s) running in VLE mode
mpc564xB / mpc564xC	• mpc564xB_booke.mac • mpc564xB_vle.mac	Core(s) running in BookE mode Core(s) running in VLE mode
mpc5676R	• mpc5676R_booke.mac • mpc5676R_vle.mac	Core(s) running in BookE mode Core(s) running in VLE mode

Target Settings for the P&E Debugger

If you are using a P&E debugger, use the **Build Extras** setting panel (Figure B.1) to specify debugger settings. To open this window, select **Edit > *Target* Settings** from the main-window menu bar. Table B.2 explains the elements that you need to specify for using an external debugger. For more information on the **Build Extras** target setting panel, see the *CodeWarrior IDE User's Guide*.

Figure B.1 Build Extras Setting Panel

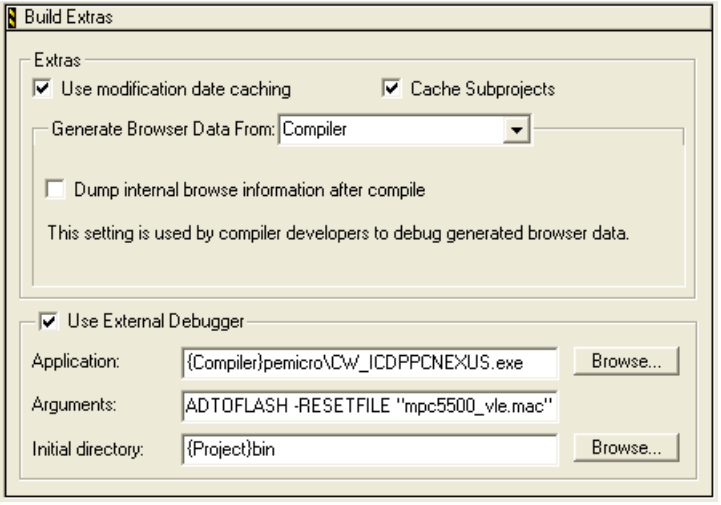


Table B.2 Build Extras Settings Panel — Elements that Pertain to the P&E Debugger

Element	Purpose	Comment
Application	Specifies the external debugger application. In this case, the P&E debugger.	Click **Browse** to select the P&E debugger application (provide the full path to the P&E debugger executable file). Alternatively, enter the path to the P&E debugger.
Arguments	Specifies the command-line arguments to pass to the P&E debugger.	Type any command-line arguments to pass to the P&E debugger when the IDE transfers control. For more information, see "Command-Line Arguments".
Initial Directory	Specifies the initial directory for the P&E debugger.	Click **Browse** to select an initial directory for the P&E debugger. Alternatively, enter the path to the initial directory.

Command-Line Arguments

When the CodeWarrior IDE transfers control to the P&E debugger, the debugger uses the arguments specified in the **Build Extras** target setting panel.

Table B.3 explains the P&E debugger command-line arguments:

Table B.3 P&E Debugger Command-Line Arguments

Argument	Description
FILETOLOAD	Optional. Specifies the full path to the debug/object files to load at startup. If you are using FILETOLOAD argument, then this argument must be the first argument. In addition, you must include either the LOADTORAM or LOADTOFLASH argument in the command line.
LOADTOFLASH	Optional. Loads the debug/object files into the microcontroller's flash memory. You can use the FALGORITHM and the FBASEADDR arguments to select an appropriate flash programming algorithm and the base address. If you do not specify either the FALGORITHM or the FBASEADDR argument, the debugger uses settings from the previous load. If the debug/object file is loading for the first time, the debugger loads a default algorithm based on the detected microcontroller.
LOADTORAM	Optional. Loads the debug/object files into the microcontroller's RAM memory.

Table B.3 P&E Debugger Command-Line Arguments

Argument	Description
LOADGOTILMAIN	Optional. If you are using the LOADGOTILMAIN argument with the FILETOLOAD argument, the LOADGOTILMAIN argument instructs the debugger to run the microcontroller until it reaches the main() function in the source code. The debugger runs code, if there is a debug label called main.
FALGORITHM "*ALGORITHMPATH*"	Optional. Specifies the flash programming algorithm file to be used if LOADTOFLASH is set.
FBASEADDR "*BASEADDR*"	Optional. Specifies the base address for FLASH programming.
-SCRIPTFILE "*MACROFILEPATH*"	Optional. Specifies a startup macro file to run each time the ICD starts up.
-RESETFILE "*RESETFILEPATH*"	Optional. Specifies a reset macro file to run each time the microcontroller is reset.
-RESETFILEOFF	Optional. Disables running the reset macro file.
-SOURCEPATH "*sourcepath1;sourcepath2…*"	Optional. Specifies the directories that contain the source files. The debugger uses this argument to find the source code files to display.

Connecting the Hardware

Follow these steps to connect the P&E USB-ML-PPCNEXUS, USB-ML-UNIVERSAL, or Cyclone MAX debugger hardware to your host and target systems:

1. Make sure the target system power is off.
2. Disconnect the debug cable from the target system.
3. Connect the P&E hardware and the debug cable to the host system.
4. On the host system, from the CodeWarrior IDE, select **Project > Debug** to start the P&E ICDPPCNEXUS software.
5. Reconnect the debug cable to the target system.
6. Switch on the target system power.

To disconnect the P&E hardware, follow these steps:

1. Switch off the target system power.
2. Disconnect the debug cable from the target system.

For more information, refer to P&E debugger documentation.

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W